KNJIGA O OPERATIVNIM SUSTAVIMA I RAČUNALIMA, RAČUNALNIM MREŽAMA, VIRTUALIZACIJI I LINUX KONTEJNERIMA

HRVOJE HORVAT

# OPERATIVNI SUSTAVI I RAČUNALNE MREŽE LINUX U PRIMJENI

REVIDIRANO IZDANJE

BESPLATNO PREUZMITE PDF INAČICU KNJIGE



Ova inačica knjige je revidirano izdanje knjige: ***Operativni sustavi i računalne mreže - Linux u primjeni***
**ISBN** broja: **978-953-59438-9-1**.

**Operativni sustavi i računalne mreže: Linux u primjeni – revidirano izdanje**

**Tehnički recenzenti** (po abecednom redu):

- teh. **Alikavazović Bojan**, senior konzultant za računalnu sigurnost, *Cisco*: CCNA/CCNP, *Offensive Security*: OSCP/OSC
- ing.el. **Buljubašić Tomislav**, softverski inženjer
- teh. **Crnković Saša**, inženjer računalnih mreža i sustava
- mag.ing.comp **Gregurić Ivan**, inženjer računalnih mreža i sustava
- doc.dr.sc. **Jakopec Tomislav** - Filozofski fakultet Osijek
- dipl.ing.el. **Ljubojević Saša,** inženjer računalnih mreža i sustava
- bacc.ing.comp. **Markanović Tomislav**, računalni programer
- prof.dr.sc. **Martinović Goran** - Fakultet elektrotehnike, računarstva i informacijskih tehnologija
- doc.dr.sc. **Pejić Petra**, mag.ing.comp - Fakultet elektrotehnike, računarstva i informacijskih tehnologija
- dipl.ing.el. **Ragić Igor**, inženjer računalnih sustava i mreža
- dipl.ing.el. **Skokić Darko**, programer *embedded* sustava
- dipl.ing.el. **Stipičević Tonči**, inženjer računalnih mreža i sustava
- mag.ing.geod. et geoinf. **Stojnović Vedran**, geoinformatčar i administrator računalnih sustava
- univ.bacc.ing.comp. **Varga Hrvoje**, računalni programer

**Drugi suradnici:** (po abecednom redu): Bahes Boris, Bakša Saša Stjepan, Bartolović Lukas, Bašić Asmir, Bradarić Demis, Bradarić Kristijan, Brkić Valent, Čagalj Ivan, Gotić Matej, Horvat Marina, Horvat Sandra, Josipović Iwan, Jukić Dinko, Komar Boris, Kovačević Karlo, Lacković Marija, Maković Tomislav, Mamula Stanko, Markotić Marin, Muhar Vanda, Olrom Zvonimir, Papić Krešmir, Petrović Saša, Poslon Miroslav, Trivunović Nenad, Tuco Ferhad, Zušćak Domagoj.

**Naslov**: ***Operativni sustavi i računalne mreže: Linux u primjeni***


**Ilustracije su napravljene u programima otvorenog koda:** *Inkscape* i *GIMP* te *Google Drawings.*
**Grafikoni i dijagrami rađeni su u:** *PlantUML*, *Google Drawings*, *Inkscape* te *LibreOffice*
**Autori ilustracija i fotografija:** Hrvoje Horvat i drugi autori, navedeni uz fotografiju ili ilustraciju
**Broj riječi u knjizi:** 723925
**Broj stranica knjige:** 1425
**Broj ilustracija i slika u knjizi:** 383
**Broj Linux naredbi i programa koji se spominju u knjizi:** 409
**Broj izvora informacija korištenih u knjizi:** 2.583.
**Radna inačica publikacije**: 2.6.15.
**Autor**: Hrvoje Horvat
**Naklada**: vlastita naklada
**Tisak:** elektroničko izdanje (*PDF*)
**Grafičko oblikovanje naslovnice:** Ana Horvat
**Mjesto i godina izdavanja:** Osijek, prosinca 2024. godine.
**Izdavač:** Hrvoje Horvat
**ISBN**: **978–953–59438–2–2**
**DOI:** ***10.5281/zenodo.14582057***

**Najnovija inačica knjige je dostupna na poveznici:** `https://doi.org/10.5281/zenodo.8119310`



# Sadržaj

# 1.Licenca i predgovor

Ova knjiga objavljena je pod licencom:
***Creative commons Attribution-ShareAlike 4.0 International license (CC BY-SA 4.0)***




To ukratko znači da ovu knjigu slobodno možete:



**Dijeliti dalje** — možete umnažati i redistribuirati knjigu na bilo kojem mediju ili formatu.



**Stvarati prerade** — što znači da možete remiksati, mijenjati (transformirati) i prerađivati ovu knjigu, u bilo koju svrhu, pa i komercijalnu.

Pod sljedećim uvjetima:

- **Imenovanje** — Morate adekvatno navesti autora, uvrstiti poveznicu (link) na licencu i naznačiti eventualne izmjene. Možete to učiniti na bilo koji razuman način, ali ne smijete sugerirati da davatelj licence izravno podupire Vas ili Vaše korištenje djela.
- **Dijeliti pod istim uvjetima** — Ako remiksirate, mijenjate ili prerađujete materijal, Vaše prerade morate distribuirati pod istom licencom pod kojom je bio izvornik (konkretno **CC BY-SA 4**).
- **Bez daljnjih ograničenja** — Ne smijete dodavati pravne uvjete ili tehnološke mjere zaštite koji će druge pravno ograničiti da čine ono što im licenca dopušta.

*Ovo je sažetak a ne zamjena za punu licencu!*

**Sve detalje licence CC BY-SA 4 pročitajte na poveznici:**
**https://creativecommons.org/licenses/by-sa/4.0/legalcode.hr**

---

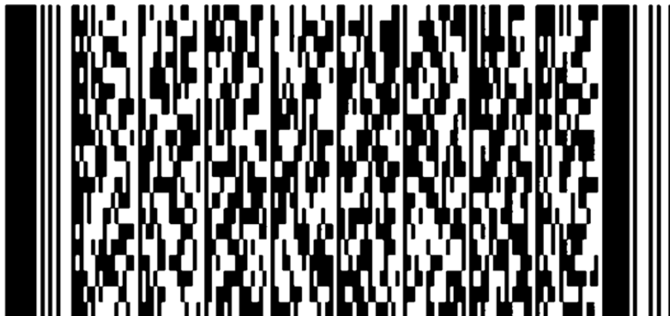
***Trenutna inačica knjige iako je revidirana, ona nije lektorirana, niti su na njoj odrađene urednička/grafička niti bilo koje druge nakladničke ili izdavačke korekcije!***

***Za sve komentare i prijedloge ili željena poboljšanja, kontaktirajte me na:***
***https://www.linkedin.com/in/hrvoje-horvat-48477b1/***

---

**Ako želite donirati autora za izradu, potaknuti na osvježavanje sadržaja ili razvoj novih inačica knjige, dobrovoljne donacije možete izvršiti na IBAN broj (mog) računa:**
`HR96 2360 000 3114 8715 66`
**Pod opis plaćanja upisati:** `Donacija za knjigu ISBN` **`978-953-59438-2-2`**

Ili skenirajte poveznicu s **QR** kodom za donaciju (primjerice **10,00 €**):



**Autor:** Hrvoje Horvat U Osijeku, prosinca 2024. godine.

# 1.1. O autoru i recenzentima

## 1.1.1. O autoru



Hrvoje Horvat se prvi puta susreće s računalima 1984. godine, s jednim od najpoznatijih računala: ***ZX Spectrum***, kao prvim osobnim računalom, pod čijim utjecajem i ulazi u svijet računala. Kasnijih godina prelazi na modernije tehnologije i konačno na prvo PC računalo te odrađuje prvo umrežavanje s kolegama u naselju. Sredinom 1990-tih po prvi puta upotrebljava GNU/Linux za prvi poslužitelj, te mu dalji životni odabir struke i zanimanja postaju i ostaju u svijetu IT-a. Uslijedili su honorarni poslovi umrežavanja i održavanja računala i IT sustava. Od 2000.g. radi u tvrtki ***Siemens***, potom u ***Siemens Convergence Creators*** zatim u tvrtki ***ATOS***. Zatim radi u tvrtki ***ENEA Software***, *a* od 2022. godine prelazi u tvrtku ***Ericsson Nikola Tesla***.
U dugom nîzu godina rada prošao je razna školovanja vani i unutar tvrtki u kojima je radio. Prvih godina radio je na razvoju i održavanju IT infrastrukture za korporativnu mrežu tvrtke *Siemens*, koja je tada imala oko 350.000 računala i poslužitelja. Tu je bilo rada s *Windows* (*Server*) poslužiteljima, *Active Directory*-jem i sličnih tehnologija te *Unix* poslužitelja; od ***HP UIX*** do ***Sun Solarisa*** i **GNU/*Linuxa.*** Sa ***Gnu/Linuxom*** od tada radi i danas.

Bilo je tu i sigurnosnih tema poput implementacije Cisco *VMPS*-a a kasnije i *802.1x*, izolacija unutarnjih mreža (testnih laboratorija) s raznim vatrozidima. Postavljanja *Proxy* poslužitelja za korporativnu mrežu te sličnih tema i projekata.
Uslijedili su mnogi projekti iz područja NAS/SAN sustava i virtualizacije, na raznim platformama, kao i naprednih mrežnih tehnologija (~ 2009. – 2012.g.), a koji su se ponavljali i proširivali sve do danas. Neke od virtualizacijskih platformi postavljenih tada još uvijek besprijekorno rade i održavaju se i danas.

Nakon nekog vremena krenuo je s raznim predavanjima iz područja računalnih mreža i mrežnih tehnologija. Potom je prešao na rad za projekte koje je radio za Telekome širom svijeta. Zbog stalne želje za učenjem došla je i ideja pokretanja udruge (***Udruga01***: 2010-2012) u kojoj su se okupljali stručnjaci iz područja IT mreža i "poslužiteljskih tehnologija".
Tijekom aktivnog učenja u Udruzi prošli su na desetke naprednih mrežnih protokola i sustava.
Ovo je bilo vrlo intenzivno iskustvo, kako vremenski tako i intelektualno, ali su znanja koja je produbio ili stekao, stvarno neizmjerna.

U *Siemens*u, nove službene teme (2013.g.) su bile **AAA** i **PCRF** servisi, na ***UNIX*** i **GNU/*Linux*** platformama, od standardnih do *Blade* poslužitelja. Ovdje je bilo i zanimljivih i naprednih mrežnih tehnologija i sustava (usmjerivača, *Load Balancera*, naprednih višeslojnih preklopnika,…). Redundancija i sustavi visoke dostupnosti su se naravno podrazumijevali na svakoj razini, od svakog mrežnog elementa i uređaja, poslužitelja do svih pojedinih mrežnih protokola.

Krajem 2014 okupila se i ekipa željna učenja i stjecanja novih znanja iz područja računalne sigurnosti. Pokrenuli su nîz radionica pod nazivom "*Offensive & Defensive security*" koje su pokrivale sve od osnova, enumeracije, mrežne sigurnosti (od mrežnih protokola do mrežnih tehnologija) sve do sigurnosti na razini operacijskog sustava. Ovih nekoliko mjeseci vrlo intenzivnog učenja na stvarnim primjerima i na "živoj" opremi donijelo mu je prilično drugačiji pogled i svijest o IT sigurnosti.
Definitivno posebno iskustvo, bez obzira na žestoki tempo učenja i rada nakon posla, do kasnih noćnih sati, danima, tjednima i na kraju mjesecima. Zahvaljuje se, svima uključenima u projekt, a poglavito mentoru iz područja informatičke sigurnosti: Bojanu Alikavazoviću te Kristijanu Bradariću, kao suorganizatoru i aktivnom članu.

Tijekom 2014/2015.g. prelazi na novo područje "*Cloud*" tehnologija i sustava s naglaskom na redundanciju i visoku dostupnost svih elemenata sustava: od virtualnih računala, mrežnih komponenti, pa sve do razine aplikacija. Bilo je ovo vrlo zanimljivo razdoblje u kojemu je usluge „u oblaku" dovodio do krajnjih granica. U ovom pilot projektu je cilj bio postići minimalnu dostupnost svake pojedine komponente u "Cloud" sustavu od minimalno 99.99% što je postavilo vrlo visoku ljestvicu. Doduše prvi zahtjevi su bili 99.995%, ali se ipak dobilo odobrenje da se minimalna granica za potrebe testiranja može spustiti na samo 99.99%. Dalje testiranje njihovih granica i mogućnosti te raznih scenarija potrajalo je oko godinu dana.
Neizmjerna iskustva i poliranje raznih mrežnih protokola i servisa su bili neizbježni.

Paralelno s ovim projektima radio je i na projektu virtualizacije upotrebom ***Proxmox VE*** sustava za virtualizaciju i Linux kontejnere u kombinaciji s klasterskim redundantnim distribuiranima sustavom za pohranu (NAS/SAN sustav) ***CEPH***.
Znanja iz mnogih područja su se također još više izbrusila.

Potom se ponovno profesionalno bavi virtualizacijskim temama na platformama: ***VMware***, ***OpenStack*** i ***Proxmox VE*** te na novim projektima i NAS/SAN sustavima za pohranu podataka, kao i sličnim tehnologijama. Od 2017. godine radi na tehničkoj podršci 3/4 razine (***TAC3/4***) za razne proizvode i rješenja tvrtke *Siemens CVC*. Prelaskom u tvrtku *ATOS* radi kao ekspert za GNU/linux i računalne mreže te nastavlja rad na tehničkoj podršci 3/4 razine (***TAC3/4***) za razne proizvode i rješenja tvrtke *ATOS* razvijene za telekome.

Zatim prelazi u tvrtku ***ENEA Software***, ali nastavlja s poslovima koji obuhvaćaju ***TAC3/4*** poslove za telekome širom svijeta (*Linux, virtualizacija, networking, AAA, PCRF, …*).

Prelaskom u tvrtku ***Ericsson Nikola Tesla*** radi kao mrežni inženjer za ***IoT*** sustave tvrtke *Ericsson*.
Navedeni ***IoT*** sustav trenutno koristi oko 9.000 tvrtki, a na sustav je spojeno oko 95.000.000+ aktivnih uređaja.

Osim školovanja koja je prošao, održavao je i redovito održava stručna predavanja unutar tvrtki u kojima je radio i radi.
U slobodno vrijeme volonterski radi za inicijativu ***Open Source Osijek*** te udrugu ***Osijek Software City*** za koje održava mnoga predavanja i radionice. Piše stručne članke te objavljuje nekoliko kratkih knjižica pod ***GPL*** licencom otvorenog kôda.
***GPL*** licenca otvorenog kôda znači da su objavljene radne knjižice javno dostupne na korištenje, mijenjanje; ispravljanje i korekcije, kao i proširivanje te dalju distribuciju, na što i poziva sve zainteresirane.
Knjižice koje su ovdje objavljene predstavljaju kostur pojedine priče, koji je potrebno (s vaše strane) proširiti.
Navedene knjižice su dostupne na sljedećim poveznicama:

- *Kratka povijest UNIXa: od UNICSa do FreeBSDa i Linuxa (**ISBN: 978-953-59438-1-5**).*
- *Kratka priča o NAS i SAN sustavima (**ISBN: 978-953-59438-4-6**).*
- *Kratka priča o mrežama – preklopnici i usmjerivači (**ISBN: 978-953-59438-5-3**)*.

---

### 1.1.2. Zahvala

Zahvaljujem se svima koji su mi pomogli svojom podrškom i znanjem. Prvo bih trebao spomenuti svoju obitelj, jer uvijek sve kreće od obitelji. Podržavali su od malih nogu moj istraživački duh, znatiželju i propitkivanje. Nije uvijek sve u talentu, ponekad je potrebno imati ljude koji će prepoznati u kojem smjeru trebaš ići i podržati te u tome. Zahvala i mojim dragim prijateljima s kojima sam također od ranog djetinjstva bio i ostao u IT svijetu sa zajedničkim istraživanjem, isprobavanjem i testiranjem svih mogućih i nemogućih tehnika i tehnologija.

Ne mogu ne spomenuti i sve ostale ljude koje sam upoznao na tom putu i sve prave prijatelje koji nisu iz IT svijeta, ali su također imali svoju ulogu te su promijenili moj način razmišljanja i pogled na svijet. Tu bih ubrojio i svoju suprugu Marinu koja mi je uvijek bila najveća podrška. Ponekada, ako ne i često, potrebno je promijeniti perspektivu kako bi se bolje sagledale mnoge stvari u osobnom i profesionalnom razvoju i životu.

I na kraju, zahvaljujem se svim volonterima s kojima sam u suradnji i pokrenuo inicijativu ***Open Source Osijek***, kao i one koji su nas prihvatili i omogućili nam predavanja i radionice te prihvatili ideju širenja znanja na volonterskoj bazi.
Zahvaljujem se i udruzi ***Osijek Software City*** kao jednim od pokretača ovih promjena u Osijeku (i šire).

**O pisanju knjige te objavljivanju pod licencom otvorenog koda**
Iako sam gotovo od početka pripremanja volonterskih predavanja i radionica te pisanja priprema koje su prerasle u knjigu imao sve veće poteškoće s vidom, nisam dopustio da me to omete ili obeshrabri. Štoviše, oko 70% prvog izdanja knjige inicijalno sam pisao na stranici www.opensource-osijek.org unutar *DokuWiki* alata pa sam ispravke i dodatke najčešće radio na mobitelu ili gdje sam već stigao (*gdje ima volje ima i načina*).
Kasnije sam sve prebacio u drugi format i na njemu nastavio raditi za računalom.

Jedan od limitirajućih faktora koji me je spriječio da knjigu objavim i ranije je bilo upravo moje stanje koje mi nije dopuštalo sate i sate čitanja, istraživanja i pisanja, već dnevno vrlo ograničeno vrijeme za to. S druge strane nisam se htio niti vremenski ograničiti i brzati znajući da ću sigurno propustiti nešto važno, radeći prema Japanskoj filozofiji prema kojoj je važno težiti kvaliteti. Podsjećajući se na pitanje i odgovor: ***kada će biti gotovo: upravo onda kada bude (spremno)!***
Naime, ipak je potrebno neko vrijeme da se slegne napisana materija te da se iznova (i iznova) pregleda i eventualno dopuni.

Stoga mi je duže vrijeme pisanja osim vremenskog, dalo i dovoljan intelektualni odmak i mogućnost sagledavanja iz drugog kuta, koji sigurno ne bih imao da sam bio ograničen s vremenom (ili nestrpljivošću).
I na kraju, s obzirom na promicanje ideja i principa razvoja programa i sustava prema principima otvorenog koda i dijeljenja znanja, odlučio sam se i ovu knjigu objaviti prema istom principu; slobodnom dijeljenju znanja i ideja, prema licenci
**CC BY-SA 4** detaljnije opisanoj u prethodnom poglavlju.

***Knjiga je ponovno revidirana tijekom 2024. godine!***

“*Budi promjena koju želiš vidjeti u svijetu!* ”
***Mohandas Karamchand Gandhi (Mahatma Gandhi)***

*Hrvoje Horvat*

---

### 1.1.3. O recenzentima

**Dodati** (budući recenzenti)

---

# 1.2. Predgovor

Vrlo brzo nakon osnutka inicijative ***Open Source Osijek (~2013.g.)***, nastala je ideja o pokretanju predavanja i radionica iz područja informacijskih tehnologija. Tada je bilo vrlo jasno kako netko treba popuniti mnoge praznine u znanjima ljudi koji se trenutno nalaze na tržištu rada, kao i onih koji se zapošljavaju u informatičkim tvrtkama. Stoga smo odlučili krenuti prvo s predavanjima i radionicama o *Linuxu*. U trenutku nastanka ideje o pokretanju predavanja „*Uvod u Linux*", sredinom 2013. godine, nastala je i ideja da bi sva predavanja mogli pretočiti u knjižicu, koja će rasti i razvijati se sa svakim novim predavanjem i radionicom. Svako poglavlje pratili su i mnogi primjeri. Po završetku svakog predavanja dodavana su i napredna poglavlja, koja nismo pokrili u samom predavanju, ali smatram kako će biti dobrodošla za one koji žele naučiti nešto više. Napredna poglavlja također prate praktično upotrebljivi primjeri, koji su odabrani, jer sam ih i sam koristio, a do kojih sam došao, nakon dugog niza godina upotrebe Linuxa u praksi. Naime smatram kako su konkretni primjeri iz prakse najbolji za ukazivanje na mogućnosti koje svatko treba isprobati i na koje se treba fokusirati. Nadalje, teorija i prateći primjeri mogu vam pomoći u raznim problematičnim situacijama ili u slučajevima kada želite optimirati određene podsustave Linuxa.
Za one koji tek ulaze u svijet Linuxa, napredna poglavlja mogu slobodno preskočiti, dok ne savladaju osnove. Nakon završetka zadnjeg predavanja, dodavana su i nova poglavlja i popunjena postojeća, sa što konkretnijim primjerima iz prakse. Kako smo se kasnije spojili s udrugom ***Osijek Software City***, naše ideje o stručnim, volonterskim predavanjima i radionicama, prerasle su u ***Code Camp*** grupu koja i danas aktivno djeluje unutar ***Osijek Software Cityja,*** *a* koja je na godišnjoj razini konstantno nadilazila tisuću polaznika. Dodatno, nakon još pokojeg predavanja i radionice, zbog mnogih pitanja i uočenih problema u radu s polaznicima, cijela knjiga je još jednom promijenila redoslijed poglavlja kao i svoj format. Nadalje, neka poglavlja su i dodana, a neka druga znatno proširena. Kasnijom suradnjom s Fakultetom elektrotehnike, računarstva i informacijskih tehnologija Osijek (***FERIT***) knjiga je uobličena te prilagođena i za predavanja.
Kasnijim usklađivanjem s tehničkim i drugim suradnicima, knjiga je dodatno prilagođena za učenje, ali i za predavanje.
S obzirom na stalni rad i usavršavanje u praksi, kao i mnogim kompleksnim i manje kompleksnim, ali svejedno zanimljivim slučajevima koje sam prošao u praksi, odlučio sam da se i oni uvrste u knjigu. Stoga je pred vama proširena i izmijenjena inačica knjige, novog naslova: ***Operativni sustavi i računalne mreže: Linux u primijeni***.

***Knjiga je ponovno revidirana i djelomično proširena tijekom 2024. godine.***



***Svi primjeri i savjeti su preporučeni za testnu upotrebu ili za učenje. U slučaju da primjere ili savjete primjenjujete na "žive" sustave odnosno sustave koji su u radnoj ili produkcijskoj upotrebi, odgovornost u slučaju neke greške ili kvara je na vama samima te se autor odriče svake odgovornosti za štetu nastalu vašim radom.***



*Navedene tehnologije, komponente, sustavi ili mehanizmi Linuxa, provjereni su, iz više različitih izvora. Svi primjeri su također testirani. Bilo je slučajeva u kojima su informacije iz desetak ili više izvora govorile jedno, a logika i dublje poznavanje sustava, nešto drugo. Tada sam se vodio Sokratovom metodom kritičkog razmišljanja. Naime s vremena na vrijeme događa se da naizgled točnu i logičnu izjavu jednog autora, mnogi drugi autori (pre)uzimaju kao činjenicu, ne promišljajući ili ne poznajući materiju dovoljno duboko ili široko. Bilo je i slučajeva u kojima je bio podjednak broj dijametralno različitih izjava, kao i lakših slučajeva u kojima je većina bila u pravu. Stoga sam takve izjave ili objašnjenja ponovnom dubljom analizom i testovima pokušao dokazati ili pobiti na testnom Linux sustavu.*
*Na kraju, ovaj proces je bio ponekad mukotrpan, naporan i dugotrajan, ali je rezultat ovdje, pred vama.*

# Za koga je ova knjiga i što sve pokriva ?

Ova knjiga obrađuje osnove rada u tekstualnom okruženju Linuxa. Namijenjena je svima koji imaju vrlo malo ili niti malo iskustva s Linuxom. Dodatno gotovo svaka važnija cjelina sadrži napredna poglavlja, tako da knjiga može biti od koristi i naprednim korisnicima u cilju detaljnijeg razumijevanja ili optimizacije Linux sustava. Nadalje, mnoga od naprednih poglavlja dodatno su proširena, za one koji traže i ekspertne detalje i primjere.
Na kraju je knjiga proširena kako bi pokrila i materiju koju obuhvaćaju neki od najcjenjenijih Linux certifikata. Stoga se knjiga može koristiti i kao priprema za učenje za **Red Hat®** certifikate poput: *Red Hat® Certified System Administrator* (**RHCSA**®), kao i za ***Linux professional Institute***® certifikate (***LPIC-1/2/3***®) ili **Linux Foundation**® certifikate **LFCA®** ili **LFCS®**.

Ako pogledamo logičke cjeline ove knjige, koje se tiču vršnih podsustava Linuxa, ona pojašnjavajući ove podsustave objašnjavaju i načine i principe rada bilo kojeg operativnog sustava, jer se svi oni sastoje od identičnih komponenti kao što su:

1. Virtualni datotečni sustav (***Virtual Filesystem***) odnosno diskovni i datotečni podsustav, koji je zadužen za pohranjivanje i rad s datotekama i direktorijima (mapama/folderima), kao i ovlastima, odnosno njihovim pravima, te u konačnici načinu i metodama pristupa i zapisivanja diskovnim kontrolerima. Te u konačnici i krajnjim diskovnim medijima: tvrdi ili SSD disk, CD/DVD-ROM, USB disk i slično.
2. Memorijski menadžment (***Memory Management***) koji je zadužen za rad sa sustavom virtualne memorije, odnosno u konačnici s memorijskim kontrolerom i RAM memorijom u koju se učitavaju i iz koje se čitaju i privremeno spremaju podaci. Naravno u konačnici se u RAM memoriju učitavaju i svi pokrenuti programi (aplikacije).
3. Sustav za komunikaciju između programa odnosno *procesa* (***IPC – Inter Proces Manager/Communication***) pomoću kojega pokrenuti programi međusobno komuniciraju.
4. Mrežni podsustav (***Virtual Network system***) koji je zadužen za svu mrežnu komunikaciju između programa, kako na lokalnom računalu tako i preko lokalne mreže, ali i interneta.
5. Proces odnosno program menadžment (***Proces Management/Scheduler***) koji je zadužen za cijeli životni ciklus svakog programa. Dakle on je odgovoran za svaki program: od njegovog pokretanja te dodjeljivanja i korištenja svih resursa računala, pomoću navedenih podsustava (1, 2, 3 i 4), preko trenutaka kada je potrebno osloboditi trenutno neiskorištene resurse ili na kraju zatvoriti program te osloboditi sve njegove resurse (CPU, disk, RAM, mreža, …).

Ovu knjigu je moguće koristiti i za praktično učenje o principima rada operativnih sustava općenito, s time da govorimo o Linuxu i nudimo primjere iz Linuxa, kroz koje je zbog njegove otvorenosti znatno bolje upoznati svaki operativni sustav i njegove mehanizme. U poglavljima u kojima govorimo o naredbenom retku (engl. *shell*) i *shell* skriptama, naučit ćete i osnove programiranja, jer ovdje ćemo se susresti s:

- Varijablama i funkcijama.
- Petljama i uvjetima.
- Ali i drugim elementima osnova programiranja.

Nadalje, ova knjiga nas upoznaje i s građom i načinom rada računala te osnovnim komponentama računala.
U konačnici možemo reći kako ovu knjigu mogu koristiti studenti, za učenje principa rada operativnih sustava, osobe koje se tek žele upoznati s Linuxom, kao i oni koji žele produbiti svoja znanja iz Linuxa ili čak stručnjaci, koji se povremeno žele prisjetiti nekih naprednih cjelina *Linuxa*. Dodatno, svi koji se žele upoznati s osnovama računalnih mreža i detaljima TCP/IP protokola, kao i oni koje zanimaju konfiguracijski i ekspertni detalji te moguće optimizacije na razini mrežnog diskovnog ili nekog drugog podsustava operativnog sustava, naći će ovu knjigu korisnom. Knjiga kreće s primjerima upotrebe raznih korisnih programa (naredbi) koji su specifični za cjelinu o kojoj se govori te se proširuje s teorijom i naprednijim primjerima. Ovo je bila nît vodilja za većinu poglavlja u kojima je taj redoslijed ili način rada bio primjenjiv. I na samom kraju, dodana su poglavlja u kojima se upoznajemo s virtualizacijom i Linux kontejnerima te sigurnosnim tehnologijama i napomenama; od teorije do praktičnih primjera u Linuxu.

# 2.Uvod

U informatici su svi ili barem velika većina pojmova na globalnoj razini svima razumljivi na **engleskom** jeziku.
Dodatno, nazivi tehnologija, alata i pojmova osmišljeni su i u širokoj upotrebi također na **engleskom** jeziku. Stoga je naglasak autora kako će se uz svaki hrvatski pojam OBAVEZNO često ponavljati i izvorni pojam na **engleskom** jeziku, toliko puta koliko će biti potrebno da ga upamtite; namjerno, više puta nego bi to bilo očekivano.
Važni dijelovi teksta su pisani **zadebljano**.
Primjeri naredbi odnosno vježbi, pisani su s posebnom sivom pozadinom, drugim fontom i to podebljano, poput naredbe:

```
ls -al
```

Ista naredba unutar teksta je posebno obojana, poput naredbe: `ls -al`.
Ispisi naredbi su vidljivi u istoj razini, poput ispisa naredbe: `ls -al` koja je podebljana, a njen ispis koji slijedi nije.

```
ls -al

total 80
drwx------  6 root root  4096 Mar  2 17:59 .
drwxr-xr-x 26 root Svi   4096 Feb 23 16:58 ..
-rw-------  1 root root 10046 Mar  2 17:59 .bash_history
drwx------  3 root root  4096 Nov  7 19:50 .cache
drwx------  4 root root  4096 Jan 27 18:43 .config
-rw-r--r--  1 root root   148 Aug 17  2015 .profile
```

Važne napomene su vidljive u nekoliko razina:

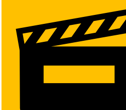

Osnovne informacije


Poveznice na druga poglavlja


Važne napomene


Upozorenja


Brze upute ili napomene


Animacije

Izvori informacija su navedeni s pripadajućim rednim brojevima izvora, poput: **(124)**, **(125)**.
Pri tome su poveznice za **PDF** knjigu direktno dostupne, te su dodatno, za navedene redne brojeve izvora informacija, navedene na kraju knjige u poglavlju: **31. Izvori informacija.**

*Wikipedija izvore koristite samo kao prvu polaznu točku u svom istraživanju konzultirajući izvore koji su navedeni na konkretnom Wikipedija članku.*

Nadalje u svakoj cjelini u kojoj se prvi puta spominje neka naredba, ta naredba je navedena s poveznicom na internet adresu s detaljnim uputama naredbe, poput primjerice naredbe: `ls`.
Kako bismo mogli razgraničiti osnove od naprednih dijelova knjige, sve napredne cjeline ili poglavlja su tako i označena.

# 2.1. Na kojoj distribuciji Linuxa ćemo raditi ?

Operativni sustav ***GNU/Linux*** sastoji se, kao i drugi operativni sustavi, od jezgre (tzv. *kernela*), sistemskog softvera, servisa, programskih biblioteka, dodatnog softvera i menadžmenta softverskih paketa te nekih drugih komponenti. Ovakav skup svih navedenih komponenti koji zajedno čine operativni sustav, u Linux svijetu se naziva distribucija Linuxa (o njoj kasnije).
Tvrtka ***Red Hat*** je jedna od vodećih tvrtki koje ulažu u razvoj Linux *kernela* (jezgre ***GNU/Linux*** operativnog sustava) i povezanih tehnologija u široj zajednici otvorenog koda. Inženjeri *Red Hata* dodatno pomažu u poboljšanju značajki, pouzdanosti i sigurnosti Linux sustava. Nadalje, tvrtka *Red Hat* razvija i održava svoju distribuciju *GNU/Linuxa* pod nazivom ***Red Hat Enterprise Linux*** koji pruža stabilnu i pouzdanu platformu, što je posebno korisno za poslovna okruženja gdje su vrijeme rada i pouzdanost sustava najvažniji. Naime njihova svaka vršna inačica sustava (pr. 9.x) uvijek se drži točno određene vršne dugoročno podržane inačice kernela (konkretno 5.14.x) koja se drži zaključanih značajki, samo uz sigurnosne i performansne nadogradnje. Isto je i s pripadajućim programima i drugim komponentama sustava, što jamči dugoročno siguran i pouzdan rad vaših aplikacija i servisa. S tisućama certificiranih hardverskih, softverskih i rješenja u oblaku, *Red Hat* je vodeći svjetski pružatelj *open source* IT rješenja za poduzeća te ima povjerenje 90% **Fortune 500** tvrtki odnosno najvećih tvrtki na svijetu.

Nadalje, tvrtka *Red Hat* sa svojim operativnim sustavom ***Red Hat Enterprise Linux*** drži oko **33%** tržišta operativnih sustava za koje se plaća licenca; pri tome ***Microsoft*** drži oko 47%, dok svi ostali odnosno sve druge komercijalne distribucije Linuxa i UNIX-a te drugih operativnih sustava, drže oko 18% tržišta. Što se podrške tiče, veliki igrač na tržištu je i ***Oracle Linux*** koji je također baziran na ***Red Hat Enterprise Linuxu*** i 100% je kompatibilan s njim, a koji također nudi komercijalnu podršku te ima veliku zajednicu ljudi, kao i veliki broj sustava u upotrebi. Navedeni ***Oracle Linux*** kao i ***CentOS***, ***Fedora***, ***Rocky Linux***, ***Alma Linux*** i neke druge distribucije Linuxa dizajnirane su tako da su binarno kompatibilne s ***Red Hat Enterprise Linuxom***.
Ova kompatibilnost omogućuje administratorima sustava neometanu migraciju između navedenih distribucija Linuxa i ***Red Hat Enterprise Linuxa*** bez problema s kompatibilnošću, pružajući fleksibilnost i kontinuitet u radu.

***Red Hat Enterprise Linux*** se može pohvaliti opsežnim spremištem softverskih paketa (tzv. *repozitorij*), koji obuhvaćaju širok raspon aplikacija, programskih biblioteka, uslužnih programa i alata. Ova sveobuhvatna zbirka pojednostavljuje instalaciju softvera i omogućuje administratorima ispunjavanje različitih zahtjeva prema sustavu. Osim komercijalnog repozitorija za programske pakete koje nudi ***Red Hat Enterprise Linux*** i za druge navedene kompatibilne distribucije Linuxa postoje javno (i besplatno) dostupni repozitoriji s tisućama programskih paketa, dostupnih za instalaciju (i korištenje).

Primarno smo *Red Hat* distribuciju Linuxa odabrali i zbog njihove politike razvoja i podrške od minimalno deset godina za svaku vršnu inačicu operativnog sustava, certificiranog hardvera, softvera i usluga u oblaku te mogućnosti plaćanja podrške u više razina, kao i mnogih drugih komparativnih prednosti u odnosu na druge distribucije Linuxa.

Stoga su svi primjeri koje ćemo spominjati u knjizi, pisani za ***Red Hat*** kompatibilne distribucije Linuxa inačice 6.x., ali navodit ćemo i razlike koje vrijede za *Red Hat/CentOS/Fedora/Rocky/Oracle* inačicu 7.x. ili novijih inačica 8.x. i 9.x.
Vratimo se na ***Rad Hat*** Linux i kompatibilne distribucije Linuxa (poput Rocky/CentOS/Fedora/Oracle) i razlike u odnosu na druge distribucije Linuxa.

***Većina navedenih primjera u knjizi radi i na drugim distribucijama Linuxa uz sljedeće napomene.***

Malo su drugačiji, odnosno s njima se radi na malo drugačiji način, dijelovi koji se tiču:

- Upravljanja sa servisima odnosno takozvanim *daemonima* prema Unix/Linux terminologiji.
- Rada sa softverskim paketima (primjerice ***RPM***).

Navedeno se ne odnosi na sve distribucije Linuxa koje su direktno bazirane na *Red Hat linuxu* poput: *CentOS*, *Rocky*, *Oracle Linux* i *Fedora* distribucija, ali i drugih distribucija Linuxa koji koriste *RPM* softverske pakete, poput *OpenSUSE*, *PCLinuxOS*, *Mandriva* i nekih drugih. Ipak, zbog kompatibilnosti s **LPIC** certifikatima, spomenuli smo i rad s ***Debian*** paketnim menadžerom. Međutim dijelovi koji se tiču:

- Upravljanja i rada sa servisima i procesima, baziranim na takozvanom `init` servisu odnose se na sve grane Linuxa i UNIX-a koje koriste `init` način rada odnosno takozvani **System V** kompatibilni način rada sustava.
- Najnovije generacija ***Red Hat***/***CentOS***/***Rocky/Oracle*** distribucija Linuxa inačice 7.x./8.x/9.x, ali i drugih distribucija Linuxa koje su inicijalizacijski servis `init` zamijenile sa `systemd` servisom i pripadajućim softverskim paketima. To je zato što oni imaju malo drugačiji način upravljanja s procesima i servisima u odnosu na `init`, a koji će također biti objašnjeni. *Systemd* koriste i mnoge druge distribucije Linuxa poput: *Alpine*, *Gentoo*, *Debian*, *Ubuntu*, *SUSE*, … .



*Za potrebe vježbi i učenja imate nekoliko opcija: instalirati Linux direktno na svoje računalo ili ga instalirati kao virtualno računalo; primjerice upotrebom programa **VirtualBox** za virtualizaciju.*



Za detalje o distribucijama Linuxa i otvorenom programskom kôdu, dobro proučite poglavlje:
**2.7. Povijest otvorenog koda (*Open Sourcea*) i Linuxa.**
Za listu svih konfiguracijskih datoteka na Linux sustavu, pogledajte poglavlje:
**29.4. Popis značajnijih datoteka Linuxa.**
Za usporedbu važnijih konfiguracijskih datoteka ovisno o distribuciji Linuxa ili Unixa, pogledajte poglavlje:
**30.3. Usporedba konfiguracijskih datoteka ovisno o distribuciji Linuxa ili Unixa.**

**Izvori informacija: (920),(1499),(1500),(1501).**

## 2.2. Koje su prednosti Linuxa i drugih sustava otvorenog koda

Korištenjem Linuxa i drugih sustava otvorenog programskog kôda (engl. *Open Source*), u odnosu na sustave zatvorenog kôda, koje često nazivamo komercijalnim [što ne mora biti točno] programima ili sustavima, dobivamo sljedeće prednosti:

**Kvaliteta**
Sve komponente Linuxa, počevši od Linux kernela (jezgre) do svih programa, kao i drugih programa i sustava otvorenog kôda, razvijani su od strane stotina ili tisuća programera, a testirane od strane stotina tisuća ili više programera i običnih korisnika. Svi uključeni u razvoj i testiranje, a ovdje govorimo ovisno o programu; i o stotinama tisuća ljudi, kako je praksa pokazala, vrlo brzo odnosno višestruko brže nego kod klasičnog komercijalnog softvera, uočavali su greške i ispravljali ih ili radili na poboljšanjima, optimizacijama (primjerice ubrzanjima) ili dodavanju novih funkcionalnosti. Sve navedeno drastično je povećalo kvalitetu konačnog rješenja. Ovdje možemo citirati tvorca Linux kernela; *Linusa Torvaldsa*:

"*Given enough eyeballs, all bugs are shallow*"

Dakle ako dovoljno veliki broj ljudi gleda u određeni problem ili grešku (*bug*) svi problemi će vrlo brzo biti otkriveni, a u konačnici i riješeni. Ako se već bavite razvojem programa, za početak si postavite samo sljedeća pitanja:

- Koliki postotak ukupnog vremena u razvoju svog programa ili sustava trošite na testiranje u širem smislu riječi?
- Koliko vremena planirate za testnu integraciju, kao i dokumentaciju vašeg programa i sustava?

**Sigurnost**
Sigurnost je usko povezana s paradigmama vezanim za kvalitetu jer kao što veliki broj programera i korisnika koji razvijaju softver povećava kvalitetu, istovremeno otkriva i sigurnosne propuste i ispravlja ih. U praksi se to događa puno brže od komercijalnih rješenja. Naime niti najveće korporacije nemaju na tisuće zaposlenih programera i testera na svakom pojedinom softveru, a pošto je njihov izvorni kôd skriven od javnosti, nitko drugi im ne može puno pomoći, osim ukazati na problem kada se već pojavio i napravio štetu. Ako se već bavite razvojem programa, postavite si i sljedeće pitanje: koliki postotak vremena u razvoju svog programa ili sustava trošite na sigurnost; od faze dizajna, preko implementacije do svih faza testiranja.

**Fleksibilnost**
Komercijalnim proizvođačima programa i sustava u konačnici nije u interesu imati najbolji, najbrži, najsigurniji i najkvalitetniji program ili sustav sa svim značajkama, koje neki korisnici ili grupe korisnika trebaju, jer za to treba imati puno resursa: od programera, testera, hardvera, … . Jedan od problema u razvoju je i fleksibilnost, ali i pitanje koliko je vremena potrebno uložiti u razvoj, a da bi program i/ili sustav bio stvarno fleksibilan. Iza svega navedenog zapravo stoje pitanja koja su specifična za razvoj sustava zatvorenog kôda odnosno proizvođača takvih sustava:

- Kako će vam nametnuti potrebu plaćanja nove inačice programa i sustava, ako vam je postojeći dobar ili čak odličan.
- Kako će vas natjerati da kupujete nadogradnje zbog novih značajki, ali i sigurnosnih nadogradnji ili popravaka nekih (njihovih) grešaka, ali i kako će vas natjerati da kupite novi hardver (računalo), ako je postojeći ili novi program odnosno sustav brz i optimalno radi.

S druge strane sustavi otvorenog kôda ne boluju od ovakvih (i drugih) boljki, te je na vama odluka kada ćete primjerice mijenjati vaš hardver (računalo) jer budimo iskreni: nova inačica programa nije napravljena na brzinu samo da se proda te se kasnije ispravljaju (i često naplaćuju) greške. U ovoj priči vi ste besplatni tester nečijeg programa/sustava, koji još dodatno i plaćate.

Također tijekom pisanja programskog kôda, ako vaš kôd nije fleksibilan u širem smislu riječi, često će vas netko već upozoriti na to i/ili ispraviti vaš loše napisan programski kôd. Ova priča se ne zaustavlja na programima. Možda i niste znali, ali postoje i drugi sustavi ili komponente poput centralnih procesora (engl. *CPU*) koji su razvijeni i objavljeni pod nekom od *otvorenih* licenci, dakle dostupni svima.

Na ovaj konkretan način dizajn procesora je otvoren i svatko može potencijalno otkriti neku manu ili predložiti poboljšanje. Primjeri otvorenog dizajna procesora su:

- ***OpenSPARC*** → https://en.wikipedia.org/wiki/OpenSPARC
- ***OpenPOWER*** → https://en.wikipedia.org/wiki/OpenPOWER_Foundation
- ***RISC-V*** → https://en.wikipedia.org/wiki/RISC-V

Postoji i cijeli nîz projekata otvorenog hardvera; pogledajte stranicu: https://en.wikipedia.org/wiki/Open-source_hardware.

No vratimo se na programe; ako se već bavite razvojem programa, postavite si sljedeće pitanje: koliko je program ili sustav koji razvijate fleksibilan te koliko pazite da je programski kôd moguće ponovno iskoristiti (*engl. Code reuse*).

**Interoperabilnost**
Programi i sustavi otvorenog kôda se u praksi (barem) pokušavaju držati standarda te ne uvode neke svoje (izmišljene) formate datoteka, standarde, protokole i slično. Stoga imate veću vjerojatnost da će program ili sustav otvorenog kôda, koji vi koristite, raditi u kombinaciji s programom i sustavom koji koriste vaši klijenti ili netko drugi. Također je velika vjerojatnost, ako primjerice koristite neki mrežni protokol koji je pisan prema načelima otvorenog kôda, iz jednog izvora ili su dio jednog rješenja ili proizvoda, da će isti raditi s drugim rješenjima ili proizvodima.
Naime ovo je najviše slučaj kod raznih sustava, te mnogih protokola (primjerice mrežnih protokola) i uređaja, ali i kod programa i njihovih međusobno, većinom namjerno ili ne, nekompatibilnih formata datoteka.

**Podrška**
Suprotno mišljenju mnogih, podrška za programe i sustave otvorenog kôda postoji u istoj mjeri kao i za komercijalne programe i sustave. U praksi, podrška za programe i sustave otvorenog koda, a pogotovo za one koje je prihvatila šira zajednica i iza kojih stoji i neka fondacija ili tvrtka, je znatno bolja, kvalitetnija i uslužnija, a sve zbog nekoliko činjenica:

- Podršku za komercijalne programe i sustave pruža vam tvrtka koja je razvila taj sustav, a njeni partneri zaduženi su samo za rješavanje određenih (manjih) problema. Pri tome je kôd samog programa i dalje u vlasništvu i dostupnosti **sâmo** tvrtke vlasnika, pa ovdje često dolazi do razvlačenja odgovornosti te gubljenja vremena za vas kao korisnika njihovog programa ili sustava. Dakle u prebacivanju odgovornosti: tvrtke vlasnika, njihove partnerske tvrtke i Vas.
    - Za sve probleme koji traže angažman tvrtke vlasnika programa/sustava, pogotovo, ako je to inozemna tvrtka, bez obzira ima li predstavništvo u vašoj zemlji, ćete se prema raznim statistikama (i osobnom iskustvu autora) načekati, a u dosta slučajeva, dobiti ćete rješenje problema koji ste prijavili, nerijetko uz cijeli nîz novo nastalih (kreiranih) problema.
- Podršku za programe i sustave otvorenog kôda, uključujući mogućnost promjene samog kôda, što uključuje: popravke grešaka, dodavanje novih značajki, raznih optimizacija prema vašim željama i slično, može vam odraditi lokalna tvrtka (iz vašeg grada/županije/države). Što je još bitnije, tih lokalnih tvrtki može biti i nekoliko, što dovodi do zdrave konkurencije. Tada će se svi boriti da pruže bolju i kvalitetniju uslugu za vas kao korisnika nekog programa i sustava otvorenog kôda, koji vjerojatno već koriste milijuni ljudi širom svijeta i za koji također postoji podrška od strane stotina, ako ne i tisuća drugih tvrtki širom svijeta.
    - Za sve programe i sustave otvorenog kôda podršku možete dobiti i od zajednice (engl. *community*) koja je i razvila određeni program ili sustav, a vrlo često je iz te zajednice izrasla i neka fondacija ili tvrtka kojoj/kojima također možete platiti za još višu (i bržu) razinu podrške.

Pitanje za one koji su i dalje skeptični prema podršci za programe otvorenog kôda i koji bi platili (i plaćaju) "*koliko god treba*": Jeste li platili komercijalnu podršku za neki program ili sustav otvorenog kôda i ostali nezadovoljniji u odnosu na *sustave zatvorenog kôda* tj. takozvane komercijalne sustave ili programe za koje (već) plaćate podršku?
Ako i sami razvijate neki program ili sustav zatvorenog kôda (*close source*), pitajte vaše krajnje korisnike, preko nekog posrednika, kakva je podrška za vaš program ili sustav odnosno koliko su oni stvarno zadovoljni?

**Troškovi**
Troškovi programa i sustava otvorenog kôda u konačnici su nekada i višestruko manji od troškova komercijalnih programa i sustava. Evo zbog čega:

- Većina takozvanih distribucija Linux operativnih sustava je besplatna, ali kao što smo vidjeli, za većinu njih možete plaćati direktnu podršku i to često zakladi [*Foundation*] ili tvrtki koja obično stoji iza njih ili naravno nekoj lokalnoj tvrtki. U slučaju kada za kompletnu podršku plaćate lokalnu tvrtku, novac ostaje u vašem gradu/županiji/državi, od poreza, prireza, plaća radnicima te tvrtke i slično. Dakle podupirete ih direktno.
- Za distribucije Linuxa za koje se plaća podrška (primjerice ***Red Hat***, ***SuSe***, …) cijena je često manja od drugih komercijalnih operativnih sustava, a uz tu cijenu dobivate i određenu osnovnu podršku, koja je prema mnogima [i autoru] puno bolja, brža i kvalitetnija od onih "*drugih*". Osim toga možete nadoplatiti i za dodatnu razinu podrške ili jednostavno kupiti samo distribuciju Linuxa uz osnovnu podršku, a sve ostalo: podršku, razvoj i opcije, platiti nekoj lokalnoj tvrtki.
- Drugi programi koji su otvorenog kôda i *besplatni* za osobnu i/ili komercijalnu upotrebu, za sve njih također možete plaćati podršku, prema sličnom modelu kao i za distribucije Linuxa.
- Troškovi potencijalnog školovanja ljudi u vašoj tvrtki zbog prelaska na programe i sustave otvorenog kôda i cjelokupne migracije na sustave otvorenog koda: kada se sve zbroji moguće je nekoliko scenarija uz određene varijacije:
    - Troškove školovanja i migracije plaćate ili odrađujete sâmi, početni troškovi su manji, a dugoročni troškovi su višestruko manji.
    - Troškove školovanja i migracije plaćate (lokalnoj tvrtki), početni troškovi su isti ili približno isti, a dugoročni troškovi su višestruko manji.
    - Troškove školovanja i migracije plaćate (lokalnoj i/ili inozemnoj tvrtki), početni troškovi su isti ili zanemarivo veći, a dugoročni troškovi su manji.

**Izvori informacija: (1752),(1753),(K-19).**

# 2.3. Tko sve koristi Linux i sustave otvorenog koda

Nakon, po nekima sâmo teorije o prednostima upotrebe Linuxa i drugih sustava otvorenog kôda, pogledajmo primjere upotrebe u praksi. Sâmo neke od statistika govore sljedeće: 90% usluga u javnom oblaku koristi Linux, a 62% ugrađenih sustava (*embedded*) također koristi Linux. Preko 60% svih web poslužitelja na svijetu je otvorenog kôda; konkretno su to: nginx i Apache. Svih 500 najsnažnijih superračunala na svijetu pokreće Linux i tako dalje.
Osim toga sve veći broj država i državnih agencija od Sjedinjenih Američkih Država, preko Europe do Japana, sve više koristi Linux kao i razne druge programe i sustave razvijene na principima otvorenog kôda.

Ne zaboravimo i sve veći broj tvrtki; od najvećih korporacija do malih tvrtki sa svega nekoliko zaposlenih. Krenimo od upotrebe Linuxa na mjestima koja će vas možda i začuditi.

**Državne agencije i ministarstva**:

- ***CERN** (European Organization for Nuclear Research)*, ***Fermilab** (Fermi National Accelerator Laboratory)*, …
- Češka: pošta – koriste *SuSe* distribuciju Linuxa s 4.000 poslužitelja i 12.000 računala.
- Europska svemirska agencija (***ESA***) – koristi razne sustave otvorenog kôda.
- Francuska:
  - Parlament: koriste *Ubuntu* distribuciju Linuxa s *OpenOffice* paketom, instalirano je ~1.100 računala.
  - Policija: koriste modificiranu *Ubuntu* distribuciju Linuxa imena *GendBuntu*, na 90.000 računala.
  - Ministarstvo pravosuđa: koristi GNU/Linux i LibreOffice.
- Hrvatska akademska i istraživačka mreža (***CARNET***) koristi *Debian* distribuciju Linuxa na svojim poslužiteljima.
- Japan:
  - Japanska agencija za tehnologiju – koristi alate otvorenog kôda za e-upravu.
  - Japanske željeznice: vlakovi tzv. *Bullet Trains* tj. *Shinkansen* za sustav kontrole i nadzora vlakova te cijelog sustava koriste Linux, … .
- Južna Koreja, Ministarstvo unutarnjih poslova koristi Linux u svojoj IT infrastrukturi.
- Meksiko – koristi alate otvorenog kôda u raznim državnim institucijama i agencijama (pr. Ministarstvo financija).
- Njemačka:
  - Grad Minhen, migrirao je oko 15.000 stolnih računala *s Microsoft Windowsa* na Linux[(kontroverza)]
  - Grad Hamburg je također usvojio rješenja otvorenog kôda, koristeći OpenOffice i druge alate otvorenog koda u upravi.
  - Njemačka savezna agencija za mrežne usluge (*Bundesnetzagentur*) koristi Linux poslužitelje za upravljanje infrastrukturnim mrežama.
  - Njemački parlament koristi rješenja otvorenog kôda za dijelove svoje informatičke infrastrukture.
  - Njemački nacionalni sustav za zapošljavanje (*Bundesagentur für Arbeit*). Koriste *OpenSuSe* distribuciju Linuxa na ~13.000 računala.
- Nizozemska:
  - Sustav za Ministarstvo unutarnjih poslova (*Internet Research and Investigation Network* (***iRN***)). Koriste *Ubuntu* Linux na ~2.200 računala.
  - Nizozemska vlada: koriste razne sustave otvorenog koda.
- **NASA** (*National Aeronautics and Space Administration*), od poslužitelja i stolnih računala do drugih sustava. Primjerice *Space Shuttle* misija *STS-83* je koristila Linux za kontrolu eksperimenata, a primjerice helikopterski dron na Marsu[(2021.g.)] koristi Linux.
- Poljska vlada – koristi alate otvorenog kôda u e-upravi.
- Svjetska zdravstvena organizacija (***WHO***) – koristi softver otvorenog kôda za analizu podataka.
- Sjedinjene Američke Države (SAD):
  - Ministarstvo obrane Sjedinjenih Američkih Država (*U.S. Department of Defense*):
    - Koriste *Red Hat* distribuciju Linuxa, prema nekim izvorima ovdje je za *Red Hat* Linux najveći broj instaliran računala unutar jedne tvrtke ili institucije.
    - Mornarica Sjedinjenih Američkih Država: Flota nuklearnih podmornica (*U.S. Navy Submarine Fleet*): poznato je da koriste Linux baziran na *Red Hat* distribuciji Linuxa.
  - Federalni sudovi (U.S. *Federal Courts*) u Sjedinjenim Američkim Državama.
  - Savezna uprava za civilno zrakoplovstvo (*Federal Aviation Administration*) Sjedinjenih Američkih Država.
  - San Francisco - sustav napredne kontrole prometa u gradu koristi sustave otvorenog kôda.
  - Pošta Sjedinjenih Američkih Država (*U.S. Postal Service*) – za poslužiteljsku infrastrukturu.
- Španjolska; većina državnih institucija i agencija te nekih gradova; primjerice Barcelona.
  - Napravili su i svoju distribuciju Linuxa, baziranu na *Debian* Linuxu, naziva: LinEx, za neke od institucija.
  - Svi poslužitelji su migrirani na Linux: preko 900 klastera sa Linux poslužiteljima, širom države.

. . .

**Školstvo**:

- Brazilsko školstvo: koriste Linux na 523.000 računala u 50.000 škola širom države.
- Filipini: instalirano je 13.000 računala (s planom od još 13.000).
- Indija: 2.650 državnih sveučilišta planira koristi (ili već koristi) Linux i sustave otvorenog koda. Razvijaju svoju distribuciju Linuxa (*KITE GNU-Linux*).
- Kina: razvija svoju distribuciju Linuxa, na procesorima ***Loongson***; (trenutno sa 150.000 računala u primjeni).
- Njemačka Sveučilišta: koriste *SuSe* distribuciju Linuxa za poslužitelje i stolna računala u 33 sveučilišta, za 560.000 studenata.
- Savezna Država Indiana (SAD), školski sustav: koriste Linux za 22.000+ studenata i učenika.
- Italija, regija *Bolzano*: školski sustav sa 16.000 studenata i učenika.
- Sjeverna Makedonija, školski sustav: Linux stolnih računala: 5.000 i 180.000 *Ubuntu Thin Client* računala.
- Rusija – veći dio školskog sustava.
- Švicarska: pojedini kantonalni školski sustavi: koriste Linux s *Open Office* paketom; primjerice Ženevski kanton: ~9.000 računala.

**Neke od velikih tvrtki ili institucija koje koriste Linux i sustave otvorenog kôda za poslovanje ili unutar svojih proizvoda:**

- ***Google***: na poslužiteljima i stolnim računalima.
- ***IBM***: na poslužiteljima i stolnim računalima.
- ***Panasonic***: koriste Linux na poslužiteljima i stolnim računalima, *platinasti su član* ***AGL*** fondacije.
- ***Cisco***: na poslužiteljima i stolnim računalima.
- ***Amazon** i **Amazon AWS***: za skoro svaku komponentu poslovnog sustava se koristi Linux, kako su i sami izjavili: "*Everything that happens in them is driven by Linux*", ***Virgin America***: sustav za zabavu unutar zrakoplova (tzv. *in-flight entertainment system*).
- ***Peugeot***: koriste: 20.000 *Novell* Linuxa i 2.500 *SuSe Linux Enterprise Server* distribucija Linuxa na poslužiteljima.
- ***Wikipedia,*** Londonska i burza u New York-u koriste *Red Hat Enterprise Linux.*
- ***Automotive Grade Linux** zaklada **(AGL)** zaklada:*
  - ***Toyota*** – *zlatni* je član Linux zaklade, kao i *platinasti* član ***AGAutomotive Grade Linux L*** zaklade– njeni odjeli prodaje koriste Linux za oko 30. različitih sustava: od sustava za naručivanje dijelova, garancije, popravke i drugo. Komunikacijski sustav i *Infotainment* sustav unutar vozila također koristi Linux.
  - ***Mazda*** – *platinasti* je član ***AGL*** zaklade; koristi Linux za sustav unutar raznih komponenti automobila (*Infotainment/navigacija*), ***Suzuki*** – *platinasti* član ***AGL*** zaklade, ***Honda*** – *zlatni* član ***AGL*** zaklade, ...
  - ***Srebrni članovi AGL*** zaklade su i:
    - ***Mercedes-benz** grupacija,*
    - ***Ford**,*
    - ***Volkswagen, ….***
- ***Facebook** i **Twitter**.*
- ***Siemens**, **Ericsson**, **HP**, **Dell**, **Oracle**,*
- ***Samsung**, **Microsoft**, **DreamWorks**, **ENEA**, **ATOS**.*
- ***NTT Telekom** grupa (Nippon Telegraph and Telephone Corporation)* - vlasnik preko 30 telekom tvrtki širom svijeta.
- ***Morgan Stanley**, **Goldman Sachs**, **McDonalds**, **Deutsche Bank**, **Bank of America**, **Bank of New York**, …*
- ***Članovi Linux fondacije (www.linuxfoundation.org):***
  - ***Platinasti članovi***:
    - ***Ericsson, Fujitsu, Hitachi, Huawei.***
    - ***Intel, Meta, Microsoft, NEC, Oracle, Qualcomm, Samsung, IBM/RedHat,*** …
  - ***Zlatni članovi:***
    - ***Baidu, Cisco, Dell, Google.***
    - ***Honda, Panasonic, Sony, Toshiba, Toyota,*** …
  - ***Srebrni članovi su:***
    - ***AMD, Adobe, AirBnB, Akamai, Alianz, Alibaba, AWS, American Express, Arm,***
    - ***AT&T, ASrock, Autodesk, Bayer, Bosch, Bull, Canonical, Citrix, Comcast, …. …***

**Izvori informacija: (807),(808),(809),(810),(811),(812),(813),(814),(815),(816),(1661).**

# 2.4. Tko još (pametan) razvija programe i sustave otvorenog koda

Dobro, mnoge državne institucije i tvrtke sve više koriste programe i sustave otvorenog kôda, ali tko još (pametan) od tvrtki razvija ovakve sustave? Navesti ćemo samo nekoliko velikih tvrtki koje su najviše uložile u programe i sustave otvorenog koda:

***IBM***

- Uložio je više milijardi američkih dolara u servise i programe bazirane na Linuxu, kroz *Linux Technology Center* koji uključuje 300+ stalno zaposlenih Linux programera.
- Donira 40 milijuna američkih dolara u ***Eclipse*** (http://www.eclipse.org) sa 12.5+ milijuna linija programskog kôda.
- Ulaže u "*Open Source Initiative*" http://opensource.org.
- Donira u razvoj "*Java-base relational database management system*" (RDBMS) Apache *Derby* (http://db.apache.org/derby).

***SUN Microsystems** (od 2009.g.u vlasništvu tvrtke **Oracle**)*

- 1990. Razvija programski jezik *Java* koji 1995.g. predstavlja javnosti te ga 2007 licencira pod GPL licencom (6.5 milijuna linija kôda). Kasnije objavljuje *OpenJDK* kao referentnu implementaciju programskog jezika *Java*.
- 1999. kupuje njemačku tvrtku ***Star Division*** za 73.5 milijuna američkih dolara i između ostaloga s njom "*StarOffice*" kojeg kasnije *Sun* objavljuje kao ***Open Office*** pod LGPL i SISSL licencama (~10. milijuna linija programskog kôda).
- 2005. je razvio i pokrenuo ***GlassFish*** (aplikacijski server) kao projekt otvorenog kôda.
- Kasnije 2005 godine kupuju tvrtku "***Storage Tek***" za 4.1 milijarde US$ te 2008 objavljuje izvorni kôd "*Sun Storage 7000 Unified Storage systems*" koji je osnova za razvoj ZFS-a, a koji je kombinacija datotečnog sustava i "*volume nanagera*".
- 2005. objavljuje izvorni kôd kernel i mrežnog dijela operativnog sustava ***Solaris***, potom pokreće ***Open Solaris*** s istim kôdom pod CDDL licencom, kasnije uključuje i DTRACE, ZFS, ***Solaris Containers*** i druge tehnologije (2+ milijuna linija programskog kôda).
- 2008. kupuje tvrtku ***MySQ***L *A.B.* s ***MySQL*** bazom podataka za 1 milijardu američkih dolara. Kasnije objavljuje *MySQL* poslužitelj pod GPL licencom. Iz ***MySQL*** programskog kôda se potom razvija *MariaDB* baza podataka.

***Google***

- 2005. kupuje tvrtku ***Android Inc***. te s njom dobiva *Android* operativni sustav (koji je baziran na Linuxu).
- 2007. izdaje ***Android*** pod Apache 2.0 licencom (10+ milijuna linija kôda).
- 2008. razvija ***Google Chrome*** web preglednik te isti izdaje pod nazivom ***Chromium*** u otvoreni kôd (2+ milijuna linija programskog kôda).
- Razvija ***Chrome/Chromium OS***.
- Razvija ***Google Web Toolkit*** (300.000+ redaka programskog kôda).
- Google *Summer of Code* (GSoC) je godišnji događaj, prvi puta održan 2005. U njemu Google nagrađuje sa 5.000 američkih dolara (u 2013.) stotine studenata koji uspješno završe zadane projekte otvorenog kôda do kraja događanja.

***Red Hat** (od 2019.g. u vlasništvu tvrtke **IBM**)*

- Praktično živi od filozofije otvorenog kôda.
- Čisto za informaciju, pogledajmo ukupne prihode tvrtke *Red Hat*, samo kroz nekoliko godina:
    - u 2015. godini: 1,79 milijardi (1.790.000.000) U$
    - u 2016. godini: 2,05 milijardi (2.050.000.000) U$

- ***Red Hat Enterprise Linux*** (*RHEL*) je dostupan samo kroz pretplatu, koja uključuje pristup softveru (ISO *image*) i zakrpama te raznim razinama tehničke podrške (ovisno o vrsti pretplate).
  Proizvod se sastoji većinom od programa koji se distribuiraju kroz licence otvorenog kôda (engl. *open source*).
    - Cijeli programski kôd za sve programske pakete *RedHat* Linuxa je javno dostupan: *CentOS i Rocky* Linux, kao i *Oracle* koriste isti taj programski kôd za izradu svojih distribucija Linuxa zvanih redom: ***CentOS, Rocky*** i ***Oracle Enterprise Linux***.
    - Razvili su (i napisali) oko 12% programskog kôda cijelog Linux kernela*, prema broju redaka programskog kôda.
    - ***JBoss*** aplikacijski poslužitelj s 2+ milijuna redaka programskog kôda, izdan je pod *LGPL* licencom.
    - *Red Hat* sponzorira ***Fedora*** projekat, kao tzv. *community* odnosno razvojnu inačicu ***Red Hat Enterprise Linuxa***.

***Apple***

- Sredinom 2015 (ljeto) objavljuje da će programski jezik *Swift* odnosno *Swift* 2, koji je nasljednik ili više zamjena *Objective-C* programskog jezika biti u potpunosti objavljen kao otvoreni kôd. Programski jezik *Swift* će se koristiti za razvoj aplikacija za ***OS X*** i ***iOS***, te će raditi na Linux OS-u. To znači kako će od sada biti moguće razvijati aplikacije za ***OS X*** ili ***iOS*** na ***Linuxu***.

***Microsoft***

- Microsoft je na 21. mjestu po sudjelovanju u razvoju Linux kernela (~1% udjela, prema broju linija kôda).
- Shvaća prednosti razvoja prema principima Open Source: http://opensource.mscommunity.hr: "*Microsoft Hrvatska 5. lipnja 2013. organizira prvu Microsoft Open Source konferenciju u suradnji sa Sveučilištem u Zagrebu, koja je namijenjena upravo vama. Microsoft je kao tvrtka proširio svoju filozofiju i sve više pridonosi open source zajednicama tako da promiče developersku suradnju kako bi se lakše razvijala i održavala interoperabilna IT rješenja.*"
- Početkom 2015 objavljuje "***.NET Core***", kao otvoreni kôd, u to su uključeni:
  - **C#** i **Visual Basic** te **Visual F# Tools** i **ASP .NET 5**.
- Nekoliko mjeseci nakon otvaranja "***.NET Core***"-a, zajednica programera i korisnika pronašla je i riješila na stotine grešaka te razvila nove funkcionalnosti.

## Tko još?

Uz navedene velike korporacije i druge velike tvrtke ulažu u razvoj programa i sustava otvorenog kôda, uključujući Linux. Prema nekim analizama od prvih milijun najposjećenijih web stranica na svijetu, njih 96.3% koristi Linux za poslužitelje.

Ako sâmo pogledamo članove Linux zaklade (engl. *Linux Fondation*), u koju ulažu mnoge tvrtke i pojedinci, možemo vidjeti da su trenutni(2020.g.) ukupni procijenjeni troškovi razvoja svih projekata koje ova zaklada podupire/razvija, nekih 16 milijardi Američkih dolara (16.000.000.000 U$) te da za njih rade stotine tvrtki i njihovih zaposlenika, kao i nebrojeno puno pojedinaca.

Osim drugih načina doniranja razvoja, poput činjenice da mnoge tvrtke plaćaju svoje zaposlenike da rade na ovim projektima, mnoge tvrtke dodatno plaćaju godišnju članarinu koja otprilike iznosi: za platinaste članove 500.000 U$, za zlatne i srebrne 100.000 U$ godišnje, te za standardne članove, ovisno o broju zaposlenika (pr. za tvrtke do 499 zaposlenih je to 10.000 U$).

Ovdje se radi o stotinama tvrtki koje osim što razvijaju ove projekte, plaćaju i godišnje članarine s kojima se i financira rad zaklade. Uz navedene zaklade postoje i druge velike zaklade i projekti slične vrste, poput projekta za razvoj Linuxa u automobilskoj industriji, imena: *Automotive Grade Linux* (***AGL***), čiji članovi osim plaćanja godišnje članarine, podupiru i razvijaju sustave iz ove domene. Članovi ove zaklade su vodeći proizvođači automobila, integriranih krugova i sklopova te tvrtke iz drugih povezanih grana proizvodnje.

**Što je s malim tvrtkama?**

I veliki broj malih tvrtki je osnovan na bazi poslovnog modela koji obuhvaća neke (ili sve) segmente komercijalne upotrebe programa i/ili sustava otvorenog koda:

- Pružanja tehničke podrške za kupce te razvoja i optimizacije prema potrebama klijenata (kupaca).
- Razvoja zasebnih komercijalnih "*Close Source*" ili "*Open Source*" aplikacija za postojeći program ili sustav otvorenog kôda te integracije sustava kod kupaca i tako dalje.

Postoje i mnoge druge fondacije i projekti koji su vrlo važni za zajednicu, ali i prvenstveno za sâm Linux. Naime Linux kao i svaki drugi operativni sustav sastoji se od više elemenata, među kojima su jezgra sustava (tzv. ***kernel***) te mnogi sistemski, uslužni i drugi programi, ali i mnoge programske biblioteke.

U prosjeku svaka od takozvanih distribucija (varijanti) Linuxa sastoji se od mnogih komponenti i nerijetko tisuća programa, koji dolaze instalirani u takozvanim programskim paketima. Pri tome se uobičajeno unutar svakog pojedinog programskog paketa nalaze desetci programa ili čak stotine raznih programskih biblioteka.

Pogledajmo okvirnu tablicu koja prikazuje broj standardno instaliranih softverskih paketa, ovisno o namjeni distribucije Linuxa:

| Distribucija Linuxa (naziv) | *Workstation (okvirni broj softv. paketa )* | *Server (okvirni broj softv. paketa* |
|---|---|---|
| **Debian** | 500 – 1000 | 100 – 500 |
| **Ubuntu** | 500 – 1000 | 100 – 500 |
| **Red Hat** | 300 – 600 | 100 – 300 |

Pogledajmo i okvirnu tablicu sa samo nekima od većih projekata ili fondacija koje razvijaju važnije programe za Linux (i druge operativne sustave).

Ovdje je vidljivo da se unutar fondacije **Free Software Foundation** i **GNU** projekta unutar nje razvija najveći broj programa otvorenog programskog kôda. U tablicu nismo uvrstili stotine neovisnih projekata otvorenog kôda.

| Razvijatelj (fondacija, projekt, …) | Broj programskih paketa koje razvija |
|---|---|
| **Free Software Foundation**<br>Od kojih **GNU projekt** | **~16.000**<br>**~ 380** |
| **Apache Software Foundation** | **Više od 350** |
| **Mozilla Foundation** | **Više od 10** |
| **Cloud Native Computing Foundation** | **Više od 10** |
| **OpenBSD projekt** | **Više od 20** |

***Linux kernel koriste sve distribucije Linuxa.**

**Izvori informacija: (805),(806),(818),(819),(1554)**

## 2.5. Filmovi i literatura koju preporučujemo

Za sve koji žele vidjeti komentare i iskustva ljudi koji su najviše doprinijeli razvoju *Open Source* inicijative, iz prve ruke, preporučujemo da pogledaju dokumentarni film: **Revolution OS,** *IMDB TT0308808*.

U ovom dokumentarnom filmu nam svoja iskustva prenose:

- *Richard M. Stallman:* programer, tvorac pokreta otvorenog koda: (Engl. *Free Software Movement)* i *GNU* projekta.
- *Linus Torvalds*: tvorac Linux kernela.
- *Eric Steven Raymond:* programer UNIX programa, kasnije i GNU/LINUX i autor knjige "**The Cathedral and the Bazaar**", suosnivač inicijative otvorenog koda (engl. *Open Source Initiative* (*OSI*)).
- *Bruce Perens:* tvorac definicije *Open Source*, programer i suosnivač *Open Source Initiative (OSI).*
- *Larry Augustin:* suvlasnik/Osnivač tvrtke *VA Linux Systems.*
- *Michael Tiemann*: suosnivač tvrtke *Cygnus Solutions* koja je razvila i ***Cygwin***, autor je GNU C++ prevoditelja (engl. *Compiler*), GNU C prevoditelja i GNU *Debuggera*, bio je predsjednik *Open Source Initiative*, tehnički direktor u tvrtki *Red Hat*, član savjetodavnog odbora za *GNOME* projekt, član je odbora za *Embedded Linux Consortium*, …

Dodatno preporučujemo i:

- Dokumentarnu mini seriju: *The Triumph of the Nerds: The Rise of Accidental Empires*, *IMDB TT0115398.*
- Dokumentarni film *Secret History of Hacking*, *IMDB TT2335921*.
- Dokumentarnu mini seriju: *Discovery Channel; Rise of the Video Game*:

Ako se bavite razvojem softvera (ne nužno i *otvorenog kôda*), preporučujemo da pročitate knjižicu (~35 stranica) koja je promijenila stavove mnogih tvrtki o *sustavima otvorenog kôda*.

Autor knjige je *Eric Steven Raymond* koji je od sredine 1980 radio na razvoju raznih programa za Unix:

- Na raznim komponentama programa *Emacs te igrama: Net Hack, Xlife,* a kasnije i na *The Battle for Wesnoth*.

On je praktično cijelo desetljeće radio na razvoju *Unix* programa ustaljenim i stabilnim metodologijama razvoja (poput *Katedrale* kako i sâm kaže), a odatle i naziv knjige. Nakon toga krenuo je u avanturu razvoja programa metodologijom koju je preporučio *Linus Torvalds*, koju on naziva modelom tržnice (engl. *Bazaar*).
Dakle razvoja u kojem sudjeluje veći ili veliki broj ljudi, od kojih svatko često objavljuje promjene programskog kôda …, ali o tome u knjizi. On nam opisuje svoja iskustva i probleme uz rješenja, na koje je nailazio u razvoju e-mail klijenta (*Fetchmail*) na kojem je počeo raditi.
Njegova knjiga je i po mnogima uzrokovala dobar dio odluke tvrtke *Netscape Communications Corporation* koja je tvorac, u to vrijeme najboljeg Web preglednika, da kompletan programski kôd *Netscape Navigator* preglednika objave kao otvoreni kôd te pokrenu ***Mozilla*** fondaciju.

*Mozilla* fondacija je neprofitna organizacija koja se bavi očuvanjem interneta kao javnog resursa koji je otvoren i dostupan svima, a iz *Mozilla* fondacije proistekli su projekti poput:

- **Firefox** – besplatan web preglednik otvorenog kôda, dostupan za mnoge operativne sustave.
- **Thunderbird** - besplatan klijent za elektroničku poštu otvorenog kôda, dostupan za mnoge operativne sustave.
- **Bugzilla** - besplatan sustav otvorenog kôda za praćenje grešaka (*bugova*). …



Taj potez je kasnije otvorio vrata za razvoj današnjih preglednika baziranih na *Mozilla Firefox* pregledniku koji je uz mnoge druge tehnologije i programe (pr. *Thunderbird* klijent za elektroničku poštu) razvijan unutar fondacije ***Mozilla***.

Navedena knjižica se zove **The Cathedral and the Bazaar**, a možete ju potražiti na internetu: njena izvorna inačica se može pronaći na poveznici: http://www.unterstein.net/su/docs/CathBaz.pdf.

Postoji i proširena inačica od stotina stranica koja je naravno komercijalna i zanimljiva, ali mislimo kako je za razumijevanje svega navedenog dovoljna i ova izvorna inačica.



Za kratki popis programa i sustava otvorenog kôda, prema kategorijama, pogledajte poglavlje:
**30.4. Popis programa otvorenog programskog kôda.**

**Izvori informacija: (68),(1567),(1568).**

# 2.6. Zašto Unix odnosno Linux

Priču o Linuxu moramo započeti puno prije njegovog vremena nastanka, točnije nekih dvadesetak godina ranije.
Ona počinje s razvojem *Unix* operativnih sustava. Što je toliko posebno oko UNIX-a?
Razne varijante UNIX-a su u upotrebi već nekoliko desetljeća, provjerene su u radu u mnogim industrijskim i drugim zahtjevnim okruženjima, poput primjene u: telekomima, bankama, industriji, medicini, transportu, energetici, vojsci, … .

- Tisuće programa koji čine UNIX/Linux su također u vrlo dugoj upotrebi te su ispolirani, testirani i višestruko optimizirani u radu.
- Unix i navedeni programi prilagođeni su mnogim okruženjima i u njima rade optimalno, pouzdano i stabilno.

## 2.6.1. Povijest Unixa

Priča o nastanku *UNIX-a* prati nevjerojatne ljude, vizionare i vrhunske inženjere. ***UNIX*** (engl. *Uniplexed Information and Computing Service*) je nastao kao odgovor na tadašnji ***Multics*** (engl. *Multiplexed Information and Computer Services*) koji je bio prevelik (za to vrijeme) i kompleksan, a sâmim time i vrlo kompliciran za rad. Razvoj UNIXa prati ljude koji su bili spremni odreći se puno toga, na putu k ostvarenju svojih vizija. Priča prati nevjerojatne pojedince i grupe ljudi koji su cijelo to vrijeme bili puni entuzijazma te nisu bili spremni na kompromise.
Ovi stvarni velikani IT svijeta su promijenili naš svijet, a da toga možda nismo niti svjesni.
Bez njih ne bi bilo niti interneta (barem onakvog kakav danas poznajemo), *Unixa*, *Linuxa*, *Mac OS*-a ili iPhone-a i Androida, a ne bi bilo niti *BlackBerry*-ja. Nadalje, bez njihovog entuzijazma ne bi bilo niti programskog jezika **C** kao niti jezika koji su nastali na osnovi programskog jezika ***C***. Naime na osnovama programskog jezika ***C*** nastali su sljedeći programski jezici:

- **C++** i **D**
- **Go** i **Rust**
- **Java**
- **JavaScript**
- **Limbo**
- **LPC**
- **C#** i **Objective-C**
- **Perl** i **PHP**
- **Python**
- **Swift**
- **Verilog**
- Kao i mnogi drugi.

Sâmo programski jezik **Java** danas koristi preko **3 milijarde** računala i raznih uređaja: od "*set-top box*" uređaja, preko pisača, web kamera, navigacijskih sustava, raznih terminala, medicinskih uređaja, kućanskih aparata, televizora i slično.
Osim toga ne bi bilo niti operativnih sustava koji su danas u upotrebi (ili bi bili znatno lošiji), te raznih vrlo važnih tehnologija koje se nalaze ugrađene u sve današnje operativne sustave. Dodatno i većina operativnih sustava je razvijena u ***C*** jeziku ili nekoj njegovoj izvedenici, a ne zaboravimo i na upravljačke programe (engl. *drivers*) za iste sustave, koji se također moraju razvijati u nekom programskom jeziku poput ***C***, ***C++*** ili sličnih. Možda niste znali, ali ***Appleovi MacOS*** i ***IOS*** su također bazirani na određenim varijantama UNIXa. Ne zaboravimo niti sve mrežne uređaje koji su okosnica Interneta, od usmjerivača (*routera*), preklopnika (*switcheva*), vatrozida (*firewalla*), ***IPS*** i ***IDS*** uređaja i slično; gotovo svi oni upogonjeni su nekom varijantom UNIX-a ili Linuxa.

Tako primjerice ***Cisco Systems*** za svoj ***IOS*** (operativni sustav) koristi varijantu *FreeBSD Unixa*, dok za ***IOS XR*** koristi *QNX Unix* odnosno za ***NX-OS*** koristi ***MontaVista Linux***. S druge strane ***Juniper Networks*** koristi *FreeBSD Unix* za sve svoje mrežne uređaje. ***Arista Networks*** na svim uređajima koristi nemodificirani Linux kernel na bazi kojeg je napravila svoj ***EOS*** (*Extensible Operating System*). Gotovo sve (ako ne i sve) digitalne telefonske centrale koriste neku varijantu *UNIXa ili Linuxa,* kao i sve bazne stanice za mobilne uređaje.
Sve mobilne mreže ili barem vrlo veliki dio uređaja, servisa (usluga) i opreme koja stoji iza njih također koristi neki *UNIX ili Linux*. Pošto pristup internetu u konačnici završava (ili počinje) kod nekog telekoma odnosno *ISP*-a, ponovno dolazimo do zaključka kako bi Internet bez *UNIXa* postao upitan, a čak bi se usudili reći i nemoguć. Neka od varijanti *UNIXa* (u što ubrajamo i Linux), u upotrebi su i u robotici, a osim robotike, UNIX je u upotrebi i u avionskoj i automobilskoj industriji, kao i u medicinskoj industriji, ali i mnogim svemirskim programima: od letjelica, orbitera, do raznih satelita.



*Ne zaboravimo i cijeli nîz uređaja u kućanstvu; od pametnih televizora, pećnica, mikrovalnih pećnica, hladnjaka i zamrzivača, kao i danas sve popularnijih; danas milijuna, a u budućnosti i milijardi* ***IoT*** *uređaja.*

Lista upotrebe neke od inačica ili varijanti *UNIXa* je prilično velika, a važno je sâmo shvatiti kako bi svijet bez njega (*UNIXa*) bio potpuno drugačiji; vjerojatno bi prema upotrebi tehnologija koje bi nam bile dostupne bez *UNIXa*, danas živjeli u 70-tim godinama prošlog stoljeća.

Operativni sustav *Unix* razvili su znanstvenici ***Ken Thompson*** i ***Dennis Ritchie*** u tvrtki ***AT&T***, točnije radeći u njihovoj sestrinskoj tvrtki ***Bell Laboratories*** krajem 1969 godine. Prva radna inačica (*verzija*) nastala je početkom 1970 godine.
Prve inačice *UNIXa* su bile razvijane u programskom jeziku ***Assembler,*** a kasnije, negdje tijekom 1972, cijeli *Unix* je napisan od početka, korištenjem programskog jezika **C**, koji je prethodno, navedeni dvojac i razvio.

Prelaskom na programski jezik ***C*** povećala se portabilnost to jest prenosivost softvera, odnosno sâmog *UNIX-a* i na druge platforme odnosno arhitekture procesora. Naime programski kôd pisan u programskom jeziku *Assembler* je čvrsto vezan za arhitekturu procesora i hardver te ga je bilo gotovo nemoguće ili vrlo teško prepisati za neku drugu arhitekturu procesora.
Prelazak na programski jezik ***C*** je pojednostavio ovaj proces, pri čemu je zadržana njegova brzina rada.
Danas se *Unix* koristi na različitim arhitekturama procesora poput:

- ***x86*** – ovo je najčešća arhitektura procesora koju proizvode tvrtke poput: *Intel*, *AMD*, *VIA/ Centaur*, *Hygon*, *zhaoxin*.
- ***PowerPC***, ***MIPS*** i ***SPARC*** – procesore ove arhitekture proizvode: *IBM*, *Freescale*/*NXP*, *MIPS*, *Broadcom*, *Fujitsu*, *Oracle*.
- ***ARM*** – procesore ove arhitekture proizvode: *Qualcomm*, *Apple*, *MediaTek*, *Samsung*, *NVIDIA*, *Huawei*/*Hisilicon*, …
- ***RISC-V*** – procesore ove arhitekture proizvode mnoge tvrtke, a osim ove arhitekture koriste se i neke druge arhitekture.

Krajem 1970. te početkom 1980. godine tvrtka **AT&T** licencirala je *Unix* drugim tvrtkama koje su krenule s razvojem svojih inačica *Unixa*.



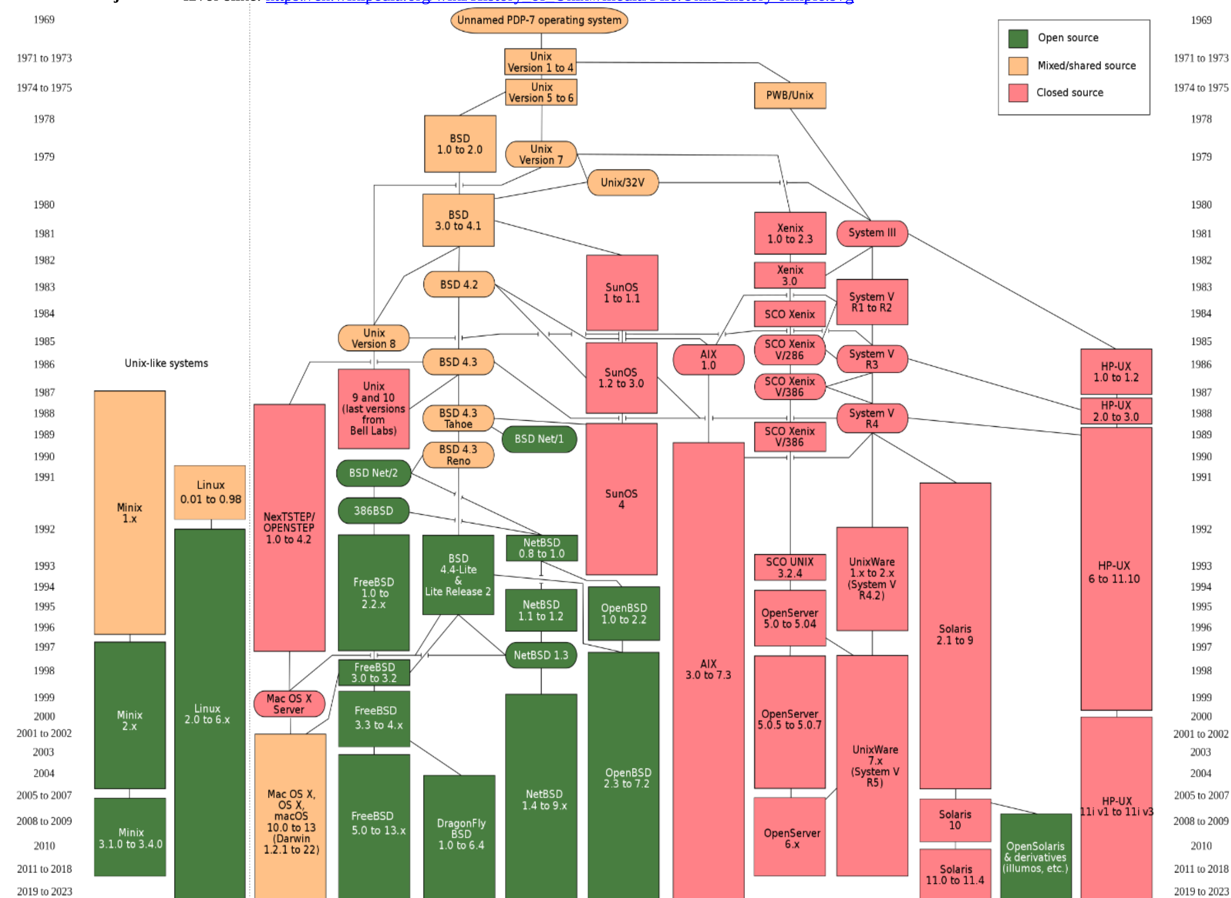
*UNIX* operativni sustavi su u širokoj upotrebi preko 50 godina i oni su utjecali na razvoj drugih operativnih sustava. Osim toga, veliki broj organizacija, tvrtki i pojedinaca su bili i jesu uključeni u njegov razvoj i napredak.



1.1.1970. godine smatra se datumom rođenja *Unixa*.

Na slici (1) dolje, vidljiv je razvoj raznih inačica (*verzija*) operativnog sustava *Unix* od 1970. godine, sve do danas:

**Slika 1. Razvoj Unix-a.** Izvor slike: https://en.wikipedia.org/wiki/History_of_Unix#/media/File:Unix_history-simple.svg




Autor fotografije: Levenez Unix History Diagram, Information on the history of IBM's AIX on ibm.com "User:Eraserhead1, Infinity0, Sav_vas"

Neke od inačica *UNIX-a*, koje su i danas u upotrebi su:

- ***BSD Unix*** (*Berkeley Software Distribution*) - razvijen u Kalifornijskom sveučilištu *Berkeley*
    - ***FreeBSD*** baziran na *BSD Unixu.*
        - ***DragonFly BSD*** baziran na *FreeBSDu.*
        - ***JunOS*** - baziran na *FreeBSD*, razvijen u tvrtki **Juniper Networks.**
        - ***NeXTSTEP*** (kasnije ***Mac OS/IOS***) - razvijen u tvrtki **NeXT Computer**, kasnije kupljen od tvrtke **Apple**, koji je kasnije postao osnova za **iOS** i **MacOS**.
    - ***NetBSD, OpenBSD i TrueOS*** bazirani su na *BSD Unixu.*
- ***SUN OS*** (kasnije ***Solaris***) - razvijen u tvrtki ***Sun Microsystems***, sada u vlasništvu tvrtke ***Oracle.***
- ***IRIX*** - razvijen u tvrtki ***Silicon Graphics***.
- ***HP-UX*** - razvijen u tvrtki ***Hewlett Packard*** (***HPE).***
- ***AIX*** - razvijen u tvrtki ***IBM.***
- ***Digital UNIX*** - razvijen u tvrtki **DEC** (*Digital Equipment Corporation*), kasnije **Compaq**, danas **HP.**
- ***QNX*** - razvijen u tvrtki ***Quantum Software Systems***, kasnije ***QNX Software Systems***, potom ***BlackBerry***.
- ***Minix*** (nastao od riječi "mini-Unix") - razvijen od strane *Andrew S. Tanenbaum*-a.

Detaljniji razvoj *UNIX-a* od 1969. do danas vidljiv je u dokumentu (koji se stalno osvježava): http://www.levenez.com/unix/unix.pdf



Zanimljivo je kako je i tvrtka **Microsoft** kupila licencu za *Unix* od tvrtke **AT&T**, krajem 1970, koji je razvijala pod imenom **Xenix**. Njihova inačica *Unixa* je sredinom 1980 postala i jedna od češće korištenih varijanti *Unixa*, ali ju je kasnije *Microsoft* prodao tvrtki **Santa Cruz Operation** (*SCO*) koja ju je nastavila razvijati pod imenom **SCO UNIX**, a danas se prodaje pod imenom **SCO OpenServer**.

**Za Unix možemo reći sljedeće**:

- Podržava *multitasking* to jest izvršavanje više zadataka odnosno programa istovremeno.
- Podržava višekorisnički rad, što znači kako je moguć rad više korisnika istovremeno.
- On je interaktivan, što znači kako omogućava interakciju s korisnikom te je izvođenje naredbi na njemu trenutno.

**Što je specifično za sve UNIX-oidne operativne sustave?**
Svi *Unixoidni* operativni sustavi (uključujući i Linux) imaju zajedničko nekoliko stvari:

- Koriste hijerarhijski datotečni sustav s točno definiranom i razrađenom strukturom direktorija (mapa).
- Konfiguracijske datoteke za sve programe i komponente sustava su obične tekstualne datoteke.
- Svi uređaji: od diskova (CD/DVD, tvrdih ili SSD), preko mrežnih, grafičkih i drugih kartica su također datoteke (doduše posebna vrsta). Stoga se komunikacija sa svim uređajima (hardverom) svodi na rad s datotekama; pisanjem ili čitanjem u ili iz njih. Čak se i komunikacija između procesa tj. **IPC** (*inter-process communication*) svodi na rad s (posebnim) datotekama.
- Svi programi i naredbe se razvijaju vođeni idejom: mali broj funkcionalnosti unutar jednog programa rezultira brzim, optimiziranim, dobro testiranim i dokumentiranim programom koji je lakše održavati. Dakle bolje je razvijati više manjih programa s manje funkcionalnosti unutar svakog od njih nego jedan glomazan program s velikim brojem funkcionalnosti, koji je znatno teže održavati. Dodatno naredbe treba razvijati tako da podržavaju prihvat i slanje tôka odnosno nîza tekstualnih podataka (Engl. *Text stream*) kako bi ih mogli zaprimiti odnosno poslati drugim naredbama (što se odnosi na sljedeću točku).
- Bilo koji program se može pokrenuti u kombinaciji s drugim programima (ulančavati), na način u kojem se obrađeni podaci koje nam daje prvi program, mogu proslijediti drugom programu, a ono što je on obradio, može se poslati trećem i tako dalje, stvarajući nove mogućnosti. Dakle bez potrebe za pisanjem novog programa, već kombinirajući funkcionalnosti postojećih programa. Ova mogućnost se zove ***Pipe*** ili cijev za komunikaciju te izgleda ovako:

```
program 1 | program 2 | program 3
```

Osnovna filozofija *Unixa* kaže: „*Everything is a file*" odnosno sve je (neka) datoteka i prema tome baratanje sa sâmim operativnim sustavom, njegovim aplikacijama kao i njegovim hardverom je vrlo jednostavno i svodi se na rad s običnim ili *posebnim* datotekama. Nadalje, moguće je *skriptirati* i automatizirati svaki djelić sustava, kao i odrađivati napredne zadaće poput filtriranja i preoblikovanja podataka ili teksta prema potrebi, upotrebom takozvanih ***shell*** skripti.

Trenutno stanje (2023.g.) po pitanju razvoja nekih od komercijalnih varijanti UNIX-a (ne i Linuxa) je sljedeće:

- *IBM* **AIX** se i dalje razvija, ali više ne za *POWER System* poslužitelje (ne zaboravimo da je *IBM* vlasnik *Red Hat* distribucije Linuxa).
- *Sun* (*Oracle*) **SOLARIS** - više ne postoji plan daljeg razvoja već samo održavanja.
- *Hewlett Packard* (***HPE***) **HP-UX** – stavljen na održavanje, bez daljnjeg razvoja novih značajki.
- *SCO Group*: **OpenServer (**sada baziran na **FreeBSD** Unixu**)** i **UnixWare** se još razvijaju.

**Izvori informacija: (69),(1185),(1436).**

# 2.7. Povijest otvorenog koda (*Open Sourcea*) i Linuxa

Za razumijevanje razvoja *Linuxa* važno je razumjeti i razvoj sustava otvorenog kôda (engl. *Open Source*). Naime 1983. godine pokreće se pokret slobodnog softvera (***Free Software Movement***), a godinu dana kasnije (1983/84) ***Richard Stallman*** pokreće ***GNU*** projekt koji 1984 kreće s razvojem prvih programa. Prema *Stallman*-u: naziv "*GNU*" je odabran jer ta riječ ima značenje (nije izmišljena), a predstavlja životinju Gnu (lat. *Connochaetes*), ujedno je i zanimljiva (smiješna) riječ u engleskom jeziku te je rekurzivna kratica za "*GNU's Not Unix*" čime se odalo priznanje UNIX-u iz kojeg je preuzeta logika funkcionalnosti operativnog sustava. Želja ***GNU*** projekta je bila sve programe koji postoje u komercijalnim *Unix* operativnim sustavima napisati "*od nule*", ne koristeći niti jednu liniju kôda od izvornih programa, jer bi to bilo kršenje autorskih prava.

Dakle željelo se napisati nove programe s istom ili proširenom funkcionalnosti u odnosu na postojeće i objaviti ih pod nekom od licenci otvorenog kôda. Veći dio ovog posla, a ovdje govorimo o tisućama programa, je dovršen negdje u toku 1992. godine.

Paralelno s ovom pričom, tijekom razvoja raznih varijanti UNIXa, došlo je i do razvoja *BSD UNIXa (Berkeley Software Distributions)* iz kojega, odnosno prema kojemu se kasnije razvio i *FreeBSD UNIX*, i to ponovno *„od nule"*.
On je također objavljen kao sustav otvorenog kôda. Ovaj potez je otvorio vrata razvoju cijelog nîza dodatnih programa otvorenog kôda, koji su prvotno zamišljeni za korištenje unutar *FreeBSD UNIXa*, a kasnije su iskorišteni i za druge operativne sustave.

Neovisno o razvoju ***FSF*** i ***GNU*** te ***FreeBSDa***, tijekom 1991. godine *Linus Torvalds* razvija jezgru sustava (Engl *Kernel*) i slijedi logičan korak odnosno povezivanje tisuća programa pisanih unutar *GNU* pokreta i sâmog *kernela*.

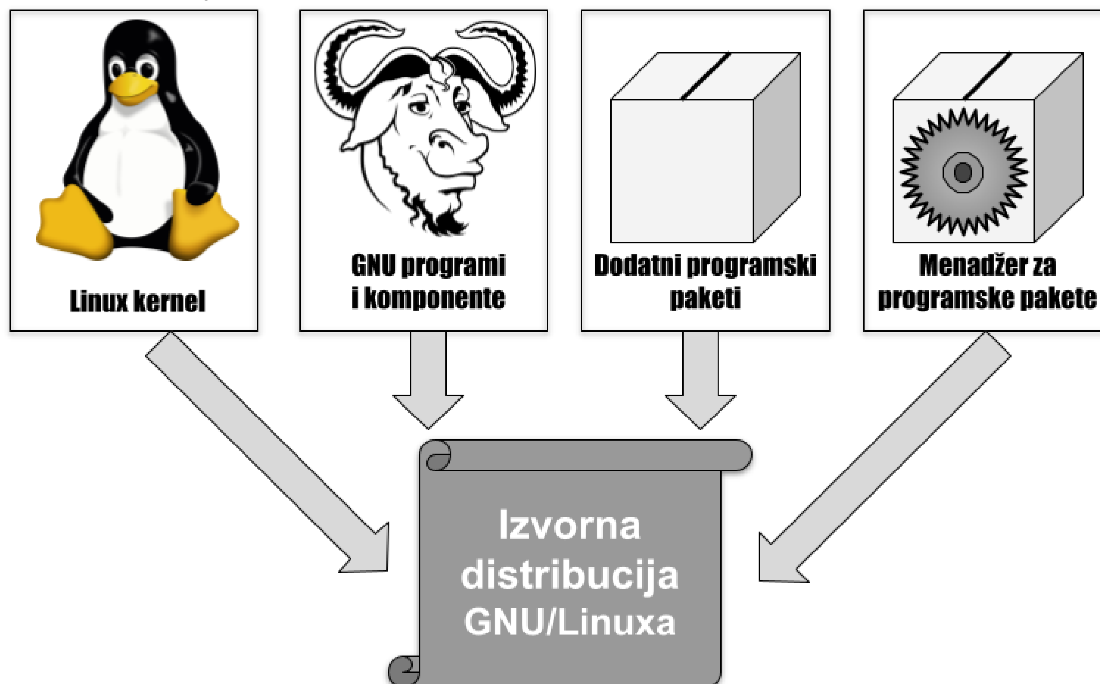
Od tog trenutka povezano je sve što je bilo potrebno za stvaranje novog operativnog sustava, danas poznatog pod nazivom *Linux*, mada ga je ispravno nazivati ***GNU/Linux***. Kasnije su dodavani i programski paketi koji su se razvijali prvotno za *FreeBSD* UNIX. Od tada do danas pojavile su se razne varijante *Linuxa*, od kojih su se ustalile tri značajnije grane, na osnovu kojih su nastale na stotine drugih *distribucija Linuxa*.

Tri najznačajnije grane takozvanih distribucija Linuxa su:

- ***Debian Linux***
- ***RedHat Enterprise Linux***
- ***Slackware Linux***

*Linux* se od prve inačice ili takozvane distribucije, nastavio razvijati preko nekoliko glavnih distribucija, što je vidljivo na sljedećim slikama (2, 3, 4 i 5). Distribucija Linuxa je naziv za operativni sustav koji se sastoji od jezgre Linuxa odnosno takozvanog kernela, [***GNU***] prôgrama i alata, dodatnog softvera i upravitelja odnosno menadžmenta softverskih paketa. Distribucija Linuxa može uključivati i poslužitelj zaslona (*X Window*) s okruženjem radne površine (engl. *Desktop environment*) kao i dodatni softver poput primjerice: *Libre Office, GIMP*, koji u pravilu dolazi u tzv. programskim paketima.

**Slika 1.A. Izvorna distribucija Linuxa**



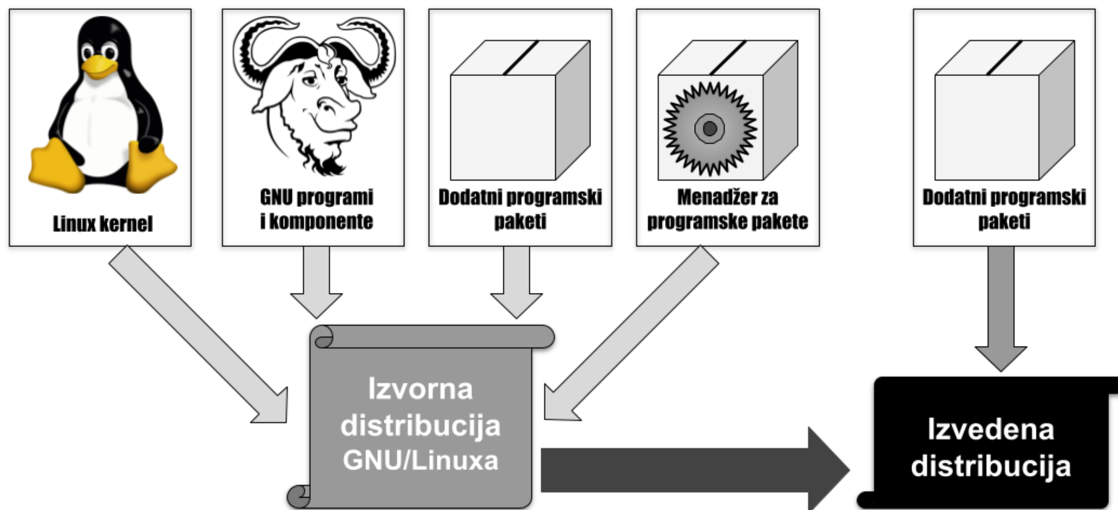

***Izvorna*** distribucija je vrsta distribucije Linuxa koja se ne temelji na bilo kojoj drugoj distribuciji Linuxa.
Na primjer, distribucije Linuxa kao što su *Debian*, *Slackware*, *Red Hat* i *Gentoo*, smatraju se izvornim distribucijama Linuxa, jer se ne temelje na nekoj dugoj distribuciji Linuxa, kako je logički prikazano na slici. 1.A..
Pojam distribucija Linuxa je nastao zato što entiteti poput *Debiana*, *Ubuntua* ili *Red Hata* ***distribuiraju*** Linux *kernel* zajedno sa svim potrebnim softverom i uslužnim programima; poput programa za konfiguraciju i rad mreže, menadžmenta softverskih paketa, radne okoline i drugog. U konačnici, distribucija Linuxa je potpuno samostalan operativni sustav.

Distribucija Linuxa je također odgovorna za osiguravanje ažuriranja: *kernela*, sistemskih, uslužnih i drugih programa, odnosno svih komponenti operativnog sustava. Pri tome se pojam Linux obično odnosi na jezgru operativnog sustava odnosno takozvani *kernel*, a pojam distribucija Linuxa, na cijeli operativni sustav.



Kako bi vam sve navedeno bilo jasnije, pogledajte i poglavlje:
**11. Od kojih komponenti se sastoji Linux.**

**Slika 1.B. Izvedena (derivatna) distribucija Linuxa**




***Izvedena*** distribucija (distribucija derivat) temelji se na izvornoj distribuciji Linuxa. Ovakva distribucija ima svoje drugačije ciljeve i namjenu od izvorne distribucije iz koje je potekla, pa obično sadrži dodatne programske pakete. Ono što je zajedničko izvornoj i izvedenoj distribuciji su: kernel, *GNU* programi i alati, inicijalni softverski paketi i paketni menadžer.
Primjerice *Linux Mint* temelji se na *Ubuntu Linuxu*, koji se pak temelji na *Debian Linuxu*. Stoga je *Linux Mint* praktično izvedenica od izvedenice. Izvedena distribucija Linuxa distribuira se kao zasebni samostalni operativni sustav.
Postoje i takozvane ***Flavor*** (engl. okus, aroma) varijante distribucija Linuxa. One imaju iste temeljne pakete i paketni menadžer kao izvorna distribucija, čak dijele i iste repozitorije kao i izvorna distribucija, ali imaju različite pakete softvera ili je instalirani softver konfiguriran drugačije. Primjerice *Xubuntu* je gotovo identičan *Ubuntu* Linuxu, ali s **Xfce** grafičkim sučeljem i nekim drugim programima, a *Kubuntu* je praktično *Ubuntu* s **KDE Plazma** grafičkim sučeljem i dodatnim programima.
Dakle *Xubuntu* i *Kubuntu* su derivati *Ubuntu* Linuxa, ali su s druge strane oni takozvane ***Flavor*** distribucije *Ubuntu* Linuxa.

***Rotirajuće odnosno takozvane Rolling distribucije Linuxa***.
U razvoju distribucija Linuxa prevladava model razvoja u kojem izdavači distribucije Linuxa periodički objavljuju nova izdanja; primjerice *CentOS* 6.1, nakon kojeg slijedi 6.2, te *CentOS* 7.1, nakon kojeg slijedi 7.2 i tako dalje. Uglavnom prema principu vršne inačice; poput *CentOS* 6.**x** ili *CentOS* 7.**x** koji praktično predstavljaju određenu generaciju, te mnogih njenih pôd inačica (**x**), u kojima i s kojima dolaze i nadogradnje sustava; u smislu ažuriranja programa i svih drugih komponenti sustava. Tako primjerice s *CentOS* Linuxom v.6.x dolaze razne inačice kernela 2.6.32.x, te određena generacija drugih programa i komponenti. Dok s *CentOS* Linuxom v.7.x dolaze razne inačice *kernela* 3.10.x, te novije (od 6.x) generacije drugih programa i komponenti sustava. Premda neke druge distribucije Linuxa nisu toliko vezane za određenu generaciju *kernela* (pr. 3.10. i sl.) već često koriste najnovije inačice *kernela*, kao što je slučaj s distribucijama koje su namijenjene stolnim računalima. Dok se kod poslužiteljskih inačica uglavnom svi drže određene provjerene i dugoročno podržane generacije *kernela* (pr. 3.10.x i sl.) te pripadajućih programa.



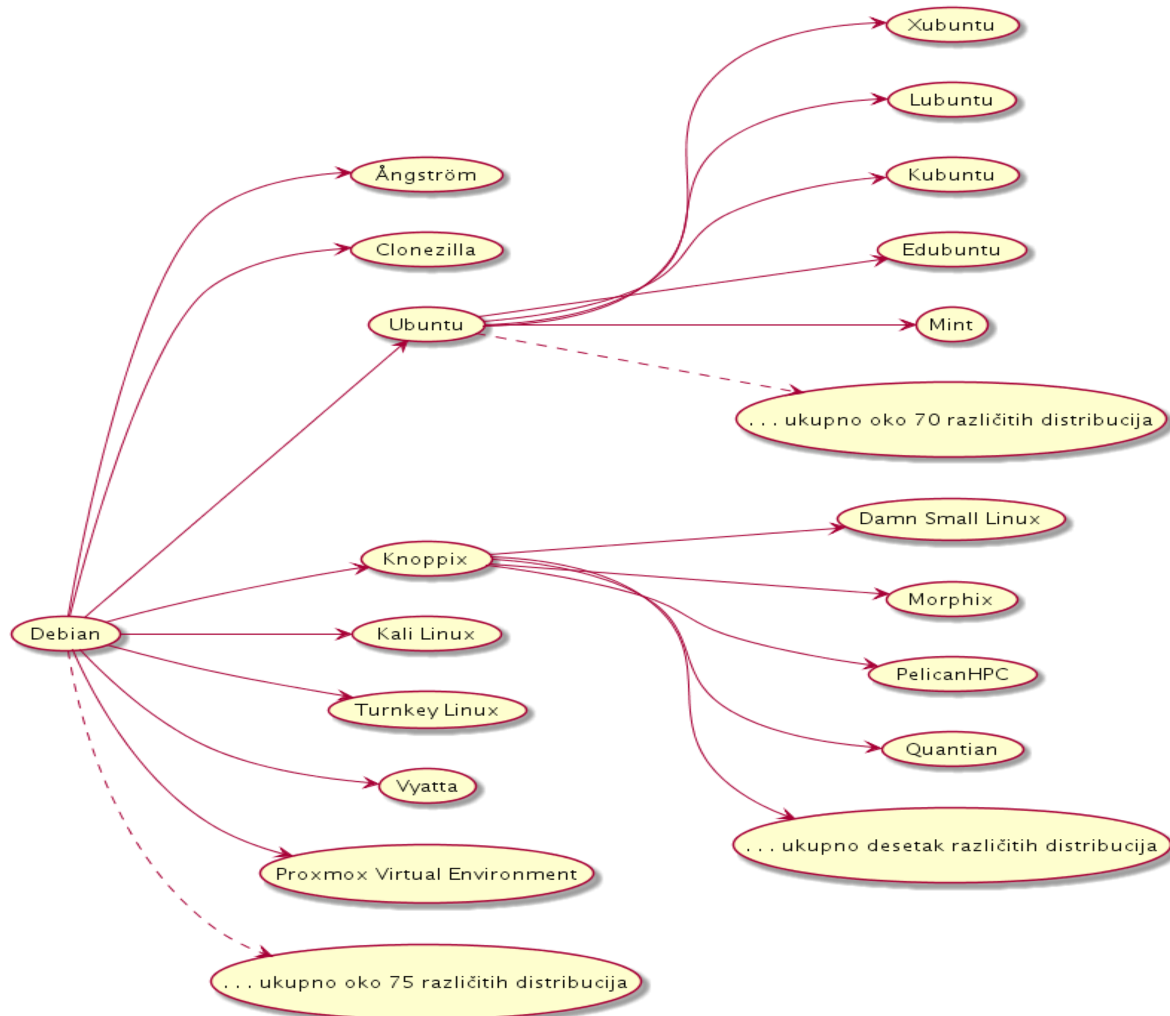
Vezano za jezgru Linuxa odnosno takozvani kernel, njegova stabla i inačice, pogledajte poglavlja:
**11.1. Kernel.**
**11.1.1. Rad s kernelom.**

Naime ovaj model razvoja se temelji na vršnim izdanjima te njihovim pod inačicama sustava (pr. *CentOS* 7.8) koje njeni izdavači periodički osvježavaju i objavljuju. Međutim postoji i jedan malo drugačiji model razvoja distribucija Linuxa koji nije toliko čest u upotrebi. To je model razvoja u kojem ne postoje klasična izdanja i inačice, već se konstantno aktivno (Engl. *Rolling*=rotirajuće, kotrljajuće) ažurira sustav, za razliku od prijašnjeg modela ažuriranja u serijama inačica (*verzija*). Na ovaj način cijeli sustav uvijek ostaje ažuran. Zbog toga ovakve distribucije Linuxa konstantno nude najnoviju jezgru Linuxa kao i svôg pripadajućeg softvera. ***Arch Linux*** je najpopularniji primjer ovakve distribucije Linuxa, dok je ***Gentoo Linux*** najstarija ovakva distribucija. Kada koristite ovakav model distribucije, dobivate mala, ali česta ažuriranja. Ovdje nema većih izdanja inačica tipa **X.**Y poput: ***Debiana*** ili ***Ubuntua***, ***Red Hata/CentOS/Oracle Linuxa***-a (ili ***Rocky*** Linuxa) ili drugih distribucija Linuxa, koji ipak čine veliku većinu svih distribucija Linuxa.

**Izvor informacija: (167).**

**Pogledajmo podjelu na standardne distribucije Linuxa: kako izvorne tako i njihove derivate (izvedene distribucije).**
Distribucije bazirane na **Debian Linuxu**(slika 2.), koji koristi `.deb` format paketa te `dpkg` menadžer paketa, kao i programe: `dpkg`, `dpkg-deb`,`dpkg-reconfigure`, odnosno programe više kategorije poput: `apt-get`, `apt-cache` i neke druge.

**Slika 2. Distribucije Linuxa bazirane na Debian distribuciji Linuxa.**



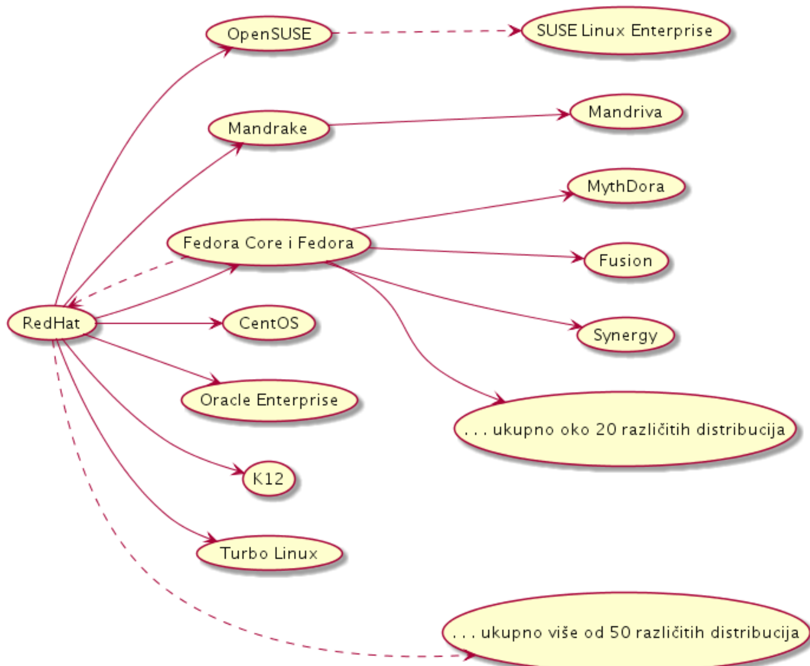

Na sljedećoj stranici ćemo vidjeti listu distribucija Linuxa baziranih na izvornoj distribuciji Linuxa naziva ***Red Hat Enterprise Linux***.

Iako smo već dosta govorili o ***Red Hat*** distribuciji Linuxa, spomenut ćemo još samo njene dugoročne inačice prema njenim generacijama, uz planiranu podršku:

- Generacija ***Red Hat Enterprise Linux*** **6**.x:
  - Objavljena 2010. godine.
  - S produženom podrškom (***ELS***) do 2024. godine.
- Generacija ***Red Hat Enterprise Linux*** **7**.x:
  - Objavljena 2014. godine.
  - S podrškom do 2024. godine.
- Generacija ***Red Hat Enterprise Linux*** **8**.x:
  - Objavljena 2019. godine.
  - S podrškom do 2029. godine.
- Generacija ***Red Hat Enterprise Linux*** **9**.x:
  - Objavljena 2022. godine.
  - S podrškom do 2032. godine.

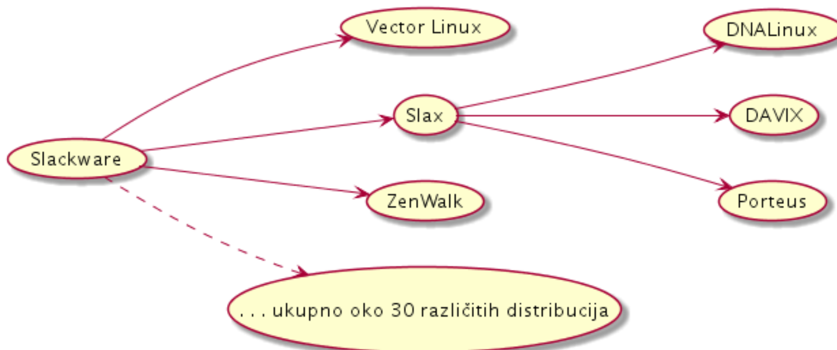
Pogledajmo i distribucije bazirane na ***Red Hat Enterprise Linuxu***(slika 3. dolje), koji koristi `.rpm` format paketa te `rpm` menadžer paketa i pripadajuće programe poput: `rpm`, `yum`, `dnf` i drugih.

**Slika 3. Distribucije Linuxa bazirane na RedHat distribuciji.**




Pogledajmo i distribucije bazirane na ***Slackware Linuxu***(slika 4. dolje), koji koristi `.txz` format paketa te takozvane `pkgtools` alate, kao i pripadajuće programe poput: `installpkg`, `removepkg`, `upgradepkg` i druge.

**Slika 4. Distribucije Linuxa bazirane na Slackware distribuciji.**




Osim navedenih značajnih grana distribucija Linuxa, nastale su mnoge distribucije Linuxa koje nisu direktno nastale od njih, dakle i one su izvorne distribucije. Neke od ovih pripadnika su:

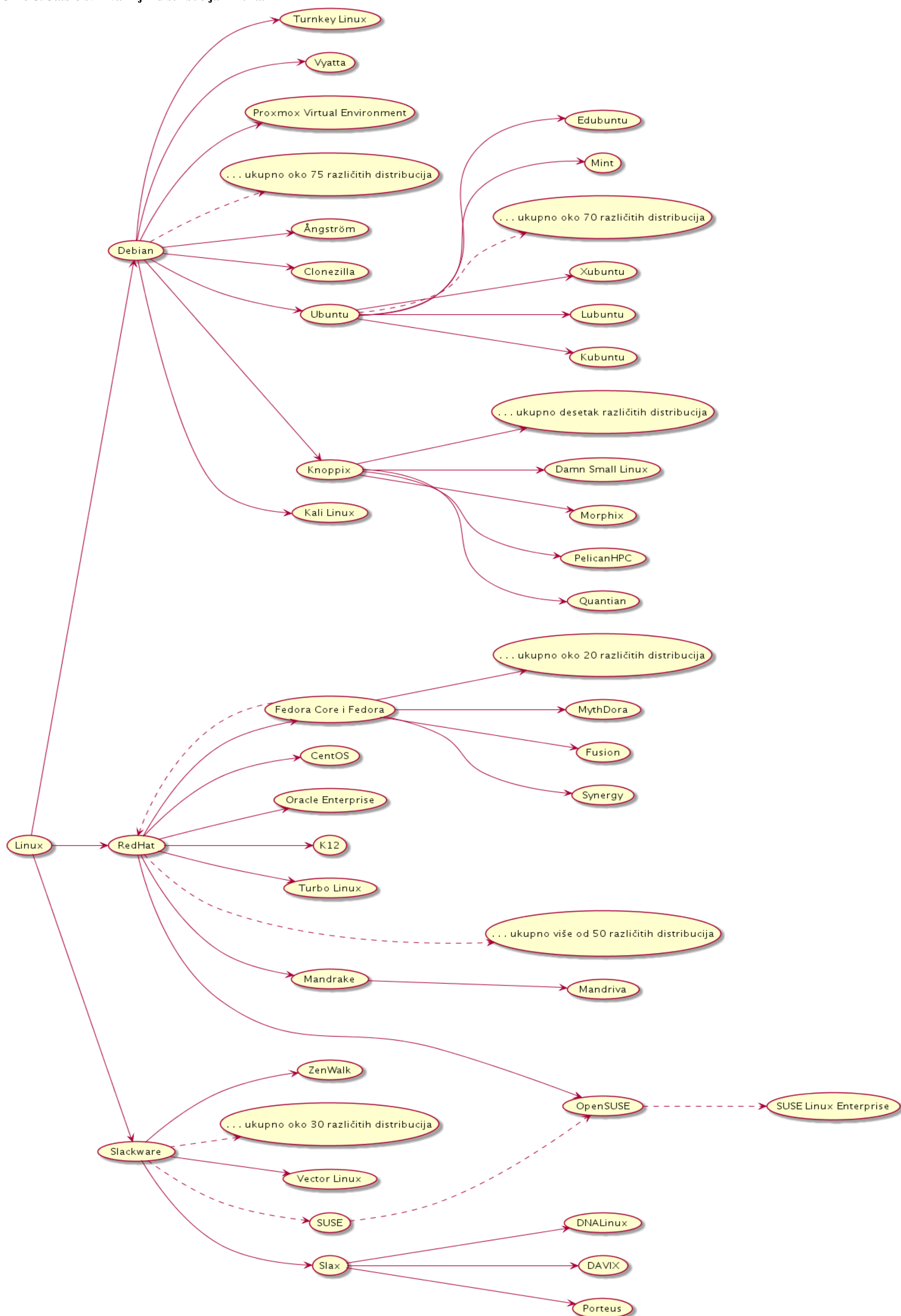
- *Android* - da i on je zapravo *distribucija* Linuxa.
- Alpine Linux - orijentiran je na sigurnost i minimalizam.
- Coyote Linux - nastao za potrebe usmjeravanje i vatrozida, iz njega su izrasli drugi projekti.
- Crux - fokusiran je na jednostavnost.
- FireOS - razvijen od tvrtke *Amazon* za vlastite potrebe. Baziran je na ***Android***-u.
- Firefox OS - razvijan od *Mozzila* fondacije i drugih, za mobitele, tablete, računala i televizore.
- KaiOS - razvijen od tvrtke Kai OS za mobilne uređaje.
- LineageOS - razvijena za mobilne uređaje, tablete, set-top-box uređaje i drugo. Baziran na Androidu.
- OpenWrt - fokusiran na usmjerivače i bežične pristupne točke (*Wireless Access Points*).
- Puppy Linux - razvijan s fokusom na minimalne hardverske zahtjeve.
- SliTaz - razvijan s fokusom na minimalne hardverske zahtjeve uz veliku bazu programskih paketa.
- SmoothWall - razvijan za usmjerivače i vatrozide.
- Tiny Core Linux - razvijan kao minimalni funkcionalni Linux (pr. za virtualna računala, testiranje i sl.).
- Tizen - razvijan uglavnom od strane tvrtke ***Samsung*** uz podršku Linux fondacije, za mobilne i TV uređaje.



Važno je razumjeti i to, da se nazivi softverskih paketa za isti softver, na različitim distribucijama Linuxa mogu (malo) razlikovati u imenu, osim očite razlike u ekstenziji, poput: `.rpm`, `.deb`, `.txz` i drugih!

Pogledajmo i cijelo stablo većine najčešće korištenih i najrasprostranjenijih distribucija *Linuxa (slika* 5.*)*:

**Slika 5. Stablo svih važnijih distribucija Linuxa.**




Lista gotovo svih distribucija *Linuxa*, nalazi se na poveznici: https://en.wikipedia.org/wiki/List_of_Linux_distributions

Lista **aktivnih** distribucija Linuxa (u 2024.g. ih je oko **300**) se može vidjeti na: https://distrowatch.com/.

## 2.7.1. Što je operativni sustav

Prije nego krenemo dalje, bilo bi važno razumjeti što je uopće operativni sustav i čemu on služi.
Operativni sustav (OS) je sistemski softver koji djeluje kao sučelje između komponenti računalnog hardvera i korisnika.
To znači da za hardverske funkcije kao što su primjerice ulazne i izlazne operacije te dodjela i korištenje memorije i uporaba centralnog procesora koji to sve i odrađuje, operativni sustav djeluje kao posrednik između programa i računalnog hardvera.
To znači da svako računalo mora imati instaliran operativni sustav za pokretanje (drugih) programa.

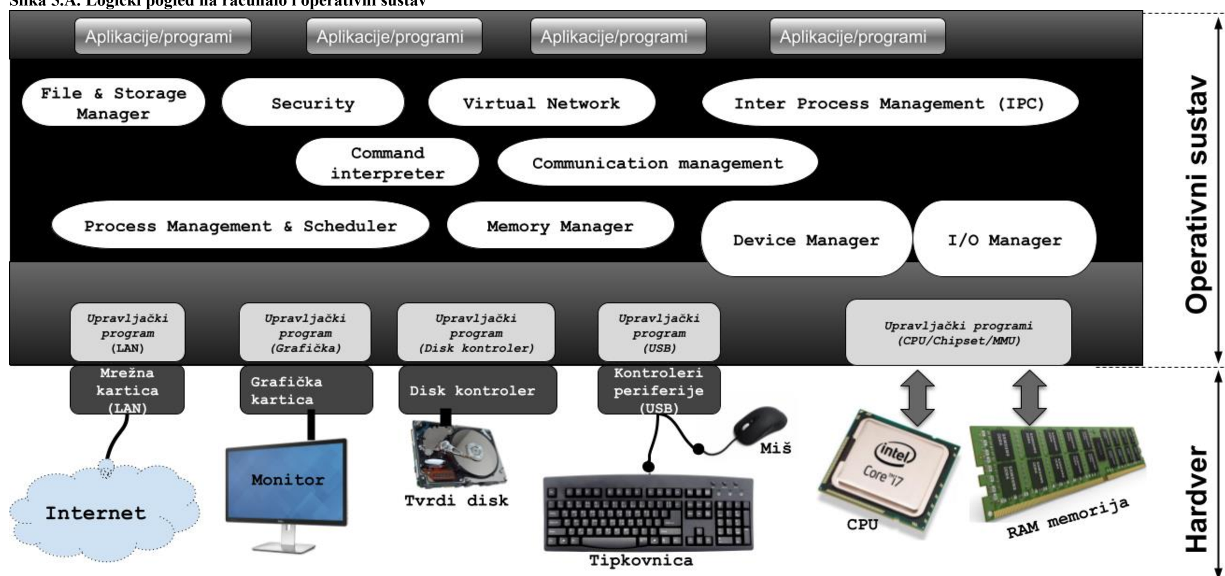
Programi odnosno aplikacije kao što su Web preglednici, uredski paket programa, poput *Libre Office*, *MS Office*, programa za obradu slika (pr. *GIMP*), *Notepad*, igara i drugih, trebaju neko okruženje za pokretanje i izvršavanje svojih zadataka.

Dakle operativni sustav (OS) je sistemski softver koji upravlja računalnim hardverom i softverskim resursima, tako da daje resurse računala (hardvera) i sve potrebne usluge za rad, računalnim programima odnosno aplikacijama.
Pri tome operativni sustav (OS) može biti vrlo mali (pr. *MenuetOS*) ili veliki kao *Microsoft Windows*.
Različiti operativni sustavi mogu se koristiti u različite svrhe. Neki se koriste za svakodnevne stvari poput klasičnog rada na osobnom računalu. Pripadnici ove skupine su primjerice ***Microsoft Windows***, ***Apple MacOS(X)*** i razne distribucije **Linuxa**, kao i mnoge varijante **UNIX**-a. Drugi su mobilni operativni sustavi poput ***Android***-a ili ***iOS***-a, dok se drugi koriste za specijalizirani rad odnosno za posebne namjene, poput specijaliziranih distribucija Linuxa ili UNIX-a, koji rade ne posebnom hardveru (pr. preklopnici, usmjerivači, IoT uređaji, …).

Ako pogledamo malo detaljnije, operativni sustav (*OS*) ima mnogo zadataka. On osigurava da svi programi mogu koristiti centralni procesor (CPU) i (RAM) memoriju sustava, zaslon (monitor) i grafičku karticu, ulazne uređaje (pr. tipkovnicu i miša) te drugi hardver.

On također daje korisniku tekstualno ili grafičko sučelje za korištenje računala.
Operativni sustav je također odgovoran za slanje podataka na druga računala ili uređaje na mreži odnosno za mrežnu komunikaciju.

**Slika 5.A. Logički pogled na računalo i operativni sustav**




Prvi operativni sustav koji je izgledao i donekle radio kao današnji operativni sustavi bio je *UNIX*, koji je razvijen 1969. godine.
Imao je malu jezgru (*kernel*) i mnogo malih programa koji su se mogli povezivati u radu te imati interakciju s korisnikom.
Mnoge njegove značajke preuzete su iz takozvanog *Multicsa*, starijeg operativnog sustava napravljenog 1964. godine.



O povijesti razvoja **UNIX**-a, pogledajte poglavlje:
**2.6.1. Povijest Unixa.**

Ako sliku (5.A) analiziramo još detaljnije, od najniže razine, vidljivo je da operativni sustav pomoću upravljačkih programa (engl. *drivers*) pristupa svom hardveru odnosno komponentama računala:

- Matičnoj ploči: čipsetu i centralnom procesoru (CPU), memorijskom kontroleru i RAM memoriji.
- Diskovnom kontroleru koji potom pristupa diskovima – na kojima koristimo direktorije i datoteke.
- Grafičkoj kartici – za prikaz na monitoru računala.
- Mrežnoj kartici – za pristup mreži i internetu.
- USB kontroleru i periferiji spojenoj na njega: tipkovnica, miš, …

Potom su u drugom (višem) sloju operativnog sustava implementirani mehanizmi (znani i kao kernel servisi) potrebni za interakciju korisnika i računala, kao i aplikacija (programa) i hardvera, ali i razne međuveze i interakcije među njima.

Navedeni mehanizmi (kernel servisi), vidljivi na slici s prethodne stranice (slika **5.A.**), prikazani su kao logičke cjeline (na sredini slike) operativnog sustava. Navedene logičke cjeline čine sloj između aplikacija i ostatka operativnog sustava, a one su:

- ***Process Management & Scheduler*** – ovo je komponenta sustava koja je zadužena za učitavanje i pokretanje programa (takozvanih *procesa*), ali i za njihovo zaustavljanje. Ona pomoću svojih mehanizama brine o točnom vremenskom okviru unutar kojeg se određeni program može pokretati te se brine o prebacivanju izvršavanja s jednog programa na drugi.
- ***Memory Manager*** – ovo je komponenta koja se brine o dodjeljivanju i oduzimanju RAM memorije, kako za korisničke programe, tako i za sistemske servise i druge komponente sustava.
- ***File & Storage Manager*** – ova komponenta upravlja svim aktivnostima vezanim za datoteke (i direktorije) kao što su pohrana, dohvaćanje, imenovanje, dijeljenje i zaštita datoteka. ***Storage Manager*** dio zadužen je za pohranu podataka. Naime operativni sustavi imaju nekoliko razina za pohranu koje uključuju: *primarnu* pohranu, *sekundarnu* pohranu i *predmemoriju*. Razne instrukcije sustava i podaci moraju biti pohranjeni u *primarnoj* memoriji ili *predmemoriji*, kako bi ih pokrenuti programi mogli referencirati odnosno povezati se s njima. Ova komponenta je stoga zadužena za pohranu u širem smislu riječi.
- ***Device Manager*** – ova komponenta upravlja svim (hardverskim) uređajima te prati rad svih uređaja. Ona je zadužena i za otkrivanje novog hardvera te za pronalazak i učitavanja potrebnih upravljačkih programa za hardver.
    - ***I/OManager*** – ovo je logička pod komponenta ***Device Manager***–a, a njena namjena je skrivanje detalja o hardveru, na način da se pristup hardveru na najnižoj razini skriva to jest pojednostavljuje za korisničke (i druge) programe.
- ***Command interpreter*** – ova komponenta tumači naredbe dane od strane sustava te ih obrađuje i povezuje s drugim elementima sustava prema potrebi.
- ***Security*** komponenta brine se o svim sigurnosnim aspektima sustava.
- ***Communication management*** – on je zadužen za koordinaciju i rad s kompilatorima, interpreterima, ali i drugih softverskim resursima različitih korisnika sustava i sâmog sustava odnosno servisa operativnog sustava.
- ***Inter process management (IPC)*** – on je zadužen mehanizme za komunikaciju između programa (procesa).
- ***Virtual Network*** – on je zadužen za mrežnu komunikaciju, kako između programa (procesa) unutar istog računala, tako i za komunikaciju s udaljenim računalima preko mreže.



Za druge detalje o načinu funkcioniranja operativnog sustava Linux, pogledajte poglavlje:
**11. Od kojih komponenti se sastoji Linux.**

## Jezgra (kernel) operativnog sustava

Većina navedenih logičkih cjelina, nalazi se u takozvanom ***kernelu*** odnosno jezgri operativnog sustava. Pri tome razlikujemo nekoliko vrsta odnosno kategorija *kernela*, a to su:

- ***Monolitni*** *kernel* upravlja svim resursima sustava a sastoji se od jednog velikog bloka programskog kôda. Primjerice Linux *kernel*(2020.g.) ima 27.8 milijuna redaka programskog kôda. Kernel pruža sve potrebne usluge odnosno navedene logičke cjeline (slika **5.A.**) koje nudi operativni sustav. Pri ovakvom dizajnu imamo jedan komunikacijski sloj između hardvera i softvera (aplikacija). Pri tome aplikacije komuniciraju s kernelom i sustavom pomoću tzv. ***sistemskih poziva***, što pojednostavljuje razvoj i održavanje. U slučaju greške na nekoj od komponenti unutar ovakvog kernela, cijeli kernel (i sustav) se može srušiti.
- ***Mikro*** *kernel* također upravlja svim resursima sustava. Međutim u ovoj vrsti *kernela*, usluge se implementiraju u različitim adresnim prostorima. Usluge *kernela* pohranjene su u adresnom prostoru *kernela* i uključuju samo osnovne mehanizme to jest upravljanje adresnim prostorom niske razine, upravljanje programskim nitima i komunikaciju među procesima. Korisničke usluge su pohranjene u korisničkom adresnom prostoru, a one obavljaju sve preostale funkcije, uključujući i sekundarne servise (pr. IPC), ali i upravljačke programe. Primjerice mikro *kernel MINIX 3* OS-a ima samo 12,000 linija programskog kôda, što povećava učinkovitost i sigurnost. Osim **MINIX OS**-a, ***Mikro kernel*** arhitekturu koriste i: ***GNU Hurd***, ***QNX*** i ***Symbian*** te inačice **macOS** i **IOS** koje su bazirane na **Mach** kernelu. Ovakav dizajn je malo kompleksniji što se tiće održavanja i razvoja, ali je potencijalno stabilniji: pr. rušenje upravljačkog programa ili drugih servisa koji nisu unutar sâmog kernela, ne uzrokuje veće probleme na sustavu.
- ***Hibridni*** *kernel* je kombinacija prethodna dva principa odnosno dizajna. Ovakav dizajn koriste primjerice: MacOS i **Windows NT**, **Huawei Harmony OS**, te inačice **macOS** i **IOS** koje su bazirane na **Darwin** kernelu.



**Linux** koristi **Monolitni** kernel, kao i: ***FreeBSD***, ***OpenBSD***, ***NetBSD*** te SunOS (**Solaris**), **AIX**, **BSD UNIX, Windows 10** i noviji, **Android** i **ChromeOS** !



Za druge detalje o načinu rada **kernela**, te komunikaciju aplikacija s kernelom (i međusobno) pogledajte poglavlja:
**9.2.1. Napredno o procesima i signalima** 🡐 cjelinu **"Signali i sistemski pozivi".**
**9.3. Task/process scheduler.**
**9.4. Komunikacija između procesa.**
**11. Od kojih komponenti se sastoji Linux.**

## Podjela operativnih sustava

Operativni sustavi se dijele na sljedeće kategorije.

**Jednozadaćni (*Single-tasking*) i višezadaćni (*Multi-tasking*) sustavi te dijeljenje vremena**

Jednozadaćni su sustavi koji su mogli istovremeno izvoditi samo jedan program (zadatak). S druge strane višezadaćni operativni sustav može pokrenuti više od jednog programa u isto vrijeme. Višezadaćnosti odnosno sposobnost sustava da može pokrenuti više od jednog programa istovremeno se odrađuje dijeljenjem vremena (Engl. *Time sharing*) procesora na izvršavanje, između više zadataka (programa).

Pri tome centralni procesor (*CPU*) svakom programu daje samo dio svog vremena za izvršavanje odnosno obradu podataka. Operativni sustavi koji dijele vrijeme za obradu, planiraju točno vrijeme obrade svakog zadatka odnosno daju mu vremenski okvir za izvršavanje, kako bi učinkovito iskorištavali sustav.

Oni rade tako da koriste posebne mehanizme za alokaciju procesorskog vremena, sustava za pohranu, ispis i druge resurse računala.

**Jedno korisnički (*Single-user*) i više korisnički (*Multi-user*) sustavi**

Jednokorisnički operativni sustavi ne mogu razlikovati korisnike odnosno ne mogu imati više korisnika na sustavu, ali u pravilu mogu dopustiti istovremeno pokretanje više programa.

Višekorisnički operativni sustav dopušta rad više korisnika i njihovu interakciju sa sustavom u isto vrijeme.

**Distribuirani sustavi**

Distribuirani operativni sustavi upravljaju grupom računala tako da rade i *ponašaju* se kao jedno računalo. Oni se najčešće koriste za složene i hardverski zahtjevne izračune (pr. simulacije), koje se zbog ograničenja hardvera ne mogu izvoditi na jednom fizičkom računalu, već se izračuni praktično raspodjeljuju to jest distribuiraju na više fizičkih računala koja se nalaze unutar distribuiranog sustava.

**Ugrađeni (*Embedded*) sustavi**

Ugrađeni operativni sustavi dizajnirani su za korištenje u ugrađenim računalnim sustavima. Dizajnirani su za rad na hardverski slabijim računalima (uređajima) s manje autonomije rada, kao što su primjerice: GPS sustavi, ugrađeni sustavi za multimediju i druge namjene u automobilima, IoT sustavi, *pametni* satovi, moderne pećnice, zamrzivači te druga bijela tehnika i slično.

Njihova osnovna značajka je da su vrlo kompaktni i iznimno učinkoviti, te mogu raditi s ograničenom količinom hardverskih resursa. *QNX Unix, Windows CE, Minix 3* kao i mnoge optimizirane distribucije Linuxa (pr. *Yocto Linux*) neki su od primjera ugrađenih operativnih sustava.

**Operativni sustavi u stvarnom vremenu**

Operativni sustav u stvarnom vremenu je operativni sustav koji jamči obradu događaja ili podataka koji imaju kritično definirana vremenska ograničenja. Operativni sustav u stvarnom vremenu može imati jedan ili više zadataka, ali kada obavlja više zadataka, koristi specijalizirane algoritme za planiranje kako bi se postigla deterministička priroda izvršavanja zadataka. Takav sustav vođen je takozvanim događajima (engl. *events*), a on se prebacuje između zadataka na temelju njihovih prioriteta ili vanjskih događaja, dok operativni sustavi s dijeljenjem vremena prebacuju zadatke na temelju vremenskih okvira u kojima se izvršavaju.

## Zastupljenost (primjena) operativnih sustava

Pogledajmo i tržišnu zastupljenost raznih operativnih sustava (2021.g.), ovisno o njihovoj namjeni:

| **Web poslužitelji** | |
|---|---|
| *Linux* | **77,4%** |
| *Windows* | **22,7%** |
| *UNIX:*<br>• *BSD*<br>• *Darwin*<br>• *Solaris*<br>• *Minix* | **< 1%** |

| **Osobna računala/Laptopi** | |
|---|---|
| *Windows* | ***87,56%*** |
| *MacOS* | ***9,54%*** |
| *Linux* | ***2,35%*** |
| *ChromeOS* | ***0,41%*** |
| *Nepoznato* | ***0,13%*** |
| *BSD** | ***0,01*** |

| **Mobiteli** | |
|---|---|
| *Android* | **71,24%** |
| *iOS* | **28.26%** |
| *Nepoznato* | **0,40%** |
| *Linux* | **0,05%** |
| *Series 40* | **0,03%** |
| *Windows Phone OS* | **0,01%** |

**Televizori i tableti**

**Televizori**

| | |
|---|---|
| *Android* | **31,1%** |
| *WebOS* | **22,8%** |
| *Tizen* | **15,4%** |
| *Fire TV* | **19,9%** |

**Tableti**

| | |
|---|---|
| *iOS* | **55,55%** |
| *Android* | **44,23%** |
| *Linux* | **0,13%** |

| **Superračunala (TOP500)** | |
|---|---|
| *Linux* | **100%** |

| **Embedded** | |
|---|---|
| *Linux* | **38,42%** |
| • *Windows10*<br>• *Windows CE7* | **10,83%** |
| • *QNX*<br>• *LynxOS* | **2,82%** |

| **Mainframe** | |
|---|---|
| *z/OS* | **72%** |
| *Linux* | **28%** |

**Izvori informacija: (967),(968),(969),(1107),(1108),(1109),(1110),(1111),(1112),(1114),(1680),(1681).**

# 3. Upoznajmo se s Linuxom

## 3.1. Unix/Linux ljuska *(Shell)* i terminali

Unix odnosno Linux ljuska (engl. *shell*) je znana i kao **naredbeni redak**, a u kojemu kolokvijalno rečeno, upisujemo naredbe, pokrećemo programe i razne skripte. Dakle kao korisnik unošenjem naredbi ili pokretanjem programa, imamo direktnu interakciju s računalom, upravo preko *shella* odnosno skraćeno govoreći *ljuske*. Korisnici obično stupaju u interakciju s Unix/Linux *shellom* odnosno *ljuskom*, pomoću emulatora terminala, ako se radi o udaljenom pristupu, ili izravnim radom ispred računala kada smo spojeni na lokalnu konzolu (tipkovnica/miš/monitor) odnosno takozvani lokalni terminal.
Važno je razumjeti da u Linuxu možemo raditi u tekstualnom okruženju u kojem ćemo raditi cijelo vrijeme u ovoj knjizi te u grafičkom radnom okruženju (***X Window sustav***), koje ćemo kasnije samo kratko spomenuti.

Međutim na prvim *Unix* operativnim sustavima, a čiji nasljednik je i GNU/*Linux*, korisnici su se na poslužitelje, koji su bili velika specijalizirana računala, spajali preko takozvanih *terminala*. Prvi *terminali* su bili posebni pisâći, jer monitori kao izlazni uređaji još nisu postojali.

Ovakav pristup preko *terminala* se na engleskom zove ***TeleType*** ili skraćeno **TTY**.
Kod ovih sustava ulazna jedinica je bio sustav za bušene kartice, a danas je to obično tipkovnica. Kasnije se pod nazivom ***terminal*** podrazumijevalo jednostavno (minimalno) računalo s monitorom i tipkovnicom, spojeno direktno na udaljeni poslužitelj, obično nekom međuvezom poput serijskog ili sličnog sučelja.

**Slika 5.1. VT100 terminal**




**Izvor fotografije: https://upload.wikimedia.org/wikipedia/commons/9/99/DEC_VT100_terminal.jpg**

Slika 5.2. program mc s vidljivim znakovima za iscrtavanje okvira (rubova)

```
 Left        File        Command        Options        Right
┌<─ /etc ──────────────.[^]>┐┌<─ /proc ──────────────.[^]>┐
│.n Name     │Size │Modify tim││.n Name     │Size │Modify tim│
│/prel~nf.d  │   24│v 19 04:25││ cmdline    │    0│v 30 07:33│
│/profile.d  │ 4096│v 19 04:24││ consoles   │    0│v 30 07:34│
│/ras        │   27│g 16 06:39││ cpuinfo    │    0│v 30 07:33│
│/rc.d       │  127│t 11 03:59││ crypto     │    0│v 30 07:34│
│~rc0.d      │   10│t 11 03:59││ devices    │    0│v 30 07:33│
│~rc1.d      │   10│t 11 03:59││ diskstats  │    0│v 30 07:33│
│~rc2.d      │   10│t 11 03:59││ dma        │    0│v 30 07:34│
│~rc3.d      │   10│t 11 03:59││ exec~ains  │    0│v 30 07:34│
│~rc4.d      │   10│t 11 03:59││ fb         │    0│v 30 07:34│
│~rc5.d      │   10│t 11 03:59││ file~tems  │    0│v 30 07:33│
│~rc6.d      │   10│t 11 03:59││ inte~upts  │    0│v 30 07:33│
│/rdma       │   38│t 11 18:24││ iomem      │    0│v 30 07:34│
│/requ~ey.d  │   53│v  9 09:42││ ioports    │    0│v 30 07:34│
│/rhsm       │   24│g 16 06:39││ kallsyms   │    0│v 30 07:33│
│/rpm        │   25│v  9 09:34││ kcore      │ 128T│v 30 07:33│
├──────────────────────────────┤├──────────────────────────────┤
│-> rc.d/rc4.d                 ││ filesystems                  │
└──────────── 10G/13G (77%) ───┘└──────────────────────────────┘
Hint: Want your plain shell? Press C-o, and get back to MC
[root@localhost proc]#                                  [^]
 1Help  2Menu  3View  4Edit  5Copy  6Re~ov 7Mkdir 8De~te
```

Autor fotografije: Jason Scott

Na slici 5.1. je vidljiv video terminal **VT100**, kojeg je u kolovozu 1978. predstavila tvrtka *Digital Equipment Corporation* (**DEC**). Bio je to jedan od prvih *terminala* koji je podržavao **ANSI** (*escape)* kodove za kontrolu kursora i druge zadatke, te je dodao niz proširenih kodova za posebne značajke kao što je upravljanje statusnim svjetlima na tipkovnici. **VT100** je uveo i znakove za crtanje okvira (Engl. *Box/Line drawing characters*). Oni su oblik grafike koja se koristi u tekstualnom korisničkom sučelju za iscrtavanje različitih geometrijskih okvira, kako je vidljivo na slici 5.2.

Naime na slici 5.2 vidimo uporabu znakova za iscrtavanje okvira koje koristi program `mc`, koji je vizualni upravitelj/preglednik datoteka. **VT100** terminal se s poslužiteljem spajao sa serijskom vezom brzine do 19.200 bitova u sekundi (***bps***). Sve navedeno je dovelo do brzog preuzimanja **ANSI** standarda, koji je postao *de facto* standard za emulatore terminala. Uslijedila je novija inačica terminala **VT200** (**VT220**), 1983 godine, a potom su uslijedile i druge. Danas se pod pojmom *terminal* podrazumijeva sučelje za pristup ljusci (*shellu*), s kojom komuniciramo pomoću tipkovnice, a njen izlaz gledamo na našem monitoru (*ekranu*). U suvremenom kontekstu su svi *terminali* zapravo emulatori *terminala* (primjerice **VT100**, **VT200**, **Linux** i drugih) jer se radi o softveru koji emulira rad nekadašnjih *terminala*. Nakon prijavljivanja na sustav, sustav nam dodjeljuje određeni *terminal* za rad, što ćemo vidjeti ubrzo.

Vratimo se na Unix/Linux ljuske. Naime sve Unix/Linux ljuske osiguravaju mnoge funkcionalnosti poput rada s datotekama i direktorijima te upotrebu posebnih znakova nad njima. Potom osiguravaju zamjenu naredbi, takozvane *unix* cjevovode (*piping*) i preusmjeravanje, upotrebu varijabli, kontrolnih mehanizama te upravljačke strukture za ispitivanje stanja odnosno programske uvjete kao i iteracije te mnoge druge funkcionalnosti.

Drugim riječima *ljuska* je sučelje između korisnika i operativnog sustava, iz nje pokrećemo razne programe, ali i upravljamo cijelim sustavom. Možemo reći i sljedeće: *ljuska* je program koji prima naredbe od korisnika i daje ga Linuxu na obradu, a potom prikazuje izlazni rezultat tih naredbi na našem monitoru (*ekranu*).

Naredbe u *ljusci* su programi koje izvršavamo odnosno pokrećemo, a one mogu biti: zasebne binarne izvršne datoteke (programi) ili one koje su već ugrađene u ljusku (pr. u ***bash*** *ljusku*), ali i dostupne kao druge izvršne datoteke poput primjerice *shell* skripti. Ljuska odnosno *Shell* se ponekad naziva i ***CLI*** (engl. *command language interpreter*) ili tekstualno sučelje.
Sâmih *ljuski* postoji poveći broj, od kojih svaka ima svoje specifičnosti. Neke od češće korištenih *ljuski* su:

- ***Bourne shell*** (*sh)* i ***Bourne-Again shell*** (*bash*).
- ***Z shell*** (*zsh*) i ***C-shell*** (*csh*), kao i ***T Shell*** (*tcsh*) i ***Korn shell*** (*ksh*).

Mi ćemo dalje u radu govoriti o ***bash*** ljusci jer se ona najviše koristi, a i nudi nam mnoge napredne mogućnosti.

**Izvori informacija: (1068),(1069),** `man tty`,`man 7 term`,`man 5 shells`, **poglavlje 6.2.2.**

### 3.1.1. Prijavljivanje na sustav

Prije prijavljivanja na sustav važno je znati da je Linux višekorisnički sustav, što znači da omogućava istovremeni rad više korisnika. Pri tome svaki korisnik mora imati svoj korisnički račun. Međutim, kako bismo se uopće mogli prijaviti odnosno spojiti (*logirati*) na sustav, vaš korisnički račun (*engl. account*) mora biti kreiran od strane administratora.
Korisnički račun identificira korisnika, a sastoji se od korisničkog imena s kojim se identificiramo te njegove pripadajuće lozinke. Administrator se u *Unix* odnosno Linux terminologiji zove ***root*** korisnik ili super korisnik (*engl. super user*).
Važno je znati da se tijekom prijavljivanja na sustav (tzv. *logiranja*) razlikuju velika i mala slova.
Vaše korisničko ime, kao i pripadajuća lozinka osjetljivi su na velika i mala slova. Osim toga važno je znati i sljedeće:

- Minimalna dužina imena korisničkog računa je u pravilu šest znakova.
- Lozinka se mijenja s naredbom `passwd`.
- Izlaz iz sustava se ostvaruje naredbama `exit` ili `logout`, ali i kombinacijom tipki: `CTRL d`.

Nakon što se tek prijavimo na sustav (*logiramo*) sustav, dodjeljuje nam se određeni terminal. Ako smo spojeni lokalno na računalo (*monitor + miš + tipkovnicu*), dobivamo pristup na neki od lokalnih terminala (`tty1`,`tty2`,`tty3`, …).
Stoga provjerimo koji smo terminal dobili za rad, pomoću naredbe `tty`:

```
tty
/dev/tty1
```

Pošto vidimo da se ovdje radi o terminalu `/dev/tty1`, znamo da smo direktno spojeni na računalo. Da smo se primjerice udaljeno spojili na računalo, preko SSH protokola, koristio bi se neki od pseudo terminala, poput: `pts/0`, `pts/1` itd.
Potom će nas sustav ubaciti u naš početni korisnički direktorij (mapu) koji se naziva i kućni, radni ili *home* direktorij, a u kojem imamo puna prava kreiranja i rada s datotekama i direktorijima. Standardno se početni korisnički direktorij nalazi unutar vršnog direktorija: `/home/`. Primjerice za korisnika imena `ivan` početni radni direktorij (*radna mapa*) je `/home/ivan/` unutar kojega on može smještati svoje datoteke, mijenjati njihove ovlasti (prava pristupa), kreirati direktorije (mape) te pôddirektorije kao i smještati svoje konfiguracijske datoteke i slično. Važno je razumjeti da na ovaj način svaki korisnik ima svoje izolirano radno okruženje, izolirano od drugih korisnika, a koje može prilagođavati svojim potrebama. Ovisno o ljusci u kojoj korisnik radi, moguće je unutar našeg početnog odnosno kućnog (radnog) direktorija kreirati konfiguracijske datoteke u kojima možemo definirati pravila i postavke za našu ljusku (*naredbeni redak*) koja vrijede samo za nas; nâs kao korisnika identificiranog s našim korisničkim računom.
U konfiguracijskoj datoteci `/etc/shells` pobrojane su sve ljuske koje se mogu koristiti na sustavu.



Pogledajte i poglavlje: **7.1.1. Osnovne naredbe za rad s korisničkim računima i grupama.**
Za detaljni proces prijavljivanja na sustav pogledajte i: **7.1.6.1. Što se događa tijekom prijavljivanja (logiranja) na sustav.**

**Izvori informacija: (95),(96),(817), (K-12), (K-13), `man bash`,`man tty`,`man 4 tty`,`man 4 pts`,`man shells`.**

---

### 3.1.2. Osnove rada ljuske *(shella)*

U ovoj cjelini upoznati ćemo se s osnovama rada u *bash* ljusci o kojoj ćemo detaljnije govoriti malo kasnije. Dodatno, nije loše i znati kako se unutar svake ljuske, pa tako i ***bash*** ljuske nalaze integrirane i razne dodatne naredbe. Sa sljedećim nîzom naredbi možemo vidjeti koje su sve naredbe integrirane odnosno već se nalaze u *bash ljusci* koju ćemo nadalje koristiti.
Pokrenimo naredbu `compgen` na sljedeći način (uz sve što piše u sljedećem retku) i dobit ćemo listu ugrađenih naredbi:

```
compgen -b | egrep -v "\:|\.|\[" | column
alias       compgen     enable      getopts     logout      readonly    times       unset
bg          complete    eval        hash        mapfile     return      trap        wait
bind        compopt     exec        help        popd        set         true
break       continue    exit        history     printf      shift       type
builtin     declare     export      jobs        pushd       shopt       typeset
caller      dirs        false       kill        pwd         source      ulimit
cd          disown      fc          let         read        suspend     umask
command     echo        fg          local       readarray   test        unalias
```

Opis svake od ovih integriranih naredbi unutar *bash* ljuske, možemo dobiti s naredbom `man` poput primjera dolje u kojemu tražimo upute za naredbu `compgen` s naredbom **`man compgen`** ili s naredbom **`man bash`.**
Nakon što smo pokrenuli navedene naredbe, dobiti ćemo upute za ovu naredbu (`compgen`). Pogledajte i animaciju

Ako za neku naredbu želimo saznati je li ugrađena u ljusku (*Engl. Shell builtin*) ili dolazi kao zaseban program, to možemo provjeriti upotrebom naredbe `type`. Pogledajmo primjer za naredbu `compgen`.

```
type compgen
compgen is a shell builtin
```

Dakle u ovom slučaju vidimo da je navedena naredba ugrađena u ljusku jer vidimo poruku: "`is a shell builtin`".
Glavnina sistemskih naredbi poput: `cat`, `chmod`, `chown`, `cp`, `cut`, `dd`, `df`, `du`, `echo`, `ln`, `mkdir`, `mv`, `rm`, `sort` i drugih, dolazi u takozvanom *softverskom paketu* imena ***coreutils***. Za njihov opis i listu, pokrenite: **`info coreutils`**.

**Izvori informacija: (95),(96),(817),(1212),(K-12),(K-13),(K-14), `man bash`,`man compgen`,`info coreutils`.**

---

## 3.1.3. Rad s korisničkim računom

Važno je znati i kako svaki korisnički račun s kojim se prijavljujemo (*logiramo*) na sustav ima neke svoje specifičnosti:

- Svaki korisnik ima svoj osobni korisnički radni direktorij (*mapu*), obično unutar direktorija `/home/`.
- Svaki korisnički račun ima svoje ime i pripadajući identifikacijski broj (***ID)*** koji se označava kao ***UID*** *(user ID)*.
- Svaki korisnički račun pripada barem jednoj korisničkoj grupi.
    - Svaka korisnička grupa ima svoje ime i identifikacijski broj koji se označava kao ***GID*** *(group ID)*.

Nakon što sjednemo za Unix/Linux računalo ili se udaljeno spojimo na njega, primjerice preko ***ssh*** ili ***telnet*** protokola za udaljeni pristup, prvo ćemo se morati prijaviti na sustav i to tako da će nas sustav zatražiti naše korisničko ime, što ćemo vidjeti ovako:

```
Server1 login:
```

Ovdje ćemo morati upisati naše korisničko ime; primjerice `root` (*administrator*) i potvrditi ga s tipkom *ENTER*.

Potom će nas sustav tražiti lozinku za korisnika s kojim se pokušavamo prijaviti na sustav, poput poruke:

```
Password:
```

Ovdje trebamo upisati lozinku za pripadajućeg korisnika te ju također potvrditi s tipkom *ENTER*.

Sada kada smo se uspješno prijavili na sustav, možemo provjeriti pripadnost našeg korisničkog računa grupi odnosno grupama korisnika, s naredbom `id -a`.
Stoga pokrenimo naredbu `id` na sljedeći način:

```
id -a

uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon)
```

Ovdje vidimo kako smo se prijavili na sustav kao korisnik imena `root` čiji identifikacijski broj (***UID***) je `0` te kako on:

- Pripada primarnoj grupi imena `root` čiji identifikacijski broj (***GID***) je `0`
- Pripada i dodatnim grupama (*groups*): `bin` čiji ***GID*** broj je `1` i grupi `daemon` čiji ***GID*** broj je `2`

O ovim detaljima (korisničkim grupama) ćemo govoriti znatno kasnije.



Za više detalja o korisničkim računima pogledajte poglavlje:
**7.1.Rad s korisničkima računima, grupama i lozinkama**

Normalni (obični) korisnici sustava imaju svoje korisničke direktorije unutar direktorija `/home/` tako primjerice korisnik `ivan` standardno ima svoj korisnički direktorij `/home/ivan/` dok administrator odnosno korisnik `root` ima svoj korisnički direktorij (mapu) koji se nalazi u zasebnom direktoriju (mapi): `/root/`.
Dakle administrator (***root***) je odvojen od drugih korisnika. Nakon što se korisnik prijavi na sustav (*logira*), sustav ga odmah ubacuje u njegov korisnički radni direktorij (mapu) koji se naziva i kućni ili početni direktorij korisnika.

**Izvor informacija: (96),(817),(K-1),(K-12),(K-13), `man bash`, `man id`.**

## 3.1.4. Osnovne kontrolne naredbe

U radu s ljuskom (*shellom)* možemo koristiti osnovne kontrolne naredbe koje aktiviramo kombinacijom sljedećih tipki, poput:

- `CTRL a` - skoči na početak reda.
- `CTRL e` - skoči na kraj reda.
- `CTRL c` - prekini izvršavanje naredbe.
- `CTRL d` - odlogiraj se sa sustava.
    - `CTRL d` - još znači i pošalji „***End of File***" (*EOF*) trenutnom procesu.
- `CTRL l` - obriši ekran.
- `CTRL z` - zaustavlja (stopira) izvršavanje trenutno pokrenute naredbe/programa.
- `CTRL ALT Fn` - prebaci se na lokalni **n**-ti terminal (**n**=1…6), ako se nalazite direktno na računalu (ne udaljeno):
    - `CTRL ALT F1` – prebacujemo se na prvi lokalni terminal (`/dev/tty1`).
    - `CTRL ALT F2` – prebacujemo se na drugi lokalni terminal (`/dev/tty2`).
    - `CTRL ALT F3` – prebacujemo se na treći lokalni terminal (`/dev/tty3`), … .
- `ALT U` - promijeni sve riječi iza trenutne pozicije kursora u velika slova.
- `ALT L` - promijeni sve riječi iza trenutne pozicije kursora u mala slova.

**Izvori informacija: (96),(817),(K-12),(K-13), `man bash`.**

### 3.1.5. Bash ljuska detaljnije

***Slijedi napredna cjelina!***
*Bash shell* (`bash`) je *unix ljuska* napisana za ***GNU*** projekt kao zamjena za *Bourne shell* (`sh`) u 1989. godini. Široko se koristi i danas je standardna ljuska za razne inačice Unixa, poput ***Mac OS X***, ***FreeBSD***, ali i ***Linuxa***. Prebačena je i na *Microsoft Windows*, *Novell NetWare* i *Android*. Možemo reći i da je *bash* naredbeni procesor koji se pokreće u tekstualnom prozoru odnosno u ***terminalu*** te omogućava korisnicima pokretanje raznih Unix/Linux naredbi. *Bash* također može čitati naredbe iz datoteka, zvanih *shell* skripte. Tijekom prijavljivanja korisnika na sustav (*logiranja*) vrlo je vjerojatno da ćete koristiti upravo ***bash*** ljusku. Definicija raznih parametara koji se čitaju tijekom prijavljivanja korisnika na sustav u zadanom (***interaktivnom***) načinu rada, za *bash* ljusku se nalazi u globalnim, ali i datotekama specifičnim za svakog korisnika, koje se učitavaju sljedećim redoslijedom:

1. `/etc/profile` datoteka je koja sadrži definicije rada *ljuske* za sve korisnike. Dakle ona je globalna za sve.

Naime globalnu konfiguracijsku datoteku `/etc/profile` konfigurira administrator sustava (*root*) i tijekom prijavljivanja na sustav ona se prva učitava. Varijable i opcije postavljene u ovoj datoteci mogu se promijeniti u datoteci `.bashrc` samo, ako nisu postavljene s ovlastima sâmo za čitanje (Engl. *readonly*) od strane administratora. Dakle moguće je konfigurirati određene vrijednosti varijabli, koje se definiraju u `/etc/profile` da budu postavljene sâmo za čitanje, a ne i mijenjanje odnosno postavljene kao „*readonly*“, tako da ih korisnici u svojoj konfiguraciji (pr. datoteci `.bashrc`) ne mogu zaobići ili mijenjati.


Za više detalja o varijablama, pogledajte poglavlje: **6.2.2 Sistemske (Environment) varijable.**

Potom se učitava konfiguracijska datoteka specifična za svakog pojedinog korisnika, iz njegovog početnog (*home*) direktorija i to prva od sljedećih, koja postoji:

2. `~/.bash_profile`
3. `~/.bash_login`
4. `~/.profile`

Znak **~** označava *početni/kućni (home)* direktorij prijavljenog korisnika (odnosno vâs kada se prijavite na sustav).

U navedenim konfiguracijskim datotekama: `/etc/profile`, `~/.bash_profile`, `~/.bash_login` ili `~/.profile` definiraju se sistemske varijable, opcije i parametri za rad naše ljuske. Objasnit ćemo kratko njih sâmo nekoliko (redak po redak):

```
USER="`id -un`"
MAIL="/var/spool/mail/$USER"
```

To znači kako će se sistemska varijabla `MAIL` u kojoj se definira ime datoteke u koju će se spremati *e-mail* poruke za korisnika postaviti tako da ona bude locirana unutar direktorija `/var/spool/mail/` i da se zove isto kao i ime korisničkog računa.

```
HISTSIZE=1000
```

Postavlja veličinu memorije za spremanje izvršenih naredbi (tzv. *history*) na 1.000 unosa (o tome kasnije).


Za više detalja pogledajte poglavlje: **3.1.7. Pamćenje izvršenih naredbi (*history*).**

```
umask 022
```

Postavlja `umask` vrijednost na `022`, što znači kako će svaki novo kreirani direktorij (mapa) ili datoteka dobiti određene ovlasti odnosno prava: čitanja, zapisivanja ili izvršavanja.


Za više detalja pogledajte poglavlje: **4.4.1. Ovlasti kod kreiranja nove datoteke ili direktorija i** *naredba* **umask.**

Pogledajmo i sljedeće unose koje ovdje možemo susresti:

```
export LANG=POSIX
export LC_CTYPE=en_US.UTF-8
```

Postavlja regionalne postavke na gore navedene (regionalne postavke na POSIX, Engleski – US te **UTF-8**).


Za više detalja o regionalnim postavkama pogledajte poglavlje: **10.4.4. Regionalne postavke odnosno *Locale*.**

Ponekada su postavljene i sljedeće varijable koje su zadužene za prikaz naredbenog retka:

- `PS1` varijablom definira se prikaz; primjerice: `root@Server1 :~$`. U navedenom primjeru se koristi sljedeća postavljena vrijednost varijable: `PS1='[\u@\h \W]\$ '`. Neke od vrijednosti koje možemo postaviti ovdje su:
  - `\u` - označava da će se prikazati ime trenutno prijavljenog korisnika (pr. `root`).
  - `\h` - označava da će se prikazati ime računala (*hostname*).
  - `\w` - označava da će se prikazati puna putanja do trenutnog direktorija u kojem se nalazimo dok se s velikim slovom `W` definira kako će se prikazivati samo zadnji trenutni direktorij u kojem se nalazimo.
- `PS2` varijablom definira se prikaz kod svakog novog reda, standardno je postavljena na `>`, poput: `PS2='> '`

Standardno postavljena `PS1` varijabla je:

```
PS1='[\u@\h \W]\$ '
```


Za više detalja o drugim sistemskim varijablama pogledajte poglavlje:
**6.2.2. Sistemske (*Environment*) varijable i postavke terminala**.

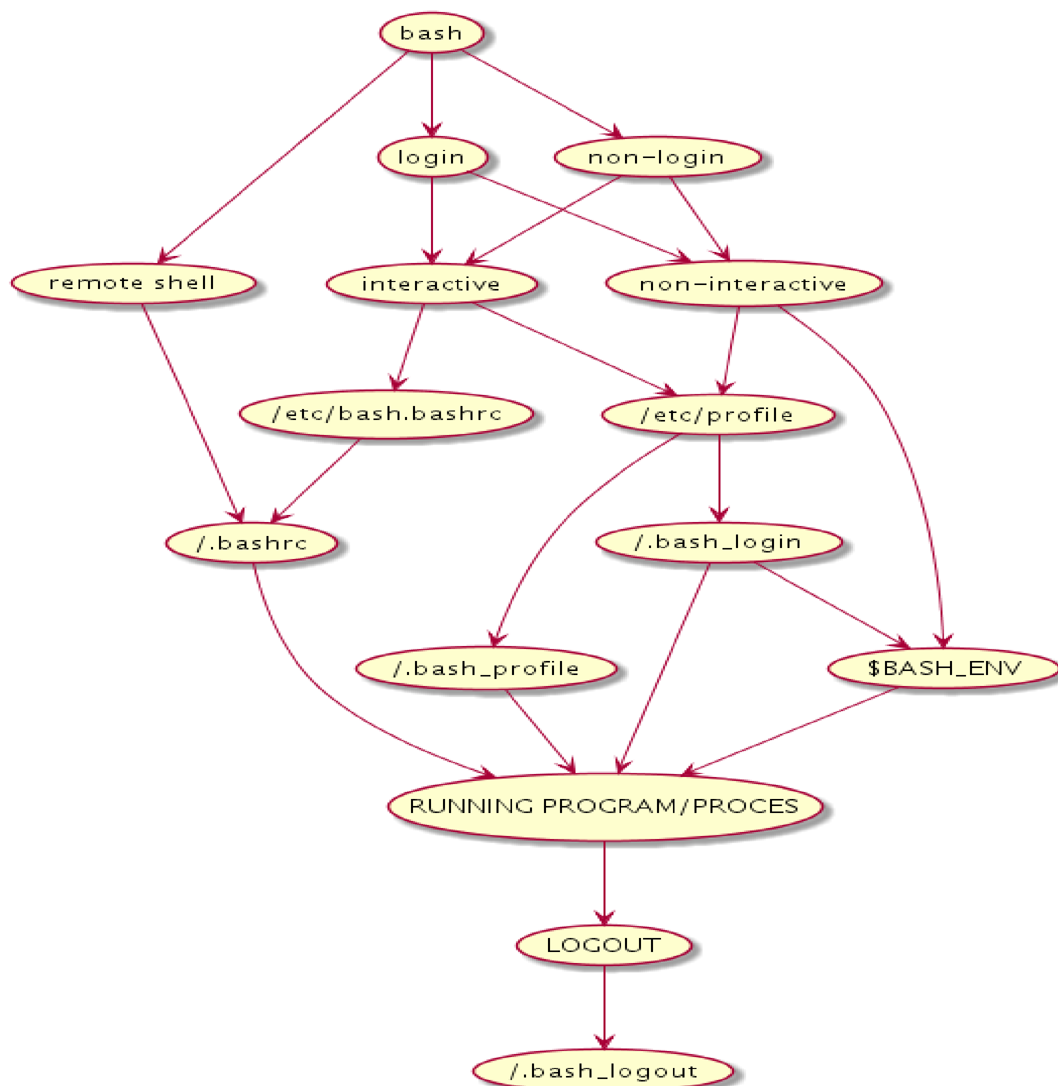
Osim toga postoji i cijeli nîz drugih opcija i varijabli te parametara koji se mogu postaviti u ovim konfiguracijskim datotekama.

Sada pogledajmo kako izgleda *ljuska* odnosno naredbeni redak, kada se nalazimo u našem korisničkom direktoriju odnosno mapi (što je vidljivo sa znakom ~):

```
root@Server1 :~$
```

Ovdje ćemo se detaljnije upoznati s konfiguracijskim *bash* datotekama i njihovim pozivanjem te izvršavanjem u pozadini.
Sve ovisno o tome jesmo li se spojili preko udaljenog pristupa (*ssh/telnet* ili slično) ili direktno; prijavljivanjem na sustav ili pokrećemo li ga iz neke skripte. Stoga su vidljivi sljedeći ***načini**** rada ljuske i datoteke koje se pritom učitavaju (slika 6.).

**Slika 6. Bash ljuska.**




- Ako se spajamo s udaljenom metodom pristupa, primjerice preko udaljenih klijenata poput ***ssh** ili **telnet***, preko mreže, kada se pokreće *bash ljuska*, kao na slici, što je vidljivo kao `remote shell` tada se prvo čita datoteka `~/.bashrc`.
Ona se učitava iz vašeg kućnog (*home)* direktorija odnosno početnog direktorija korisnika koji se tek spojio na sustav.
- Ako se ipak spajamo lokalno na računalo, te se također pokreće *bash ljuska*, slijed učitavanja konfiguracijskih *bash* datoteka je vidljiv od točke `login` pa na dalje.
- Treća mogućnost je takozvani `non-login` način rada koji se događa kada se neki program ili skripta pokreću odnosno kada oni pokreću *bash ljusku* za svoj rad (unutar sebe).

**Nadalje, i `non-login` i `login` metode korištenja *bash ljuske* mogu biti:**

- Interaktivne (`interactive`) - ovo je normalan način rada u kojem ili mi ili *shell* skripta rade u interakciji ili s nama ili s nekim drugim programom ili skriptom.
- Ne interaktivne (`non-interactive`) metode su malo rjeđe i one su obično ograničene za posebne skripte, koje nemaju nikakvu direktnu interakciju s drugim skriptama ili korisnikom.

I na sâmom kraju, kada se odlogiramo ili kada se odspojimo s udaljenim pristupom (preko mreže), učitava se zadnja konfiguracijska datoteka imena: `~/.bash_logout`. Ova datoteka se učitava iz našeg korisničkog direktorija, ako ta konfiguracijska datoteka uopće postoji.

 Vezano za postavke ***načina**** rada ljuske pogledajte poglavlje: **5.12.2. Drugi napredni primjeri upotrebe *bash* ljuske.**

**Izvor informacija: (96),(817),(1201),(K-1),(K-12),(K-13),(K-14), `man bash`.**

### 3.1.6. Pseudonimi naredbi (*Aliasi*)

Pseudonimi naredbi odnosno *aliasi* se koriste za dodjeljivanje nîza naredbi nekoj novoj (*alias*) naredbi. Možemo ih zamisliti kao postavljanje nama razumljive naredbe koja u pozadini poziva cijeli nîz naredbi, zbog bržeg/lakšeg rada. Dakle moguće je postaviti *alias* naredbu, čijim pokretanjem, zapravo pokrećemo neku drugu naredbu: drugu naredbu s nîzom prekidača, opcija i parametara ili cijeli nîz ulančanih naredbi. Aliasi se koriste zbog jednostavnosti rada u *ljusci*. Naime ponekad pozivamo određene naredbe s cijelim nîzom prekidača, opcija i parametara, za čije tipkanje treba dosta vremena. Ako takve naredbe koristimo više puta, jednostavnije je od njih napraviti pseudonim odnosno *alias*, te njega pozivati. Pogledajmo primjere.

**1.** Stvorimo pseudonim (*alias*) imena `ll` koji će pokrenuti naredbu `ls -alhi`.
Vidljivo je kako se alias naredbe definiraju korištenjem naredbe `alias` definicijom unutar jednostrukih apostrofa: `''`

```
alias ll=´ls -alhi´
```

Za izlistanje svih postavljenih aliasa koristi se naredba `alias` bez opcija ili prekidača.

**2.** Izlistajmo sve trenutno postavljene *aliase* (na kraju je vidljiv i naš novo kreirani):

```
alias

alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias ll='ls -alhi'
```

Pogledajte animaciju:

Za brisanje *aliasa* koristi se naredba `unalias`. Sada pomoću nje primjerice obrišimo alias `ll` na sljedeći način:

```
unalias ll
```

**Izvor informacija: (96),(817),(K-1),(K-12),(K-13), `man bash`.**

### 3.1.7. Pamćenje izvršenih naredbi (*history)*

Od trenutka prijavljivanja korisnika na sustav, a to je i trenutak kada se pokreće ljuska (*shell*), pamte se sve naredbe koje smo pozivali odnosno pokretali. Stoga je naredbe koje smo pozivali moguće:

- Pozvati ponovno.
- Pretraživati.

Naredbom `history` izlistavamo sve naredbe koje smo pozivali u prošlosti. Pozovimo ovu naredbu kako bismo vidjeli koje smo sve naredbe pozivali odnosno pokretali u prošlosti, od trenutka *logiranja* na sustav s našim korisničkim računom:

```
history
```

Pogledajte animaciju:

Dodatno je moguće definirati i koliko zadnjih naredbi će sustav pamtiti, sa: `set HISTSIZE = xxx`, a pri tome podešavamo koliko zadnjih naredbi (`xxx`) će sustav zapamtiti.

**Primjer**
*History size* je obično postavljen na pamćenje zadnjih 1.000 naredbi, a to ćemo povećati na 5.000 upotrebom naredbe `set`:

```
set HISTSIZE = 5000
```

Još neke od korisnih mogućnosti *history* sustava su:

- `!123` - pozovi naredbu iz povijesti (*History*) pod rednim brojem `123`
- `!ls` - pozovi zadnju naredbu iz povijesti (*History*) koja počinje sa `ls`
- `CTRL r` - pretraži *history* odnosno listu svih naredbi koje smo upisali, pri tome ovo važi za *bash ljusku*.
  Nakon ovoga se upisuje pojam/naredba koja se traži iz povijesti pokrenutih naredbi *(historyja).*

**Izvor informacija: (96),(817),(K-1),(K-12),(K-13), `man bash`.**

## 3.2. Tko je sve logiran na sustav i što radi ?

S vremena na vrijeme želimo saznati tko je sve prijavljen (*logiran*) na sustav odnosno u ljusku (*shell)* i što sve radi.
Naime važno je razumjeti kako na Linux računalu ili poslužitelju može raditi više korisnika istovremeno odnosno više njih može biti spojeno (*logirano*) na sustav istovremeno. Stoga nam je ponekada potrebno provjeriti tko je sve *logiran* na sustav i eventualno, koje je naredbe ili programe pokrenuo određeni (pojedini) korisnik. To možemo saznati pomoću sljedećih naredbi:

- `who` – prikazuje nam tko je sve logiran na sustav, uz još neke dodatne informacije.
- `w` – prikazuje tko je logiran na sustav te koje je sve procese/programe pokrenuo.
- `last` – prikazuje tko je sve bio logiran na sustav.

U narednim cjelinama upoznat ćemo se s navedenim naredbama kroz primjere koji slijede.

### 3.2.1. Naredba *who*

Naredba `who` daje nam detaljne informacije o korisnicima prijavljenima (*logiranim*) na sustav.
Najčešći prekidači ove naredbe su sljedeći:

- `-a` – ispiše nam sve detalje o korisnicima logiranim na sustav.
- `-b` – ispiše samo informaciju kada se operativni sustav pokrenuo (startao).
- `-m` – ispiše samo kombinaciju imena računala (*hostname*) i korisničkog imena (*username*).
- `-r` – ispiše trenutni *Runlevel*. Za Runlevele pogledajte poglavlje: ***7.3.* Stanja rada Linuxa (*Runlevels*)**.
- `-u` – ispiše koji su sve korisnici prijavljeni na sustav (*logirani*).

**Primjeri**:
1. Ispišimo sve dostupne podatke (prekidač `-a`) o korisnicima spojenim na sustav (ispis smo malo skratili):

```
who -a

           system boot  2014-07-31 20:05
           run-level 3  2014-07-31 20:05
root     + tty1         2014-07-31 20:05 00:02        3294
root     + pts/0        2014-07-31 20:17   .          1199 (10.0.2.2)
ucenik   + pts/1        2014-07-31 20:38   .          1419 (10.0.2.2)
```

U ispisu je vidljivo sljedeće:

- Da je operativni sustav pokrenut `31.07.2014` u `20:05.h.`
- Da je trenutni *Runlevel* broj `3`. O tome znatno kasnije u poglavlju: **7.3 Stanja rada Linuxa (*Runlevels*)**.
- Da imamo sljedeće prijavljene (logirane) korisnike; a navest ćemo ih od gore prema dolje:
  - `root` je spojen preko lokalnog terminala `tty1` (onoga koji predstavlja fizičku vezu na računalo).
  - Zatim su vidljivi datum i vrijeme (treći stupac: pr. `2014-07-31 20:17`) kada su se korisnici prijavili.
  - `root` i `ucenik` spojeni su preko pseudo terminala `pts/0` i `pts/1`. Ovdje se radi se o udaljenom pristupu, preko ***ssh*** protokola pa su vidljive i njihove IP adrese (`10.0.2.2`), za oba korisnika konkretno.
  - Za sve prijavljene (logirane) korisnike imamo i broj procesa (programa) odnosno ***PID*** brojeve u predzadnjem stupcu, a koji identificiraju pokrenuti program; pr. ***PID*** `3294` za `root` korisnika fizički spojenoga na računalo. O procesima ćemo pričati kasnije u poglavlju: **9. Proces menadžment.**

2. Ispišimo sâmo tko je sve prijavljen (*logiran*) na sustav:

```
who -u

root       tty1         2014-07-31 20:17 00:05        3294
root       pts/0        2014-07-31 20:17   .          1199 (10.0.2.2)
ucenik     pts/1        2014-07-31 20:38 00:01        1419 (10.0.2.2)
```

**Izvor informacija: (K-1),(K-12), `man who`.**

---

### 3.2.2. Naredba *w*

Naredba `w` daje nam informacije o tome koje programe (procese) je pokrenuo koji od prijavljenih korisnika na sustav.
Samim pokretanjem (bez argumenata) naredbe `w` dobivamo informacije o svim *logiranim* korisnicima te programima (procesima) koje su oni pokrenuli. Pogledajmo i primjere dolje, te animaciju:

**1.** Ispišimo sve korisnike logirane na sustav te programe (procese) koje su pokrenuli pomoću naredbe `w`:

```
w

20:56:20 up 51 min,  2 users,  load average: 0.00, 0.00, 0.00

USER     TTY     FROM        LOGIN@   IDLE    JCPU   PCPU   WHAT
root     pts/0   10.0.2.2    20:17    0.00s   0.47s  0.03s  w
ucenik   pts/1   10.0.2.2    20:38    3.00s   0.11s  0.06s  /usr/bin/mc -P /tmp/mc …
```

**Opis:** Vidimo kako je korisnik `root` pokrenuo naredbu `w` (to smo mi). Također je vidljivo da je korisnik `ucenik` pokrenuo naredbu `mc` (*Midnight Commander*), uz sve detalje o procesu i vremenu pokretanja naredbe/procesa kao i pripadajuće IP adrese s koje se korisnik spojio, ako se spojio s mreže putem *SSH* ili nekog drugog protokola za udaljeni pristup.

**2.** Ispišimo podatke o pokrenutim procesima samo za korisnika `ucenik`

```
w ucenik

21:00:44 up 55 min,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY    FROM        LOGIN@ IDLE  JCPU   PCPU  WHAT
ucenik   pts/1  10.0.2.2    20:38  4:27  0.11s  0.06s /usr/bin/mc -P /tmp/mc-ucenik/mc.pwd.1423
```

Vidimo da se korisnik `ucenik` spojio preko mreže i da koristi virtualni terminal `pts/1`, koliko procesorskog vremena koristi (`JCPU` i `PCPU`), kao i koju naredbu je pokrenuo (`/usr/bin/mc`) te kada (`20:38`) i s koje IP adrese se spojio (`10.0.2.2`).

**Izvor informacija: (K-1), `man w`.**

### 3.2.3. Naredba *last*

Naredba `last` daje nam informacije o tome tko se sve prijavio na sustav i kada. Pokretanjem naredbe bez argumenata, dobivamo informacije o tome s kojim korisničkim računima je ostvareno uspješno *logiranje* na sustav, kako lokalno tako i s udaljenim pristupom; primjerice spajanjem preko ***telnet*** ili ***ssh*** protokola preko mreže. Zapravo se sve statistike dohvaćaju iz sistemske *log* datoteke u koju se upisuju sve statistike vezane za logiranje/prijavljivanje na sustav. To je datoteka: `/var/log/wtmp`.

Pogledajmo kako to izgleda, kada pozovemo naredbu `last` (ostavljeno je samo nekoliko statistika zbog jednostavnijeg prikaza):

```
last

root     pts/0        142.158.247.12   Wed Mar 25 21:15 - 22:59  (01:44)
root     pts/0        142.158.247.12   Wed Mar 25 20:12 - 20:55  (00:42)
ivan     pts/0        213.147.100.123  Wed Mar 25 12:12 - 17:06  (04:53)
root     pts/0        213.147.100.123  Wed Mar 25 12:02 - 12:07  (00:05)
pero     pts/0        213.147.100.123  Wed Mar 25 11:57 - 12:00  (00:02)
root     pts/0        213.147.100.123  Wed Mar 25 11:55 - 11:56  (00:01)
```

- U prvom stupcu je ime korisničkog računa koji se spojio na sustav (primjerice: `root`, `ivan`, `pero`).
- U drugom stupcu je ***terminal*** na koji se spojio. Pri tome su `pts` *pseudo* terminali, za udaljeni pristup računalu.
- U trećem stupcu je IP adresa korisnikovog računala, ako se radi o udaljenom pristupu.
- Četvrti pa sve do sedmog stupca označavaju datum i vrijeme od kada do kada je ta veza bila otvorena.
- Zadnji stupac prikazuje ukupno vrijeme rada na sustavu, za pojedinog korisnika.

Moguće je tražiti sâmo statistiku spajanja za određenog korisnika. Pogledajmo sada statistiku za korisnika `root`.
Sada kao prvi parametar naredbi `last` moramo dodati korisnički račun; u ovom slučaju je to korisnički račun korisnika `root`.

```
last root

root     pts/0        142.158.247.12  Wed Mar 25 21:15 - 22:59  (01:44)
root     pts/0        142.158.247.12  Wed Mar 25 20:12 - 20:55  (00:42)
root     pts/0        213.147.100.123 Wed Mar 25 12:02 - 12:07  (00:05)
root     pts/0        213.147.100.123 Wed Mar 25 11:55 - 11:56  (00:01)
```

Pomoću prekidača `-n` moguće je ograničiti samo zadnje unose iz statistike. Ograničimo se na zadnja četiri unosa:

```
last -n4

root     pts/0        142.158.247.12  Wed Mar 25 20:12 - 20:55  (00:42)
root     pts/0        213.147.100.123 Wed Mar 25 12:02 - 12:07  (00:05)
root     pts/0        213.147.100.123 Wed Mar 25 11:55 - 11:56  (00:01)
root     pts/0        207.117.221.78  Sun Mar 22 12:27 - 17:53  (05:12)
```

Moguće je i vidjeti točno vrijeme kada je sve sustav restartan. Pogledajmo kako:

```
last reboot

reboot   system boot  2.6.32-44-pve    Wed Mar 16 21:51 - 13:17 (1+15:25)
reboot   system boot  2.6.32-44-pve    Wed Mar 16 21:20 - 21:46 (00:26)
reboot   system boot  2.6.32-44-pve    Thu Jan 21 15:57 - 21:46 (55+05:48)
reboot   system boot  2.6.32-43-pve    Sat Jan 16 20:25 - 15:52 (4+19:27)
```

**Izvor informacija: (K-1), `man last`.**

## 3.3. Osnovna komunikacija između korisnika

Na gotovo svakom *Unix* ili *Linux* sustavu možete pronaći dvije naredbe (programa) s kojima možete slati tekstualne poruke drugim korisnicima koji su *logirani* na sustav. ***Slijede dvije napredne cjeline!***

### 3.3.1. Naredba *write*

Naredba `write` koristi se za slanje poruke željenom korisniku koji mora biti logiran na sustav.
Upotreba ove naredbe radi se na sljedeći način:
`write` ***`username tty`***

- ***`username`*** je pri tome ime korisnika (korisničkog računa) kojem želimo poslati poruku.
- ***`tty`*** je ime *terminala* na koji je taj korisnik spojen. Ako ne navedemo ime terminala (pr. `pts/1`), poruka će biti poslana na sve *terminale* na koje je taj korisnik spojen (ako se navodi ime korisnika odnosno ***`username`***).

**Primjer:** kao korisnik `root` pošaljimo poruku: "`Ovo je poruka`" korisniku `pero`

```
write pero

Ovo je poruka
```

Kada želimo završiti pisanja poruke, potrebno je stisnuti slijedeću kombinaciju tipki: `CTRL` `d`.

U trenutku kada smo završili s pisanjem, s druge strane korisnik `pero` dobiva sljedeću poruku (pogledajte i animaciju: 🎬):

```
Message from root@localhost.localdomain on pts/1 at 17:33 ...

Ovo je poruka
EOF
```

**Što ako ne želimo primati poruke od drugih korisnika preko naredbe `write`?**
Za tu namjenu postoji naredba `mesg` s kojom svaki korisnik može onemogućiti primanje ovakvih poruka.
Onemogućavanje primanja poruka možemo definirati sa:

```
mesg n
```

Ponovno omogućavanje primanja poruka:

```
mesg y
```

**Kako provjeriti tko je onemogućio primanje poruka?**
Za provjeru tko ima omogućeno, a tko onemogućeno primanje poruka, možemo pokrenuti naredbu: `who -T`.
Pogledajmo kako to izgleda:

```
who -T

root     + tty1         2016-05-23 17:19 (:0)
root     + pts/0        2016-05-23 17:19 (:0.0)
root     - pts/1        2016-05-23 17:20 (:0.0)
pero     + pts/2        2016-05-23 17:26 (192.168.11.42)
```

Vidimo kako je korisnik `root` koji je spojen na pseudo [*virtualni*] terminal `pts/1`, onemogućio primanje poruka jer ima znak `-` ispred imena terminala, dok svi ostali imaju omogućeno primanje poruka (znak `+`).
**Izvor informacija: (K-1), `man write`,`man mesg`,`man who`.**

### 3.3.2. Naredba *wall*

Naredba `wall` (Engl. *Write to All*) koristi se za slanje poruka svim *logiranim* odnosno korisnicima prijavljenim na sustav. Najčešće se koristi za slanje poruke o gašenju cijelog sustava (poslužitelja) i to od strane administrator (`root` korisnika).
Moguća su dva način upotrebe ove naredbe, koja može prihvatiti ulaznu poruku:

- Na standardni ulaz odnosno takozvani ***stdin***.
- Pisanjem poruke slično kao i s naredbom `write`.

Pogledajmo oba primjera. Slanje poruke preko standardnog ulaza. Poruka je "*Gasimo sustav za 5.minuta*".

```
echo "Gasimo sustav za 5.minuta" | wall
```

Dakle sve što šaljemo preko `echo` naredbe dolazi kao standardni ulaz (preko *pipea* **|** ) na naredbu `wall` koja onda taj tekst šalje kao tekstualnu poruku svima. Drugi primjer je slanje poruke poput naredbe `write`. S time da prvo pokrenemo naredbu `wall` (ENTER) pa potom pišemo poruke. Kraj pisanje poruka je isto s kombinacijom tipki `CTRL d`

```
wall

Gasimo sustav za 5.minuta
```

`wall` poruke primaju svi *logirani* korisnici kojima je postavljen: `mesg y`. Odnosno nije (ručno) isključeno primanje poruka, a koje vrijedi i za `write` poruke. Kako bismo se mogli pobliže upoznati sa Linux sustavom na kojem radimo, potrebne su nam i neke osnovne naredbe, pomoću kojih ćemo saznati nešto više o sâmom sustavu.
Također ćemo se upoznati s naredbama koje će nam pomoći da pronađemo potrebne naredbe ili komponente sustava.
**Izvor informacija: (K-1), `man wall`.**

## 3.4. Saznajmo nešto više o Linuxu i Linux naredbama

Na većini *Unix* i *Linux* sustava standardno su instalirane i detaljnije upute za svaku naredbu, koje je moguće i pretraživati.

### 3.4.1. Naredba *man*

Kada želimo pronaći detaljnije upute naredbe koju poznajemo, kao pomoć ćemo koristiti naredbu `man`
Ovakve upute (engl. *manuals*) pozivamo s naredbom `man`.
Upute su podijeljene u nekoliko područja (takozvanih *sekcija*), označenih s brojevima:

1. Korisničke naredbe [engl. *User Commands*].
2. Sistemski pozivi [engl. *System Calls*].
3. Funkcije C biblioteka [engl. *C Library Functions*].
4. Uređaji i posebne datoteke [engl. *Devices and Special Files*].
5. Formati datoteka i konvencije [engl. *File Formats and Conventions]*.
6. Igre i slično [engl. *Games*].
7. Razno [engl. *Miscellanea*].
8. Sistemski servisi (*daemoni*) i alati [engl. *System Administration tools and Deamons*].

Moguće je pretraživati upute za određene naredbe prema definiranim područjima ili bez njih (što je zadano ponašanje).
U slučaju da navodimo i područje, pretraživanje će se vršiti samo unutar definiranog područja to jest brojeva od **1.** do **8.**

Želimo li saznati unutar kojih vršnih direktorija su trenutno pohranjene *man* stranice, pokrenimo naredbu `manpath`:

```
manpath
/usr/local/share/man:/usr/share/man
```

**Primjeri**:

1. Pronađimo sve upute (iz svih područja uputa) za sâmu naredbu `man` (pogledajte animaciju: )

```
man man
```

1.1. Pronađimo upute za naredbu `ls` (naredbu za izlistavanje sadržaja direktorija)

```
man ls
```

2. Pronađimo upute za datoteku `/etc/passwd`, koje se nalaze u području (sekciji) **5** [*formati datoteka i konvencije*]:

```
man 5 passwd
```

3. Pronađimo upute za **TCP** protokol, koje se nalazi u području (sekciji) **7** [*razno*]:

```
man 7 tcp
```

4. Pretražimo `whatis` bazu pomoću `man` naredbe, za ključnu riječ `ls`:

```
man -k ls
```

*Inicijalna **man** baza dolazi u programskom paketu imena* `man-pages`*. Važno je razumjeti i da se instalacijom svake nove naredbe (softverskog paketa) uz sâmu naredbu instalira i njena **man** stranica s uputama.*

**Izvor informacija: (K-1), `man man`,`man manpath`.**

## 3.4.2. Naredbe *whatis* i *apropos*

Naredba `whatis` daje nam samo osnovni opis naredbe za koju tražimo informacije. Osnovni opisi svih naredbi (uz pripadajuću naredbu) se pohranjuju u posebnu bazu podataka; tzv. *whatis database*. Pogledajmo primjer za `man` naredbu:

```
whatis man

man   (1)  - format and display the on-line manual pages
man                   (1p)  - display system documentation
man                   (7)  - macros to format man pages
man [manpath]         (1)  - format and display the on-line manual pages
man-pages             (7)  - conventions for writing Linux man pages
man.config [man]      (5)  - configuration data for man
```

Naredba `whatis` nam je vratila sve opise gdje postoji opis `man` naredbe. Brojevi `(*)` u drugom stupcu ukazuju na poglavlje `man` naredbe u kojemu se nalaze detaljniji opisi. *Whatis* baza se obično nalazi u: `/var/cache/man/whatis` i nju može (re)kreirati samo *root* odnosno administratorski korisnik. *Whatis* baza sadrži sâmo imena svih naredbi uz njihov osnovni opis, izvučen iz svih `man` opisa. U slučaju kada smo instalirali novu naredbu u sustav, uz koju su se instalirale i *man* stranice (upute), potrebno je kreirati i `whatis` unose u *whatis* bazi. To se radi pomoću naredbe `makewhatis` koju može pokrenuti samo *root* korisnik. U *RedHat/Centos 7.x/8+* za tu funkciju je uvedena novija naredba `mandb` koja mijenja naredbu `makewhatis`. Izvršavanje ove naredbe može potrajati neko vrijeme, dok se ne izradi nova baza podataka. Pokrenimo ju ovako:

```
makewhatis
```

Naredba `apropos` nam omogućava pretraživanje bilo koje ključne riječi iz *whatis* baze podataka. U slučajevima kada ne znamo točan naziv neke naredbe, ali znamo što bi ta naredba trebala raditi, možemo pretraživati bilo koji pojam ili ključnu riječ. Potom će nam `apropos` dati popis svih naredbi čiji opisi sadrže ono što tražimo, uz pridruženu naredbu.

**Slijede primjeri**: **1.** Tražimo sve naredbe koje sadrže ključnu riječ: *`pci`* (pogledajte i animaciju: )

```
apropos pci

lspci   (8)  - list all PCI devices
setpci                (8)  - configure PCI devices
update-pciids         (8)  - download new version of the PCI ID list
```

Vidimo da smo dobili ponuđene tri naredbe koje u opisu imaju pojam ili riječ *`pci`*: `lspci`, `setpci` i `update-pciids`

**2.** Zanima nas koja naredba u opisu sadrži točan pojam: "***remove a directory***", jer nas zanima kako obrisati neki direktorij:

```
apropos "remove a directory"

rmdir    (3p)  - remove a directory
unlink   (3p)  - remove a directory entry
unlinkat (2)  - remove a directory entry relative to a directory file descriptor
```

Ovdje smo dobili tri moguće naredbe: `rmdir`, `unlink` i `unlinkat` od kojih nas je zapravo zanimala naredba `rmdir`. Sada možemo potražiti `man` stranicu (upute) od naredbe `rmdir` koja nas je i zanimala:

```
man rmdir
```

**Izvori informacija: `man whatis`,`man apropos`,`man mandb`.**

### 3.4.3. Osnovne naredbe vezane uz operativni sustav i komponente računala

Popis osnovnih sistemskih naredbi, pomoću kojih možemo doznati nešto više o sâmom Linux sustavu te o hardveru na kojem se nalazi je sljedeći:

- `uname` – daje nam osnovne informacije o Linux sustavu.
- `dmesg` – ispisuje sistemske poruke (*HW Events*) prilikom pokretanja sustava, pa na dalje.
- `dmidecode` – ispisuje DMI (*SMBIOS*) informacije o hardveru računala (matična ploča, BIOS, RAM memorija, …).
- `lspci` – ispisuje sve PCI/PCI express uređaje spojene na matičnu ploču računala.

Pogledajmo što nam govore gore navedene naredbe.

**Naredba `uname`** daje nam osnovne informacije o Linuxu. Pokrenimo ju na sljedeći način:

```
uname -a

Linux server.lab.hr 2.6.32-23 #1 SMP Tue Aug 6 07:04:06 CEST 2013 x86_64 x86_64
x86_64 GNU/Linux
```

Pokretanjem naredbe `uname -a` dobijemo nekoliko osnovnih informacija o sustavu:

- `Linux` – vidimo da je ovo Linux (može biti i neka od varijanti UNIX-a).
- `server.lab.hr` – ovo je ime računala (*Hostname*) (`server`) s domenom (`lab.hr`).
- `2.6.32.-23` – ovo je inačica Linux kernela koji je trenutno u upotrebi.
- `SMP` – vidimo da imamo podršku za *Symmetric multi-processing* za više procesora (CPU) ili više jezgri.
- `Tue Aug 6 07:04:06 CEST 2013` – ovo je datum kada je linux kernel kompiliran.
- `x86_64 x86_64` – ovo je arhitektura procesora za koju je kernel kompiliran; konkretno `x86`, `64` bîtan.
- `x86_64` – ovo je arhitektura ovog sustava (x86 kompatibilni: *AMD*, *Intel*, …).
- `GNU/Linux` – ovo je oznaka operativnog sustava (OS), koja konkretno označava: GNU/Linux.

**Naredba `dmesg`**

Naredba `dmesg` daje nam listu poruka koje je snimio sustav, u trenutku pokretanja računala pa na dalje. Pokrenimo ju:

```
dmesg

Initializing cgroup subsys cpuset
Initializing cgroup subsys cpu
Linux version 2.6.32-431.11.2.el6.x86_64 (mockbuild@x86-027.build.eng.bos.redhat.com) (gcc
version 4.4.7 20120313 (Red Hat 4.4.7-4) (GCC) ) #1 SMP Mon Mar 3 13:32:45 EST 2014
Command line: ro root=UUID=fb38dcf6-d76d-4191-b6e7-a5378c4fd6cc rd_NO_LUKS  KEYBOARDTYPE=pc
KEYTABLE=us LANG=en_US.UTF-8 rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=auto rd_NO_LVM
rd_NO_DM rhgb quiet rhgb quiet
KERNEL supported cpus:
  Intel GenuineIntel
  AMD AuthenticAMD
  Centaur CentaurHauls
sd 0:0:0:0: [sda] 585871964 512-byte logical blocks: (299 GB/279 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sda: sda1 sda2 sda3 sda4 <sda5 sda6>
sd 0:0:0:0: [sda] Attached SCSI disk
serial8250: ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
00:08: ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
```

Naredba `dmesg` ispisuje sve poruke sustava, od trenutka pokretanja i učitavanja *kernela* i *kernel modula* (*drivera*) na dalje. Na početku našeg ispisa vidimo inicijalizaciju procesora (CPU), a kasnije i diskova `sd [sda]`, kao i raznih portova te slično. Dakle na početku se inicijalizira cijeli sustav i provjeravaju i inicijaliziraju sve hardverske komponente. Nakon što se sustav pokrene, s naredbom `dmesg` se vide i sve tekuće sistemske poruke; kao što su poruke kada ubacite CD ili USB disk ili o grešci neke komponente sustava i slično. Ako želimo ljepši prikaz vremena poruka, možemo ju pozvati i sa prekidačem `-T`.

**Naredba `dmidecode`**

Naredba `dmidecode` daje nam podatke o našem hardveru. Pokrenimo ju (*skratili smo ispis zbog lakšeg razumijevanja*):

```
dmidecode

Handle 0xD100, DMI type 209, 68 bytes
HP BIOS NIC PCI and MAC Information
NIC 1: PCI device 04:00.0, MAC address D8:9D:68:42:3D:F1
Handle 0x110F, DMI type 17, 34 bytes
Memory Device
Total Width: 72 bits
Data Width: 64 bits
Size: 8192 MB
Form Factor: DIMM
Locator: PROC 2 DIMM 8
Type: DDR3
Type Detail: Synchronous Registered (Buffered)
Speed: 1600 MHz
Manufacturer: HP
Configured Clock Speed: 1600 MHz
. . .
```

Naredba `dmidecode` daje nam ispis statusa svih važnih hardverskih komponenti, koje su standardno podijeljene u nekoliko kategorija, prema nazivima kako ih i možemo koristiti (*nazivi kategorija pišu se malim slovima*):

- ***bios*** – za informacije o ***BIOSu*** (engl. *Basic Input/Output System*) odnosno osnovnom ulazno/izlaznom sustavu.
- ***system*** – za hardverske informacije o sustavu (proizvođač računala/poslužitelja, model i slično).
- ***baseboard*** – za hardverske informacije o komponentama na matičnoj ploči (pr. disk kontroler i mrežne kartice).
- ***chassis*** – za hardverske informacije o kućištu (obično za poslužitelje).
- ***processor*** - za informacije o centralnom procesoru (CPU).
- ***memory*** – za informacije o memorijskom kontroleru (pr. max. memorije koju podržava) te svim memorijskim modulima.
- ***cache*** – za informacije o među memorijama ugrađenim u procesor (CPU); dakle: **L1**,**L2** i **L3** memorijama.
- ***connector*** – za informacije o vanjskim portovima na matičnoj ploči; poput USB, serijskog i paralelnog porta i sl.
- ***slot*** – za informacije o svim dostupnim sabirničkim utorima na matičnoj ploči; poput: ***PCI***, ***PCI express*** i slično.

Ovom naredbom možemo zatražiti ispis stanja svih hardverskih komponenti ili naredbu pozvati s prekidačem `-t` uz specifikaciju gore navedene komponente računala, koja nas konkretno zanima, za koju ćemo potom dobiti informacije.
Primjerice, ako nas zanimaju samo podaci o *BIOSu* matične ploče, to možemo saznati pomoću sljedeće naredbe:

```
dmidecode -t bios
```

```
# dmidecode 2.9
SMBIOS 2.31 present.

Handle 0x0000, DMI type 0, 20 bytes
BIOS Information
        Vendor: ASUS // Phoenix Technologies Ltd.
        Version: 7.00 R4.14.1561.02
        Release Date: 11/25/2013
        Address: 0xE49F0
        Runtime Size: 112144 bytes
        ROM Size: 2048 kB
        Characteristics:
                PCI is supported
                PNP is supported
                APM is supported
                BIOS is upgradeable
                BIOS shadowing is allowed
                ESCD support is available
                Boot from CD is supported
```

**Naredba `lspci`**

Naredba `lspci` daje nam ispis svih uređaja na **PCI** sabirnici, bilo da su integrirani na matičnu ploču ili da se radi o uređajima koji su na zasebnim karticama. Naredba `lspci` dolazi u softverskom paketu `pciutils`.
Pogledajmo primjer njenog pozivanja odnosno pokretanja i ispisa koji nam ona daje:

```
lspci

00:00.0 Host bridge: Intel Corporation Xeon E5/Core i7 DMI2 (rev 07)
00:01.0 PCI bridge: Intel Corporation Xeon E5/Core i7 IIO PCI Express Root Port 1a (rev 07)
00:05.0 System peripheral: Intel Corporation Xeon E5/Core i7 Address Map, VTd_Misc, System
Management 00:11.0 PCI bridge: Intel Corporation C600/X79 series chipset PCI Express
00:1a.0 USB controller: Intel Corporation C600/X79 series chipset USB2 Enhanced Host Controller
#2 00:1c.0 PCI bridge: Intel Corporation C600/X79 series chipset PCI Express Root Port 1
00:1d.0 USB controller: Intel Corporation C600/X79 series chipset USB2 Enhanced Host
01:00.1 VGA compatible controller: Matrox Electronics Systems Ltd. MGA G200EH
?02:01.0 Ethernet controller: Intel Corporation 82547EI Gigabit Ethernet Controller
03:00.0 RAID bus controller: Hewlett-Packard Company Smart Array Gen8 Controllers (rev 01)
04:00.0 Ethernet controller: Broadcom Corporation NetXtreme II BCM57810 10 Gigabit Ethernet
```

Naredba `lspci` ispisuje nam podatke o svim **PCI** sabirnicama i uređajima koji su se prijavili na te **PCI** sistemske sabirnice, bilo da se radi o klasičnoj **PCI**, **PCI-X** ili najnovijoj **PCI Express** sabirnici i uređajima na njoj.
Ovdje je vidljiv dobar dio hardvera samog računala. Osim osnovnog pozivanja ove naredbe, moguće je istu naredbu pozvati i sa prekidačem `-v` ili s `-vv`, ako želimo još više detalja o hardveru. To možemo postići ovako:

```
lspci -v
```

ili ovako

```
lspci -vv
```

Ako želimo i identifikacijske podatke o uređajima (takozvane **PCI ID** brojeve/identifikatore), to možemo dobiti dodavanjem prekidača `-nn`. Odnosno naredbu možemo pokrenuti primjerice ovako:

```
lspci -vv -nn
```



Pogledajte i poglavlja:
**10.2. (Re)konfiguracija sabirnice i uređaja na njoj** te: **11.1.2.1. Uređaji (*devices*) detaljnije.**
**10.5.1. IRQ (*Interrupt request*)** odnosno u cjelinu: **MSI (Message Signaled Interrupts)**
Vezano za instalaciju softverskih paketa, pogledajte poglavlje:
**7.2.2.1. Rad s *YUM-om*.**

U ovom slučaju pogledajmo detaljnije samo mrežnu karticu `INTEL 82547EI` (*skratili smo ispis*):

```
lspci -vv
```

```
02:01.0 Ethernet controller: Intel Corporation 82547EI Gigabit Ethernet Controller
        Subsystem: Fujitsu Technology Solutions Device 101e
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR-
        FastB2B- DisINTx-
        Status: Cap+ 66MHz+ UDF- FastB2B- ParErr- DEVSEL=medium>TAbort-<TAbort-<MAbort->SERR-
<PERR-INTx+
        Latency: 0 (63750ns min), Cache Line Size: 32 bytes
        Interrupt: pin A routed to IRQ 18
        Region 0: Memory at e0100000 (32-bit, non-prefetchable) [size=128K]
        Region 2: I/O ports at 3000 [size=32]
        Capabilities: [dc] Power Management version 2
                Flags: PMEClk- DSI+ D1- D2- AuxCurrent=0mA PME(D0+,D1-,D2-,D3hot+,D3cold+)
                Status: D0 NoSoftRst- PME-Enable- DSel=0 DScale=1 PME-
        Kernel driver in use: e1000
        Kernel modules: e1000
```

Ovdje vidimo i koji se kernel modul (upravljački program) koristi za naš **PCI** uređaj odnosno mrežnu karticu: `Kernel modules: e1000`. Dakle upravljački program za našu konkretnu mrežnu karticu (sučelje) se zove `e1000`.

**O uređajima i sabirnicama nešto kasnije, u naprednijim poglavljima!**
Ako uz informaciju koji se kernel modul (upravljački program) koristi za određeni **PCI** uređaj, želimo vidjeti i mogućnosti (`Capabilities`) određenog **PCI** uređaja, možemo koristiti i proširenje ove naredbe (gledamo **PCI** uređaj: `02:01.0`):

```
lspci -vnk -s 02:01.0
```

Potom ćemo dobiti nešto poput (filtrirali smo ispis):

```
        Capabilities: [48] Power Management version 3
        Capabilities: [50] Vital Product Data
        Capabilities: [58] MSI: Enable- Count=1/8 Maskable- 64bit+
        Capabilities: [a0] MSI-X: Enable+ Count=17 Masked-
        Capabilities: [ac] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [13c] Device Serial Number 00-00-e0-db-55-0f-98-70
        Capabilities: [150] Power Budgeting <?>
        Capabilities: [160] Virtual Channel
        Capabilities: [230] Transaction Processing Hints
```

Iz čega vidimo sve hardverske mogućnosti i opcije naše mrežne kartice.

**Iz kojih sve datoteka i s kojim naredbama možemo vidjeti na kojoj inačici i distribuciji Linuxa se nalazimo?**
Naime bez obzira na kojoj distribuciji Linuxa radili: ***Debian***, ***Ubuntu***, ***RedHat***/***CentOS***, ***OpenSuSe*** ili nekoj drugoj, za sve njih su zajedničke datoteke u kojima se zapisuju podaci o točnoj inačici i imenu distribucije Linuxa na kojem radimo.
Tako imamo datoteku `/etc/os-release` ili neku od datoteka imena: `/etc/`***`XY`***`-release` pri tome ***XY*** ovisi o distribuciji Linuxa, pa to može biti: `os`, `system`, `redhat`, `centos` i slično. Pogledajmo datoteku `/etc/os-release`

```
cat /etc/os-release
```

```
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"
CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
```

Iste podatke nam može dati i naredba `hostnamectl` (za ***CentOS/RedHat*** 7.x ili noviji), stoga ju pogledajmo/pokrenimo:

```
hostnamectl
```

```
   Static hostname: server-1
         Icon name: computer-desktop
           Chassis: desktop
        Machine ID: 1a07b95a79414b8db99d4fa31053a1dc
           Boot ID: d4a34c90b35e4f8e8209cfcd43d8fede
  Operating System: CentOS Linux 7 (Core)
            Kernel: Linux 4.18.14-1.el7.elrepo.x86_64
      Architecture: x86-64
```

Ovime dobivamo iste podatke kao i iz prethodnog primjera, samo malo ljepše formatirane.

**Izvori informacija: (99),(100),(101),(102),(K-12),(K-14), `man uname`,`man dmesg`,`man dmidecode`, `man lspci`,`man hostnamectl`.**

# 4. Datotečni sustav (*File System*)

Datotečni sustav (engl. *file system*) je zadužen za organizaciju, upravljanje i pohranu podataka na diskovima i drugim medijima. Datotečni sustav je u konačnici zadužen za rad s datotekama i direktorijima (*mapama*), odnosno zadužen je za njihovo smještanje na disk i njihovu dalju upotrebu. Dakle na datotečni sustav, a na kraju i na disk, zapisuju se datoteke i direktoriji, s kojima i koje koristi i sâm operativni sustav, kao i svi programi na sustavu, što uključuje i nas kao korisnike sustava (računala).

## Povijest razvoja datotečnih sustava

Razvoj datotečnih sustava još od vremena razvoja prvih UNIX-a (1969.g.) do modernih Linuxa odvijao se kroz nekoliko desetljeća, prateći evoluciju samih operativnih sustava, hardvera, ali i potreba korisnika. Ovaj razvoj može se podijeliti u ključne faze, koje obuhvaćaju nastanak temeljnih koncepata na UNIX-u, te kasnije inovacije i prilagodbe u svijetu Linuxa.
Pogledajmo kako su se datotečni sustavi razvijali kroz povijest.

**Počeci UNIX-a i rani datotečni sustavi (1970.g.)**
UNIX, razvijen krajem 1960-ih i početkom 1970-ih u *Bell Labsu*, donio je temeljne ideje o organizaciji datoteka i direktorija koje su utjecale na sve kasnije datotečne sustave.

Neki rani, ali ključni sustavi iz tog vremena su bili:

- **Originalni UNIX datotečni sustav (1971.g.)**
  - Osnovni koncepti – prvi UNIX datotečni sustav (koji nije imao ime) podržavao je jednostavnu hijerarhijsku strukturu direktorija s datotekama. Svaka datoteka imala je numerički identifikator (***inode***) i bila je organizirana u blokove.
  - Ograničenja – sustav je bio vrlo jednostavan, s maksimalnom veličinom datoteke od 64 KB i primitivnim rukovanjem diskom.

- **UNIX V6 i V7 File System (1975 - 1979.g.)**
  - **UNIX V6 File System** - donio je koncept ***inode*** strukture koja pohranjuje *metapodatke* o datotekama (vlasništvo, prava pristupa, veličina, vremenske oznake). V6 također uvodi direktorije kao specijalne datoteke koje pohranjuju popis imena i ***inode*** brojeva.
  - **UNIX V7 File System** – uveden krajem 1970-ih, unaprijedio je prethodni sustav povećanjem maksimalne veličine datoteke i ukupnog broja datoteka.

- **BSD i FFS (1980-te godine**.) - UNIX je bio otvoren za modifikacije i poboljšanja, pa su mnogi fakulteti i istraživački instituti proširili njegove mogućnosti. Posebno se ističe *Berkeley Software Distribution* (**BSD**), koji je donio mnoge inovacije. Neke od njih su:
  - ***Fast File System*** (**FFS**) - BSD (1983.g.) - *FFS* je stvoren u BSD inačici UNIX-a (4.2BSD) i bio je veliko poboljšanje u odnosu na stari UNIX V7 datotečni sustav. FFS je postao *de facto* standard za mnoge UNIX sustave tijekom 1980-ih i temelj za mnoge buduće datotečne sustave, Novosti koje je uveo su:
    - *Efikasnije alociranje prostora* – uveo je cilindre (cilindarske grupe) i bolju raspodjelu blokova kako bi smanjio fragmentaciju diska te poboljšao performanse.
    - *Podrška za veće datoteke* – povećanjem maksimalne veličine datoteka i ukupnog broja datoteka.
    - *Simbolički linkovi* - omogućena je uporaba simboličkih linkova (engl. *symbolic links*), što je poboljšalo fleksibilnost datotečnih sustava.

  - ***FFS i nastanak UFS-a***
    - *Unix File System* (**UFS**) je evoluirao iz FFS-a i često se koristi kao opći termin za datotečne sustave temeljene na *FFS-u*. *UFS* je zapravo standardizirana verzija *FFS-a* s dodatnim značajkama i optimizacijama. *UFS* je postao zadani datotečni sustav u mnogim inačicama BSD-a i u komercijalnim UNIX varijantama kao što su ***SunOS*** i ***Solaris***. Ključne značajke ***FFS-a*** koje je naslijedio i ***UFS*** su:
      - Hijerarhijska struktura direktorija s podrškom za simboličke linkove.
      - *Inode* struktura za pohranu *metapodataka* o datotekama.
      - Cilindarske grupe (*cylinder groups*) za bolju distribuciju podataka i smanjenje fragmentacije diska.
      - *Journaling* (kod UFS-a na *Solarisu*) – ***Solaris*** je dodao UFS *journaling* kako bi osigurao bolju otpornost na padove sustava, dok izvorni FFS nije imao tu značajku.
    - ***UFS 2*** - UFS2 je poboljšana inačica UFS-a uvedena u *FreeBSD UNIX-u*. Dodaje podršku za veće datoteke i poboljšava rad sa 64-bitnim adresiranjem.

- ***Linux i EXT serija datotečnih sustava (1990 godine)***:
  - ***Extended File System (EXT) – 1992.g.*** – prvi izvorni Linuxov datotečni sustav *EXT*, razvijen je 1992. godine. Bio je inspiriran FFS-om iz BSD-a, ali je imao brojna ograničenja. Primjerice maksimalna veličina datoteke bila je samo 2 GB, a maksimalna veličina particije također samo 2 GB.
  - ***Extended File System 2 (EXT2) – 1993.g.*** – *EXT2* je značajno poboljšao svojeg prethodnika. Postao je vrlo popularan i dugo je bio zadani Linuxov datotečni sustav. Njegove osnovne značajke su:
    - Proširena su ograničenja maksimalne veličine datoteka i particija.
    - Uveo je tzv. *bitmape* za praćenje slobodnog prostora, što je poboljšalo efikasnost.
    - EXT2 je bio tzv. *non-journaling* datotečni sustav, što je značilo da je bio brz, ali manje otporan na kvarove.

- ***Extended File System 3 (EXT3) – 2001.g.*** – uveo je *journaling* te kompatibilnost s *EXT2*.

- ***Extended File System 4 (EXT4) – 2008.g.*** – uveo je mnoga poboljšanja, podršku za veće datoteke (16TB) i veće particije (1EB). Uvodi tzv. ***extent*** umjesto tradicionalne blok-strukture, što je smanjilo fragmentaciju. Podržava i brzo montiranje i provjeru konzistencije podataka. Naime *EXT4* koristi tzv. ***uninit_bg*** značajku, koja omogućava brzo provjeravanje datotečnog sustava nakon pada sustava.

- **2000-te godine na dalje** – uslijedio je razvoj:
  - ***XFS*** (2001.g.) – ***XFS*** je visoko skalabilan datotečni sustav prvotno razvijen za operativni sustav IRIX, a kasnije je prebačen i na Linux. Popularan je u scenarijima koji zahtijevaju rad s velikim datotekama i velikim količinama podataka. I on podržava *journaling*, velike datoteka (do 8EB), i velike particije (8EB), te koristi efikasnu dinamičku alokaciju prostora (tzv. *delayed allocation*) koja omogućuje efikasnije korištenje diskovnog prostora i poboljšava performanse. Iako XFS izvorno ne podržava *snapshote (snimku stanja u vremenu)*, moguće ih je koristiti u kombinaciji s Linux *LVM-om*.

  - ***Btrfs*** (2009.g.) – ***Btrfs*** je moderan datotečni sustav razvijen kako bi ponudio napredne značajke kao što su *snapshoti*, kompresija i integrirano upravljanje RAID-om. On koristi *copy-on-write* (***COW***) mehanizam, što znači da se promjene ne upisuju odmah u iste blokove na disku, nego se prvo stvara nova kopija podataka, što poboljšava integritet podataka. *Btrfs* ima ugrađenu podršku za RAID konfiguracije, uključujući RAID 0, RAID 1 i RAID 10, bez potrebe za posebnim RAID kontrolerom te podržava transparentnu kompresiju datoteka. Korištenjem *provjernog* zbroj (*checksum*) za *metapodatke* i podatke, *Btrfs* može detektirati i ispraviti oštećenja podataka.

  - ***ZFS*** – on je izvorno razvijen za *Sun Solaris UNIX*, ali je kasnije prebačen na Linux kao projekt otvorenog kôda. Poznat je po robusnoj zaštiti podataka i naprednim funkcijama upravljanja diskom. Poput *Btrfs*-a, *ZFS* koristi **COW** mehanizam za bolju zaštitu podataka i eliminaciju potrebe za *defragmentacijom*. ZFS omogućava stvaranje gotovo neograničenog broja *snapshota* i klonova datotečnog sustava, što je korisno za backup i testiranje. ZFS ima ugrađeni sustav za upravljanje diskovima koji omogućuje korištenje različitih RAID konfiguracija, s poboljšanim zaštitnim mehanizmima za redundanciju podataka. Osim toga on koristi *checksume* kako bi detektirao i ispravio oštećene podatke. Kada otkrije oštećenje, automatski ih ispravlja koristeći redundantne kopije. ZFS podržava i takozvanu *deduplikaciju* podataka, što smanjuje prostor na disku eliminacijom dupliciranih blokova podataka.

  - **F2FS** (*Flash-Friendly File System*) – on je posebno dizajniran za *flash* memorije (***SSD***, ***eMMC***), kako bi bolje iskoristio karakteristike tih uređaja i produžio njihov vijek trajanja. *F2FS* je dizajniran tako da minimizira broj zapisivanja na disk, čime smanjuje trošenje SSD diskova. To produžuje životni vijek *flash* memorija. Osim toga on koristi tzv. *zone-aligned* strukturu koja je optimizirana za rad s logičkim blokovima na SSD diskovima. *F2FS* je razvijen da minimizira fragmentaciju podataka na disku, čime se poboljšava ukupna brzina pristupa podacima

Razvoj datotečnih sustava od prvih UNIX-a do modernog Linuxa obilježen je stalnim tehnološkim inovacijama. Razvoj je krenuo s jednostavnim, no učinkovitima sustavima, a danas imamo napredne datotečne sustave poput ***XFS***, ***Btrfs*** i ***ZFS***, koji nude ne samo pouzdanu pohranu podataka, već i napredne značajke za upravljanje i zaštitu. Linux je s vremenom postao platforma koja podržava razne vrste datotečnih sustava, ovisno o potrebama korisnika, što ga čini vrlo prilagodljivim operativnim sustavom za različite svrhe, od osobnih računala do velikih poslužitelja.

**Osnovne značajke datotečnih sustava**

Osnovne značajke koje se očekuju od svih datotečnih sustava na Linuxu odnose se na performanse, stabilnost, sigurnost i fleksibilnost u upravljanju podacima. Iako različiti datotečni sustavi mogu imati dodatne specifične značajke, svi bi trebali osigurati osnovne funkcionalnosti koje omogućuju sigurno i učinkovito rukovanje datotekama.

Najvažnije značajke koje bi svaki moderni datotečni sustav na Linuxu trebao imati su sljedeće:

1. **Podrška za osnovne operacije nad datotekama** – datotečni sustav mora omogućiti temeljne operacije s datotekama i direktorijima, uključujući:
   a. Stvaranje, brisanje, preimenovanje i kopiranje datoteka i direktorija.
   b. Čitanje i zapisivanje podataka unutar datoteka.
   c. Premještanje datoteka i direktorija unutar datotečnog sustava.

2. **Hijerarhijska struktura direktorija –** svi datotečni sustavi na Linuxu koriste hijerarhijski sustav direktorija. To znači da korisnici mogu organizirati datoteke u direktorije (*mape*) koje mogu sadržavati druge direktorije i datoteke, omogućujući strukturiranu organizaciju podataka.

3. **Metapodaci o datotekama** – svi datotečni sustavi trebali bi pohranjivati osnovne *metapodatke* o datotekama, uključujući:
   a. Veličinu datoteke.
   b. Vlasništvo (vlasnika i grupu) nad datotekom i direktorijem.
   c. Dozvole odnosno *ovlasti:*
      i. Standardne ovlasti (***read***/***write***/***execute***) za vlasnika, grupu i sve ostale korisnike.
      ii. *Access Control Lists* (***ACL***) – mnogi moderni datotečni sustavi omogućuju korištenje ACL-ova za detaljniju kontrolu pristupa, omogućujući određivanje specifičnih dozvola za različite korisnike ili grupe. Dok oni napredniji podržavaju i još finiju kontrolu (pr. ***atributi*** i ***capabilities*** značajke).
   d. Vremenske oznake o stvaranju, izmjenama i pristupa datotekama.
   e. Različite vrste datoteka, poput: obične datoteke, direktorija, simboličkog linka, specijalne datoteke i sl.

4. **Podrška za velike datoteke i particije** – moderna uporaba diskovnih sustava zahtijeva da datotečni sustavi podržavaju velike datoteke i particije (pr. minimalno veličine terabajta [TB] ili više).
5. ***Journaling*** – većina modernih datotečnih sustava koristi *journaling* kako bi se spriječio gubitak podataka u slučaju pada sustava. *Journaling* omogućava zapisivanje promjena prije nego što se stvarno upišu na disk, čime se smanjuje mogućnost oštećenja datoteka i ubrzava oporavak nakon kvarova.
6. ***Integracija s LVM-om*** (*Logical Volume Manager*) – mnogi Linux sustavi koriste *LVM* za fleksibilno upravljanje diskovnim particijama. Datotečni sustav mora biti kompatibilan s *LVM-om*, omogućujući jednostavno proširenje ili smanjenje prostora na disku bez ponovnog formatiranja ili gubitka podataka.
7. **Podrška za simboličke i hard linkove** – simbolički (*soft*) i *hard linkovi* su ključne značajke za Linux datotečne sustave pa se očekuje mogućnost njihovog korištenja.
8. **Oporavak od oštećenja** – svaki datotečni sustav bi trebao imati mehanizme za oporavak od grešaka i oštećenja podataka, što uključuje:
    a. **Konzistentnost podataka** – nakon pada sustava, datotečni sustav mora moći vratiti integritet podataka, što je često omogućeno korištenjem *journaling-a*.
    b. **Alate za provjeru i popravak** – datotečni sustavi trebaju imati alate poput **fsck** (*file system check*) koji mogu provjeriti i popraviti oštećenja podataka na disku.
9. **Podrška za različite kategorije uređaja** - datotečni sustavi na Linuxu moraju biti fleksibilni i podržavati različite vrste uređaja poput: tvrdih/mehaničkih diskova (*HDD*), *SSD* i NVMe diskova, USB uređaja, RAID sustava i slično.
10. **Performanse i optimizacija –** osnovni datotečni sustavi na Linuxu moraju biti optimizirani za rad s različitim vrstama opterećenja uz:
    a. **Nisku fragmentaciju** – datotečni sustav bi trebao minimizirati fragmentaciju datoteka kako bi poboljšao performanse.
    b. **Efikasnu upotrebu prostora** – sustav mora pametno raspoređivati prostor na disku, osobito kod rada s malim datotekama.
11. **Podrška za snapshote, kompresiju i deduplikaciju –** napredni datotečni sustavi često podržavaju stvaranje *snapshota*, koji omogućuju spremanje trenutnog stanja datotečnog sustava. Neki moderni Linux datotečni sustavi, poput ***Btrfs*** i ***ZFS***, nude i kompresiju podataka, što smanjuje zauzeće prostora na disku. Također, ako postoji, **deduplicacija** omogućava uklanjanje dupliciranih podataka radi bolje iskoristivosti diskovnog prostora.

## Logička podjela datotečnih sustava

Datotečne sustave u grubo možemo podijeliti u nekoliko logičkih kategorija:

- **Diskovni datotečni sustavi** – ova kategorija obuhvaća najčešće korištene datotečne sustave koji se nalaze na fizičkim diskovima poput hard diskova (HDD), *solid-state* diskova (SSD) i drugih memorijskih uređaja. Predstavnici ovakvih datotečnih sustava su: ***EXT2/3/4***, ***XFS***, ***NTFS***, ***Btrfs***, ***ZFS*** i slični.
- **Mrežni datotečni sustavi** – ovi datotečni sustavi omogućuju dijeljenje datoteka preko mreže, omogućavajući pristup istim podacima na udaljenim lokacijama za više korisnika ili sustava. Neki od predstavnika ove skupine su: **NFS** (*Network File System*), **SMB/CIFS** (*Server Message Block*), **GlusterFS** i **Ceph.**
- **Datotečni sustavi za specijalizirane medije** – ova kategorija obuhvaća datotečne sustave dizajnirane za specifične vrste medija, kao što su SSD-ovi, *flash* memorije ili optički diskovi. Primjeri su: ***F2FS*** (*Flash-Friendly File System*): Optimiziran za *flash* memoriju te ***exFAT*** koji je razvijen za korištenje na *flash* pogonima, te omogućuje interoperabilnost između različitih operativnih sustava. Sljedeći primjer je ***ISO 9660*** koji je standard za optičke diskove odnosno *CD*, *DVD* i *Blu-Ray* medije.
- **Datotečni sustavi za pohranu u memoriji (In-Memory File Systems)** – ovi datotečni sustavi koriste RAM memoriju umjesto diskova za pohranu podataka. Obično se koriste za privremenu pohranu zbog visoke brzine, ali i činjenice da se podaci gube pri gašenju sustava. Jedan od primjera jest ***tmpfs*** koji pohranjuje privremene datoteke u RAM memoriju. Dok je drugi primjer ove kategorije ***ramfs*** koji radi slično ***tmpfs-u***, ali bez mogućnosti ograničenja veličine.
- **Virtualni datotečni sustavi** – oni omogućuju prikaz podataka koji nisu fizički pohranjeni na disku, već su generirani u stvarnog vremenu ili predstavljaju informacije o samom sustavu. Uobičajeno se u Linuxu nalaze unutar tzv. **VFS** sloja datotečnog sustava koji je pohranjen u RAM memoriji. Jedan od njih je ***procfs*** a on omogućuje pristup informacijama o pokrenutim procesima i nekim komponentama sustava, a drugi je ***sysfs***. **Sysfs** omogućuje pristup kernel modulima i informacijama o hardveru, kao i postavkama sustava na najnižoj razini.
- **Copy-on-Write datotečni sustavi** (**COW**) – ovi sustavi koriste *Copy-on-Write* mehanizam, gdje se prilikom modifikacije podataka stvara kopija postojećih podataka umjesto da se direktno mijenjaju, čime se povećava sigurnost te omogućuju funkcije poput *snapshota*. Jedan od ovih predstavnika jest ***Btrfs***, a on ima ugrađenu podršku za COW s naprednim značajkama. Drugi primjer je ***ZFS*** koji je poznat po ***COW*** funkcionalnosti, zaštiti podataka i mnogim dodatnim i naprednim značajkama.
- **Distribuirani datotečni sustavi** – sustavi su to koji omogućuju pohranu podataka na više čvorova odnosno poslužitelja, često u cloud ili okruženju podatkovnih centara (engl. *Data center*). Podaci se distribuiraju na više fizičkih lokacija kako bi se postigla redundancija i visoka dostupnost. Neki od predstavnika ove kategorije su:
    - **Ceph** – visoko distribuirani sustav (na stotine poslužitelja i desetke tisuća diskova) za pohranu podataka s podrškom za visok stupanj skalabilnosti, redundancije i dostupnosti**.**
    - **GlusterFS** - Omogućuje distribuiranu pohranu podataka na više poslužitelja.

  - **HDFS** (*Hadoop Distributed File System*) - dizajniran za masovno pohranjivanje podataka u velikim klasterima.
- **Kriptirani datotečni sustavi** – ovi sustavi omogućuju kriptiranje (šifriranje) podataka kako bi se povećala sigurnost pohranjenih datoteka. Enkripcija osigurava da samo autorizirani korisnici mogu pristupiti takvim podacima.

Do sada smo naučili da je u *UNIX-u i Linuxu* i svaki uređaj u konačnici neka (posebna) datoteka pa sâmim time diskovni podsustav operativnog sustava Linux dobiva dodatno na važnosti. Za razliku od operativnih sustava *Windows*, Linux ne koristi ***FAT, FAT32, extFAT*** ili ***NTFS*** datotečni sustav za pohranu podataka odnosno datoteka i direktorija na disk. Naime, moderne distribucije Linuxa koriste: ***ext3***, ***ext4***, ***XFS, ZFS*** ili neke druge napredne datotečne sustave.

## Sumirajmo prednosti modernih Linuxovih datotečnih sustava

Osnovne prednosti navedenih modernih Linuxovih datotečnih sustava su sljedeće:

- Ne postoji fragmentacija podataka, koja se inače događa nakon brisanja određenih dijelova datoteke ili cijelih datoteka, ali i kreiranja novih datoteka. Osim toga podržavaju i dodatne napredne značajke (mogućnosti).
- Oni su u pravilu *transakcijski datotečni sustavi* (koriste tzv. *journaling*), te su stoga sigurniji na nekontrolirane događaje, poput rušenja sustava, naglog nestanka el. energije i slično. To se postiže praćenjem svih promjena na datotekama (ili direktorijima) koje još nisu zapisane na površinu diska; jer se u datom trenutku mogu nalaziti u (RAM) međumemoriji. Ovakav sustav, tek kada se podaci stvarno i pohrane (snime) na površinu diska, označava ih kao dovršenu transakciju odnosno kao sigurno pohranjene. Stoga se sustav u slučaju neuspjele transakcije vraća na onu prethodnu koja je završila uspješno, odnosno u kojoj su podaci uredno zapisani na površinu diska.
- Druga prednost jest ***COW*** način zapisivanja većine datotečnih sustava, kod koje se prilikom modifikacije podataka stvara kopija postojećih podataka umjesto da se direktno mijenjaju, čime se povećava sigurnost te omogućuju funkcije poput *snapshota*.

## Od čega se sastoji klasični UNIX/Linux datotečni sustav?

Svi *Unix/Linux* datotečni sustavi se sastoje od sljedećih važnih struktura:

- Direktorija (*mapa*) i datoteka.
- Takozvanih ***I-node*** tablica.

***I-node*** tablica sadrži dodatne podatke o svakoj datoteci i direktoriju (*mapi*) na sustavu, odnosno daje nam podatak:

- Tko je njen vlasnik (engl. *owner*).
- Kojoj korisničkoj grupi (engl. *group*) ona pripada.
- Koja prava imaju vlasnik, korisnička grupa ili svi ostali: čitanja(***r***) i/ili pisanja(***w***) i/ili izvršavanja(***x***) odnosno:
  - ***r*** *(engl. read)*, ***w*** *(engl. write)*, ***x*** *(engl. execute)*.
- O drugim naprednim podacima i *metapodacima* (pr. napredne ovlasti [***ACL***], ***atribute*** i mogućnosti [***capabilities***]).



Osim naziva ***direktorij***, u praksi se možete susresti i s pojmovima: ***folder*** ili ***mapa***. Pri tome direktorij označava strukturu unutar koje se spremaju datoteke odnosno podaci, kao što se u našem svijetu u fascikle ili registratore spremaju neki dokumenti. Međutim, u Linux i UNIX svijetu ***mapa*** ili ***folder*** se uvijek nazivaju **direktorij**. Datoteke još zovemo i *file-ovi*, a one na razini operativnog sustava, odnosno datotečnog sustava, na najnižoj razini sadrže binarne podatke.

**Pogled na datoteke i direktorije**

S razine aplikacija podaci se prikazuju kao nama (ili aplikaciji) upotrebljiviji podaci, poput: teksta, slika, zvuka, videa i slično. Ti podaci se pohranjuju u datoteke, kao nama vidljivi i razumljivi podaci (pr. tekst) iako se oni zapravo na najnižoj razini sustava pohranjuju u malo drugačijem formatu odnosno pohranjuju se kodirani na poseban način. To nas za sada ne treba previše zamarati, a ako vas ipak zanimaju detalji o pohrani podataka unutar datoteka na najnižoj razini sustava, pogledajte dolje predložena poglavlja. Ipak, za početak, važno je znati da se podaci pohranjuju odnosno spremaju u datoteke.



Vezano za način pohrane podataka u datoteke (na najnižoj razini sustava) pogledajte poglavlja:
**4.7. Format datoteka (*MIME type*).**
**10.3.2.1. Kodiranje i dekodiranje.**

Kako bi se sâme datoteke mogle raspoređivati prema nekoj logici ili hijerarhiji, koriste se direktoriji odnosno mape, koje obično i sâmi kreiramo i strukturiramo tako da bi nam pronalaženje datoteka unutar njih bilo što logičnije i lakše.
Prema osnovnoj logici bi tada mogli primjerice imati zasebni direktorij u koji bi spremali datoteke za glazbu, a posebni direktorij za datoteke s dokumentima, kao što su PDF dokumenti i knjige, pa zatim poseban direktorij za filmove (video), i tako dalje.

**O detaljima vezanim za hijerarhijsku strukturu direktorija pogledajte cjeline koje slijede!**

**Izvori informacija: (103),(1003),(1030),(1031),** `man 5 filesystems`, `man 7 file-hierarchy`.

# 4.1. Struktura datoteka i direktorija (*mapa*)

U sâmoj strukturi i nazivima datoteka i direktorija (*mapa*), postoje određena pravila:

- Razlikuju se velika i mala slova.
- Trebala bi se koristiti hijerarhijska struktura direktorija.
- Struktura direktorija je u obliku stabla s korijenom i njegovim grânama (tzv. *hijerarhijska struktura*).
- Osnovne grâne direktorija i poddirektorija su već definirane i rezervirane za operativni sustav.
- Imena direktorija i datoteka mogu biti dužine do 255 znakova, a mogu se koristiti svi znakovi osim:
  ! @ $ ^ & * , ; ' ` "( ){ } [ ] < > | / \ ? ~

## Struktura sistemskih direktorija (mapa)

Osnovne grâne direktorija i poddirektorija, odnosno struktura sistemskih direktorija je definirana unutar standarda koji definira *hijerarhiju* datotečnog sustava odnosno takozvanog ***FHS*** (engl. *Filesystem Hierarchy Standard*). Osnovna struktura direktorija u svim Unix ili Linux operativnim sustavima je *hijerarhijska* te izgleda kao stablo, kojemu je početni odnosno korijenski (engl. *root*) direktorij označen sa znakom `/`. Iz tog korijena stabla se razvijaju direktoriji koji su kategorizirani prema namjeni.

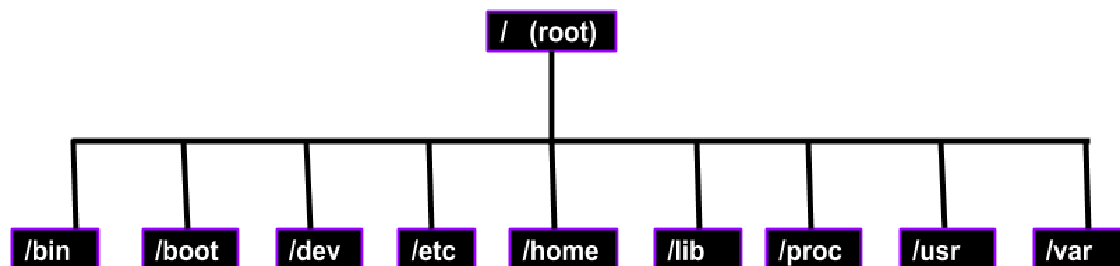
*Hijerarhija* direktorija i datoteka na Linuxu drži se ***POSIX*** standarda.



Vezano za **POSIX** standard pogledajte poglavlje: **10.4.4.1. POSIX standard.**

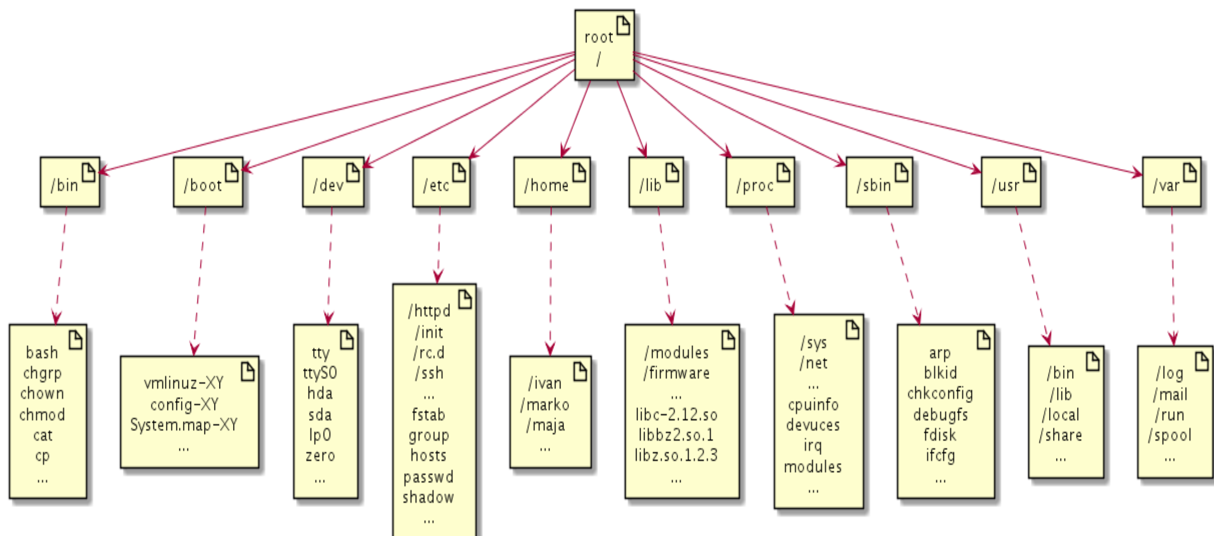
Prikaz osnovne hijerarhijske strukture sistemskih direktorija, vidljiv je na slici 7.

**Slika 7. Hijerarhijsko stablo vršnih direktorija (mapa) Unix/Linux sustava.**




Pogledajmo i malo detaljniju sliku hijerarhijskog stabla *Unix*/Linux direktorija (*mapa*), uz opis što se obično nalazi u njima:

**Slika 8. Stablo direktorija (mapa) Unix/Linux sustava.**




Što se nalazi u navedenim osnovnim direktorijima (*mapama*) operativnog sustava Linux, a vidljivo na slici 8. je sljedeće:

- `/` – ovo je korijenski (engl. *root*) direktorij cijelog sustava. Svi ostali direktoriji i datoteke su smješteni ispod njega.
- `/bin` – ovdje se nalaze osnovne sistemske naredbe (uglavnom *binarne* izvršne datoteke), koje moraju biti dostupne i u jednokorisničkom, posebnom načinu rada *(single user* načinu rada). Ovdje su locirane Linux naredbe koje su dostupne i administratoru (*root* korisniku), ali i svim ostalim korisnicima na sustavu. Ovdje se primjerice nalaze naredbe poput: `ls`, `cp`, `mv`, `bash`, ...

- /boot – ovdje se nalazi sve potrebno za pokretanje (*boot*) Linux sustava, a to su:
  - ***Kernel*** – koji je zapravo datoteka imena poput primjerice: **vmlinuz-2.6.32-**XY.
  - ***Inicijalni RAM disk*** koji sadrži „*root*" datotečni sustav koji se učitava u RAM memoriju, prilikom podizanja sustava. Ovo je datoteka imena, poput **initrd.img-2.6.32-**XY ili **initramfs-XY**.
  - **Konfiguracijska** datoteka koja dolazi uz pripadajući *kernel* i *initrd*, koja opisuje s kojim vanjskim kernel modulima i onim ugrađenima u kernel je *kompiliran* pripadajući kernel, ime joj je pr.:**config-2.6.32-**XY.
  - ***Datoteka*** u kojoj je definirana takozvana tablica simbola (Engl. *symbol table)* koja povezuje simbole kernela i njihove pripadajuće adrese. Njeno ime je primjerice: **System.map--2.6.32-39-XY**.



Pogledajte poglavlje:
**11.1.1.4. Kompiliranje novog kernela.**

- /dev – ovdje se nalaze datoteke koje omogućuju komunikaciju s hardverom. Odnosno ovdje se nalaze posebne datoteke koje predstavljaju posebne uređaje poput generatora nula /dev/zero te datoteke koje predstavljaju sâv hardver (engl. *devices*). Posebne datoteke koje predstavljaju hardver su datoteke poput sljedećih posebnih datoteka:
  - /dev/hda – predstavlja prvi **ATA** d isk (i to primarni **MASTER** disk).
  - /dev/sda – predstavlja prvi **SATA** ili **SCSI** disk.
  - /dev/ttyS0 – predstavlja prvo serijsko sučelje (port) računala.
  - /dev/lp0 – predstavlja prvo paralelno sučelje (port) na računalu.



Pogledajte poglavlje:
**11.1.1.2. Napredno: Rad s kernel modulima.**

- /etc – ovdje se nalaze sve konfiguracijske datoteke. Obično su raspoređene unutar pôddirektorija (pôd mapa), od kojih svaki od njih pripada određenom servisu (tzv. *daemonu*). U vršnom stablu ovog direktorija, nalaze se i važne sistemske konfiguracijske datoteke poput:
  - fstab – definicije *auto mount* particija odnosno datotečnih sustava, koji se automatski povezuju (*montiraju*) prilikom pokretanja sustava.
  - passwd – ovdje se nalazi lista i definicija korisničkih računa.
  - shadow – ovdje se pohranjuju kriptirane (zapravo *hash* vrijednosti) lozinki za sve korisničke račune.
  - group – ovdje se nalazi definicija/lista korisničkih grupa i pripadajućih korisnika.
  - hosts – tablice s imenima računala i njihovim pripadajućim IP adresama; po principu: ***IP - ime računala***.

- /home – ovdje se nalaze korisnički osobni direktoriji korisnika (engl. *home directory*). Svaki korisnik ima svoj korisnički (početni/kućni) pôddirektorij unutar ovog vršnog direktorija.

- /lib – ovdje se nalaze sve sistemske biblioteke te biblioteke za razne aplikacije (engl. *library*). Odnosno ovdje se nalaze dinamičke biblioteke (slično DLL datotekama u Windowsima) potrebne za pokretanje osnovnih programa smještenih u /bin i /sbin direktorijima. Ove programske biblioteke su primjerice datoteke poput: libc.so, ld-linux.so i drugih.
  Osim toga ovdje se nalaze i pôddirektoriji u kojima se nalaze upravljački programi (*kernel moduli*) za sâv hardver, te *firmware* za sâv hardver za koji je to potrebno i slično.
  Vezano za kernel module to je uobičajeno poddirektoriji poput: /lib/modules/***inačica-kernela***/... .
  Primjerice: /lib/modules/5.15.0-72-generic/.



Pogledajte poglavlje:
**11.2. Sistemske biblioteke (*Libraries*).**

- /mnt ili /media – ovi direktoriji se koriste kao točke za montiranje privremenih datotečnih sustava, poput USB uređaja ili vanjskih diskova. Ovdje stoga možemo vidjeti točke montiranja poput: /mnt/disk, /media/usb i sl.

- `/proc` – ovo je virtualni datotečni sustav, koji se kreira u RAM memoriji, tijekom svakog pokretanja sustava. To znači da datoteke u njemu nisu stvarne datoteke već virtualne, a koriste se za pristup informacijama o sustavu. One nam primjerice daju informacije o svim procesima, poruke kernela te sistemske atribute, koje je moguće i mijenjati.
Neke od datoteka koje možemo vidjeti ovdje su: `/proc/cpuinfo`, `/proc/meminfo`, ...
Primjerice naredbe `ps` `top`, `free` su namijenjene za čitanje iz datoteka unutar ovog direktorija, a primjerice `sysctl` i druge naredbe koriste se za čitanje i zapisivanje u specifične datoteke unutar ovog direktorija (mape).
Kreira ga "***proc(fs)***" datotečni sustav u trenutku pokretanja sustava.



Pogledajte poglavlje:
**13.9.1. Direktorij /proc.**

- `/run` – ovdje se nalaze privremeni podaci specifični za trenutno pokretanje sustava, poput PID datoteka ili *socketa*. Primjerice ovdje možemo vidjeti datoteke poput: `/run/sshd.pid`.

- `/sys` – ovo je također virtualni datotečni sustav, koji se kreira u RAM memoriji, tijekom svakog pokretanja sustava, a koji strukturirano i uniformirano prikazuje informacije o sustavu te nudi kontrolu nad njim. Upravljački programi (engl. *Driver*) pomoću kernela kreiraju svoju strukturu direktorija i datoteka; slično kao kod `/proc/`, ali na strukturiran način. Ovdje se također nalaze i informacije i konfiguracije (koje je moguće mijenjati.)
Kreira ga "***sysfs***" datotečni sustav, u trenutku pokretanja sustava. Svi noviji procesi koriste (i) ovaj sustav, uz već spomenuti: `/proc/`.



Pogledajte poglavlje:
**13.9.2. Direktorij /sys.**

- `/sbin` – ovdje se nalaze sistemske naredbe, koje su dostupne sâmo administratorima: *root* korisniku i svim ostalim korisnicima s administratorskim (*root*) ovlastima. Dakle on sadrži binarne datoteke koje su potrebne za administraciju sustava. Neke od naredbi koje možemo ovdje pronaći su: `ip`, `ifconfig`, `fsck`, `mount`, `reboot`. Ove naredbe koriste se za ključne administrativne zadatke, posebno tijekom pokretanja sustava ili u situacijama kada sustav radi u ograničenom načinu rada, poput "*single-user mode*".

- `/usr` – ovdje se nalaze standardni višekorisnički (*multiuser*) programi, podaci i drugo; za sve korisnike. On sadrži korisničke programe i biblioteke koji nisu nužni za osnovno funkcioniranje sustava. Često se koristi za instalaciju novog softvera. Neki od poddirektorija koje često možemo ovdje vidjeti su:
    - `/usr/bin/` – ovdje se pohranjuju korisnički binarni programi.
    - `/usr/sbin/` – ovdje se pohranjuju administratorski binarni programi. Dakle ovaj direktorij sadrži administrativne alate koji nisu nužni za osnovno održavanje sustava, već su namijenjeni za napredne funkcije administracije koje nisu potrebne u ranim fazama pokretanja sustava. Sve naredbe u ovom direktoriju postaju dostupne tek nakon što se montira `/usr` direktorij.
    - `/usr/lib/` – ovdje se pohranjuju korisničke biblioteke (za korisničke programe).
    - `/usr/local/` – ovdje se pohranjuju korisnički programi (oni koje instaliraju korisnici).
    - . . .

- `/var` – ovdje se nalaze promjenjivi (varijabilni) podaci, poput *log* datoteka, privremenih *cache* datoteka i slično. Datoteke se ovdje obično nalaze u pôddirektorijima, pri tome je svaki od njih za svoju namjenu. Tako se recimo u poddirektoriju `/var/log/httpd` nalaze *log* datoteke za web poslužitelj, jedna datoteka za greške (*error_log*), jedna u koju se upisuje svaki pristup na web poslužitelj (*access_log*) i tako dalje.

- `/opt` – sadrži softver ili dodatne programe koji nisu dio osnovne distribucije Linux. Primjerice: `/opt/google/chrome`.

- `/tmp` – ovdje se privremeno spremaju (privremene) datoteke to jest privremeni podaci za sve korisnike. Sadržaj ovog direktorija sustav periodički čisti.



Pogledajte poglavlje:
**13.9.2.2. Direktorij /tmp i privremene datoteke.**

**Izvori informacija: (103),(1030),(1031),(K-1),(K-12), `man 7 file-hierarchy`.**

# 4.2. Direktoriji (mape) i datoteke

Vrste direktorija (*mapa*) i datoteka su vidljive prilikom izlistanja datoteka ili direktorija, ako gledamo potpuno lijevu oznaku njihovog ispisa. Za izlistavanje datoteka i direktorija, koristit ćemo naredbu `ls`.
U ovom primjeru, pogledat ćemo samo nekoliko specifičnih datoteka:

- `/dev/sda` – ovo je posebna datoteka koja predstavlja prvi **SATA** ili **SCSI** tvrdi disk.
- `/dev/tty` – ovo je također posebna datoteka koja predstavlja radni terminal.
- `/etc/rc.d/rc.local` – ovo je datoteka koja je zapravo *shell* skripta.
- `/etc/rc.d/rc3.d/S90crond` – ovo je datoteka koja je također *shell* skripta zadužena za rad s ***cron*** servisom.

Pomoću naredbe `ls` izlistajmo datoteke i direktorije te pogledajmo što nam govore njihove oznake, koje su vidljive, gledamo li ispis i to krajnji lijevi dio odnosno njegov krajnji lijevi stupac :

```
ls -al /dev/sda /dev/tty /etc/rc.d/rc.local /etc/rc.d/rc3.d/S90crond

brw-rw----  1 root disk 8, 0  Apr 8 15:57 /dev/sda
crw-rw-rw-  1 root tty  5, 0  Apr 7 18:15 /dev/tty
lrwxrwxrwx. 1 root root 14    Dec 2 11:34 /etc/rc.d/rc3.d/S90crond -> ../init.d/crond
-rwxr-xr-x. 1 root root 882   Dec 2 11:34 /etc/rc.d/rc.local
```

Primjerice za datoteku: `/etc/rc.d/rc.local` gledamo lijevi dio donjeg ispisa, koji predstavlja vrstu datoteke i njene ovlasti, a koji ćemo dalje promatrati. Dodatno smo ga i zatamnili kako bi bilo još jasnije što ovdje proučavamo:

```
-rwxr-xr-x. 1 root root  882  Dec 2 11:34 /etc/rc.d/rc.local
```

***Ovlasti***

Pogledajmo koje sve oznake postoje za datoteke i direktorije, ako gledamo prvu oznaku s lijeve strane:

- [d] – odnosno ovo prvo slovo predstavlja direktorij.
- [-] – odnosno ovaj prvi znak (ako je - prvi znak) predstavlja običnu datoteku.

Osim standardnih, postoje i posebne vrste datoteka, koje predstavljaju logičke jedinice, koje nam daju određene značajke (funkcionalnosti), poput:

- `/dev/zero` – koja predstavlja generator nula [*0*].
- `/dev/random` – koja predstavlja generator slučajnih brojeva [engl. *Random number generator*].

Osim navedenih logičkih jedinica odnosno vrsta datoteka, postoje i dodatne vrste posebnih datoteka kojima pripadaju:

- [l] – predstavlja ***simbolički link*** odnosno simboličku poveznicu s drugom datotekom (ili direktorijem).

- [p] – predstavlja ***named pipe*** datoteku ili sâmo *pipe* (*FIFOs*). *Named pipe* radi slično poput "*character device*" uređaja, ali se ne radi o direktnom pristupu uređaju preko njegovog upravljačkog programa. Ovdje se radi o tôku podataka prema drugom procesu (programu). Za više detalja o njoj pogledajte poglavlje: **9.4.2 Named pipe.**

- [s] – predstavlja **socket file**. On predstavlja posebnu vrstu ulazno/izlaznog (I/O) tôka podataka koji se koristi za mrežnu komunikaciju, slično kao i kod komunikacije između procesa (engl. *Inter process communication*) odnosno **IPC**. U slučaju njihove upotrebe kod mrežne komunikacije otvara se mrežni komunikacijski kanal, na koji se može direktno spajati odnosno primati ili slati podatke. Ista stvar je i kod komunikacije između procesa (programa).
Za više detalja, pogledajte napredno poglavlje: **9.4.3. Slijedeći koraci: *Unix* i *network socketi*** .

Postoji i druga grupa posebnih datoteka koja predstavlja određenu vrstu uređaja, a ovo su takozvani *device fileovi*:

- [c] – predstavlja uređaj koji barata s ulazno/izlaznim uređajima (I/O) i to s karakterima odnosno znakovima (***character device)***. Omogućavaju čitanje ili pisanje nîza znakova. Na vrlo niskoj razini, kada čitamo ili zapisujemo na neki *character device*, kernel prosljeđuje nîz znakova ("*karaktera*") direktno na odredišni uređaj, bez ikakve kontrole ili zapisivanja bilo kakvih aktivnosti ili stanja tijekom operacije čitanja ili zapisivanja. Pristupa se zapravo direktno na upravljački program uređaja te sâmim time (in)direktno na uređaj. Uređaji koji se koriste kao *character* su obično: *miš*, *tipkovnica*, *pisač* i slični uređaji prema kojima se obično ne očekuje veliki protok podataka.

- [b] – predstavlja blok (***block)*** uređaj koji barata s ulazno/izlaznim (I/O) blokovima podataka. Ovim uređajima se pristupa, odnosno na njih se zapisuje ili se s njih čita u većim blokovima podataka. Za razliku od karakter (*character)* uređaja na vrlo niskoj razini, blokovi podataka koji se čitaju ili zapisuju, mogu prolaziti kroz međuspremnike (engl. *buffers*) te se u pravilu bilježe razni parametri pristupa ovim uređajima, kao i transakcijama s i na njih. Uređaji koji se koriste kao blok (*block*) uređaji, uglavnom su diskovni sustavi ili sustavi za pohranu podataka. Tipično su to uređaji poput: *tvrdih diskova*, ***SSD*** *diskova*, uređaja za izradu sigurnosnih kopija (engl. *Backup devices*) i slično, od kojih i prema kojima se očekuje prijenos velike količine podataka pa se sâmim time prema njima i od njih podaci šalju u većim blokovima, kako bi se što više podataka prenijelo u što kraćem vremenu.

Iz nekoliko primjera izlistanja direktorija i datoteka će sve biti puno jasnije.
Izlistajmo sve datoteke i direktorije, počevši od vršnog direktorija (/) pomoću naredbe ls:

```
ls -al /

drwxr-xr-x 20 root root 4096  Jun 13 12:53   ./
drwxr-xr-x 20 root root 4096  Jun 13 12:53   ../
drwxr-xr-x 2  root root 4096  Mar 25 2011    bin/
drwxr-xr-x 2  root root 4096  Oct 6 1997     boot/
drwxr-xr-x 17 root root 69632 Nov 11 15:20   dev/
drwxr-xr-x 19 root root 4096  Nov 11 15:20   etc/
drwxr-xr-x 2  root root 4096  Oct 6 1997     home/
drwxr-xr-x 4  root root 4096  May 4 2011     lib/
dr-xr-xr-x 17 root root 0     Nov 11 15:20   proc/
drwxr-xr-x 15 root root 4096  Jun 13 12:53   usr/
drwxr-xr-x 11 root root 4096  Jun 13 12:53   var/
```

Ovdje su vidljivi samo obični direktoriji [lijeva strana ima oznaku d]. Stoga pogledajmo i jednu datoteku: /dev/sda1

```
ls -al /dev/sda1

brw-rw----. 1 root disk 8, 1 Nov 11 17:36 /dev/sda1
```

Vidljivo je kako je ona posebna datoteka (tzv. *device file*) koja predstavlja uređaj, konkretno prvi **SATA** ili **SCSI** tvrdi disk i to njegovu prvu particiju (o tome kasnije). Ovo je posebna datoteka stoga jer joj je prva oznaka [b], a koja označava blok uređaj (o njima znatno kasnije). Sada pogledajmo još jednu posebnu datoteku: /dev/ttyS0

```
ls -al /dev/ttyS0

crw-rw----. 1 root dialout 4, 64 Nov 11 17:36 /dev/ttyS0
```

Ova datoteka (/dev/ttyS0), predstavlja prvo serijsko sučelje na računalu (RS232). Vidimo kako je njena oznaka [c], što nam govori kako je ona vrste karakter uređaja (*character)*, a o njima ćemo također govoriti tek znatno kasnije.

A sada pogledajmo još jednu datoteku, koja predstavlja CD/DVD-ROM (/dev/cdrom) uređaj (*device file*), ali indirektno. Dakle lijeva oznaka ovlasti joj je [l] što znači kako je ovo simbolički link odnosno poveznica na drugu datoteku. U ovom slučaju to znači kako je datoteka imena /dev/cdrom zapravo poveznica na drugu datoteku imena /dev/sr1, koja stvarno i predstavlja CD/DVD-ROM uređaj odnosno ona upućuje na njegov upravljački program (*engl. driver*).

```
ls -al /dev/cdrom

lrwxrwxrwx. 1 root root 3 Nov 11 17:36 /dev/cdrom -> sr1
```

## Nazivi: folder, direktorij i mapa

Ako koristite računala s operativnim sustavom ***Windows***, vjerojatno ćete koristiti izraz *mapa* (engl. *folder*). Međutim, kada prijeđete na Linux ili neku od varijanti UNIX-a, vidjet ćete da se mape često nazivaju *direktorijima*. Mapa (prema logici rada kao *omotnica* ili *fascikl*) može se koristiti za držanje nekoliko datoteka (ili drugih stavki) u njoj. Dok se ono što se naziva direktorij (ili imenik) može koristiti za održavanje indeksa odnosno popisa stavki, tako da možete pronaći koje se stavke gdje nalaze.
Pojam direktorij je postojao i prije postojanja Linuxa. Naime, on dolazi iz ere UNIX-a od početka 1970. godine odnosno prvog **UFS** datotečnog sustava. S obzirom da je Linux naslijedio puno stvari od UNIX-a, ovo je samo jedna od njih.

Nadalje, kako ćemo kasnije naučiti, *direktorij* zapravo NE drži datoteke unutar sebe. Direktorij je posebna datoteka u datotečnom sustavu koja sadrži informaciju o tome gdje se (sadržaj) datoteke nalazi odnosno gdje je stvarno pohranjen i to pomoću takozvanih ***inode*** unosa. Naziv ***inode*** je skraćenica od *index node*, a on praktično opisuje objekte datotečnog sustava kao što su direktoriji i datoteke.



Za detalje o ***inode*** unosima, pogledajte poglavlje:
**4.5. Datotečni sustav detaljnije.**

Sada ima smisla zašto se zove *direktorij* zove *direktorij* odnosno možemo ga nazvati i imenik. Direktorij čuva indeks stavki, ne nužno i same stavke. Dakle *direktoriji* u Linuxu i UNIX-u ne pohranjuju datoteke u sebi.
Oni samo imaju informacije o lokaciji datoteka pomoću *inode* unosa koji pokazuju na te datoteke odnosno njihov sadržaj, veličinu, ovlasti, vrijeme kreiranje i drugo.

**Izvori informacija:(107),(108),(1031),(1481),(K-12)**,`man ls`,`man 7 file-hierarchy`.

## 4.2.1. Putanje do datoteka i *PATH* varijabla

Putanja do neke datoteke unutar *direktorija* (*mape*) u kojoj se nalazi, može biti:

- Apsolutna, odnosno ona koja počinje od vršnog direktorija (`/`) poput primjerice datoteke `messages`, kojoj je tada apsolutna putanja: `/var/log/messages`.
- Relativna, odnosno ona koja počinje od trenutnog direktorija u kojem se nalazimo. Primjerice, ako se nalazimo u direktoriju `/var` tada nam je putanja do navedene datoteke: `log/messages`.

Važno je razumjeti kako na razini cijelog sustava postoji posebna *shell varijabla* imena `PATH` unutar koje su definirane putanje do svih ***važnijih*** direktorija u kojima se nalaze izvršne datoteke odnosno naredbe, ali i skripte koje su važne za funkcioniranje cijelog sustava, kao i datoteke koje su potrebne za pokretanje raznih programa.

Naime u trenutku kada želimo pokrenuti neki program ili naredbu odnosno *shell* ili neku drugu skriptu, sustav prvo pregledava je li ona dostupna u putanji direktorija koja je navedena u varijabli `PATH`.
Dakle provjerava se, nalazi li se ta izvršna datoteka odnosno program, naredba, *shell* ili neka druga skripta koju želimo pokrenuti, u nama dohvatljivoj putanji direktorija. Ako se **ne** nalazi, sustav **ne** može doći do nje i sâmim time ju ne može niti pokrenuti. Ako se ipak nalazi, to znači da se izvršna datoteka (program/aplikacija/skripta) može i pokrenuti.

Standardna putanja (*PATH*) za *root* korisnika (administratora) za izvršne datoteke je sljedeća:

```
/usr/local/sbin
/usr/local/bin
/sbin
/bin
/usr/sbin
/usr/bin
```

Varijabla `PATH` se može definirati u konfiguracijskim datotekama poput `/etc/profile` i drugima specifičnim za vašu ljusku, ali i unutar svake skriptne datoteke; primjerice unutar *shell* skripte. Nove putanje direktorija se u ovoj varijabli odvajaju s dvotočkom (`:`).

Pogledajmo primjer definiranja ove varijable za korisnika `root` odnosno pogledajmo kako ona izgleda:

```
PATH=/usr/bin:/bin:/sbin:/usr/sbin:/sbin:/usr/local/sbin:/usr/local/bin
```

Vrijednost ove varijable za trenutnog korisnika možemo vidjeti sa naredbama `set`, `env` i `echo $PATH` .

Dakle ispis ove varijable i njene vrijednosti možemo dobiti s pozivanjem naredbi:

```
set
```

ili

```
env
```

ili

```
echo $PATH
```



Za postavljanje ove sistemske varijable pogledajte poglavlje:
**6.2.2. Sistemske (*Environment*) varijable i postavke terminala.**

**Sažetak upotrebe PATH varijable:**

- PATH varijabla sadrži popis direktorija, razdvojenih dvotočkama (:).
- Kada korisnik pokuša izvršiti naredbu, naredbeni redak (naredbena ljuska/*shell*) pretražuje direktorije u PATH varijabli, redom, tražeći izvršnu datoteku s imenom naredbe.
- Ako naredbeni redak pronađe izvršnu datoteku u jednom od direktorija navedenih u PATH varijebli, ona se pokrene.
- Ako naredbeni redak ne pronađe izvršnu datoteku ni u jednom od direktorija, korisnik će dobiti poruku o pogrešci.

**Izvori informacija: (106),(K-1),(K-12).**

# 4.3. Ovlasti (*permissions* & *modes*)

Svaki *direktorij* kao i svaka datoteka, imaju određene ovlasti (engl. *permissions*). Sjetimo se da smo nakon prijavljivanja na sustav (*logiranja*), prijavljeni kao određeni korisnik koji pripada određenoj korisničkoj grupi.
Taj ili bilo koji drugi korisnik i njegova pripadajuća grupa se uz druge ovlasti zapisuju uz svaku datoteku i direktorij zbog ograničavanja prava pristupa na njih.
Naime svaki puta kada se kreira neka datoteka ili direktorij, za nju se zapisuju i njene ovlasti, a to su:

- Tko je njen vlasnik (*owner*).
- Kojoj korisničkoj grupi pripada (*group*) *[ovdje se zapisuje **primarna** korisnička grupa korisnika]*.
- Te ovlasti za sve ostale korisnike koji nisu vlasnici ili ne pripadaju definiranoj grupi (Engl. *All other users*).

Ove ovlasti se mijenjaju s naredbom: `chown` (o njoj kasnije). Stoga je prema vlasništvu datoteke ili direktorija, moguće definirati:

- Tko im je vlasnik (*owner*).
- Kojoj korisničkoj grupi pripadaju (*group*).
- Za sve ostale, koji nisu u kategoriji vlasnika ili primarne grupe (*all other users*).

Osim toga, definiraju se i prava pristupa i rada s datotekama i direktorijima (mapama), koja se vežu za: vlasnika, korisničku grupu i sve ostale. Ovlasti se mijenjaju s naredbom: `chmod`.
Prema pravima pristupa nad datotekama i direktorijima (mapama), moguće je definirati tko ima pravo:

- Čitanja (engl. ***read***) [r].
- Pisanja (engl. ***write***) [w].
- Izvršavanja odnosno pokretanja (engl. *execute*) [x].

**Za datoteke vrijede sljedeća pravila**
Za datoteke, pravo čitanja [r] omogućava sâmo pravo čitanja određene datoteke, ali ne i njeno mijenjanje. Pravo čitanja datoteke je potrebno i za njeno kopiranje, jer se prilikom kopiranja, kopira njen sadržaj. Dok pravo pisanja [w] omogućava i mijenjanje sadržaja datoteke, što uključuje i upotrebu operatora za redirekciju (> ili >>). Za razliku od *Windows* operativnih sustava, u *UNIX/Linux* sustavima se pravo izvršavanja odnosno pokretanja datoteke dodjeljuje tako što se određenoj datoteci dodaje pravo izvršavanja [x] (engl. ***execute***), bez obzira na ekstenziju datoteke. U **Windows** svijetu datoteka mora imati ekstenziju poput primjerice `.exe` kako bi bila izvršna odnosno da bi se mogla pokrenuti. Dok u *UNIXu ili Linuxu* to nije potrebno, dovoljno je samo postaviti pravo izvršavanja odnosno ***execute*** ovlast na bilo koju datoteku, naravno definirano za: vlasnika, grupu ili za sve ostale, koji će ju smjeti pokrenuti odnosno izvršavati. Od tog trenutka ova datoteka postaje izvršna, neovisno o njenoj ekstenziji.

**Za direktorije vrijedi sljedeće**
Pravo izvršavanja [x] važno je i za direktorije jer bez njega nije moguće: promijeniti radni direktorij u (ovaj) direktorij (primjerice s naredbom `cd`), pokrenuti programe i skripte unutar tog direktorija ili pregledavati sadržaj direktorija (primjerice s naredbom `ls`). Pravo čitanja [r] omogućuje čitanje sadržaja direktorija (primjerice s naredbom `ls`). Dok pravo zapisivanja [w] omogućuje: kreiranje novih datoteka, kopiranje datoteka u direktorij (primjerice s naredbom `cp`), ili micanja (primjerice s naredbom `mv`) odnosno brisanja datoteka unutar direktorija (primjerice s naredbom `rm`).

## Napredne ovlasti

Osim standardnih ovlasti (r, w i x) postoje i napredne ovlasti, a to su:

- `Set UID` (oktalno **4**, simbolički s ili S) – postavlja "*User ID*" što omogućava trenutnom korisniku da dobije prava koja zapravo ima sâmo vlasnik određene datoteke.
- `Set GID` (oktalno **2**, simbolički s ili S) – postavlja "*Group ID*" odnosno omogućava trenutnom korisniku da dobije prava koja zapravo ima sâmo vlasnička grupa nad određenom datotekom.
  - `Set UID` i `Set GID` ovlasti se postavljaju kada je potrebno dozvoliti korisnicima koji inače nemaju ovlasti nad određenim izvršnim datotekama, kako bismo dobili ovlasti koje ima njihov vlasnik (Set UID) ili njihova vlasnička grupa (Set GID).
- `Sticky bit` (oktalno **1**, simbolički t) - služi za označavanje datoteka ili direktorija (mapa) koji ne smiju biti obrisani, odnosno ovo je ovlast pomoću koje ih može obrisati sâmo vlasnik ili *root* korisnik.

Bilo koji korisnik koji primjerice pokrene izvršnu datoteku koja ima postavljen ***Set UID*** ili ***Set GID*** dobiva ovlasti njenog vlasnika tj. njegovog ***UID**-a* (*User ID* – korisnički identifikacijski broj) ili njegove pripadajuće grupe ***GID**-a* (*Group ID* – identifikacijski broj grupe). To je i potencijalan sigurnosni problem, jer u slučaju malicioznih radnji, problematični korisnik dobiva prava vlasnika (***UID***) ili vlasničke grupe (***GID***) od pripadajuće, a obično izvršne datoteke.
S druge strane neki programi koji su u širokoj upotrebi, moraju biti pokrenuti s višim ovlastima od ovlasti "običnih" korisnika, a obično s ovlastima *root* korisnika. Stoga se u razvoju ovakvih programa, posebna pažnja posvećuje sigurnosti. Neki od programa koji se moraju izvršavati sa **SETUID** ovlastima su: `su`, `mount`, `umount`, `sudo`, `passwd` i slični.

Pogledajmo kako počevši od *root* direktorija (/) pronaći sve datoteke koje imaju postavljen **SETUID**, pomoću naredbe `find`:

```
find / -xdev -type f -perm /4000
```

Pogledajmo kako počevši od *root* direktorija (/) pronaći sve datoteke koje imaju postavljen **SETGID**, pomoću naredbe `find`:

```
find / -xdev -type f -perm /2000
```

**Izvori informacija: (107),(108),(1598), `man 5 acl`,`man find`.**

### 4.3.1. Što nam govore ovlasti

Vratimo se na osnovne ovlasti. Ovlasti nad direktorijem ili datotekom se čitaju od lijevo na desno i kod izlistanja datoteka ili direktorija se nalaze s lijeve strane nakon prve oznake koja označava radi li se o datoteci (oznaka `-`) ili direktoriju (oznaka **`d`**).
Izlistajmo vršno stablo direktorija (*root* `/`), a pogledat ćemo samo `/etc` direktorij pomoću naredbe `ls` na sljedeći način:

```
ls -al

drwxr-xr-x 20 root root 4096 Jun 13 12:53 /etc
```

Sada pogledajmo samo ove oznake s lijeve strane (zatamnjeno)

```
drwx r-x r-x .. .. . .  ..  ..   .     . /etc
```

Prvi znak s lijeve strane (ovdje je to slovo **`d`**) ima posebno značenje te predstavlja direktorij.



Za detalje o direktorijima (mapama) pogledajte poglavlje:
**4.2 Direktoriji (mape) i datoteke**.

Nakon prvog znaka (**`-`** ili **`d`**) koji smo objasnili, u konkretnom primjeru slijede naredni nîzovi:

- Prvi nîz od tri znaka (oktet) čine oznake za vlasnika (*owner*); u ovom slučaju su to prve tri oznake **`rwx`**.
- Drugi nîz od tri znaka (oktet) čine oznake za grupu (*group*); u ovom slučaju je to drugi nîz od **`r-x`**.
- Treći nîz od tri znaka (oktet) čine oznake za sve ostale (*others*); u ovom slučaju je to treći nîz od **`r-x`**.



Postoje i dodatne ovlasti koje se tiču ***SELinux*** sustava pa stoga pogledajte poglavlje:
**28.2. SELinux sustav.**

Međutim, postoji i iznimka za zadnji znak unutar svakog okteta. Naime, na mjestu znaka `x` (*execute*) odnosno prava izvršavanja/pokretanja, mogu stajati sljedeći znakovi koji imaju posebnu svrhu odnosno namjenu.
Dakle iznimke mogu biti znakovi:

- `s` ili `S` ili `t`, i to svaki prema svom značenju:
  - ***Setuid*** (`s`) odnosno ovlast posebnog izvršavanja kada se program izvršava s ovlastima ne onoga koji ga pokreće, već vlasnika programa.
  - **Setgid** (`s`) odnosno ovlast posebnog izvršavanja kada se program ne izvršava s ovlastima vlasnika programa poput ***setuid***, već s ovlastima korisničke grupe u kojoj je vlasnik koji ga je kreirao. *Setgid* se može postaviti i na direktorije (mape).
- ***Sticky*** (`t`) - što znači da preimenovanje ili brisanje datoteke može raditi samo vlasnik (*owner*) iste.

Pogledajmo datoteku `/usr/bin/passwd` s vidljivo postavljenim ***setuid*** (`s`) na poziciji ovlasti za vlasnika (`root` konkretno):

```
-rwsr-xr-x 1 root root 59680 May 17  2017 /usr/bin/passwd
```

To za ***setuid*** datoteku odnosno naredbu `passwd` znači, da kada ju bilo tko pokrene, ona će se pokrenuti ne sa ovlastima onoga tko ju pokreće već sa konkretno ovlastima vlasnika ove datoteke, a to je `root` korisnik (*administrator* UNIX/Linuxa-a).

Pogledajmo i datoteku `/usr/bin/wall` u kojoj je postavljen ***setgid*** (`s`) na poziciji ovlasti za grupu vlasnika (`tty` konkretno):

```
-rwxr-sr-x 1 root tty 27448 Mar  7  2018 /usr/bin/wall
```

To za ***setgid*** datoteku odnosno konkretnu naredbu `wall` znači, da kada ju bilo tko pokrene, ona će se pokrenuti ne s ovlastima korisničke grupe onoga tko ju pokreće već konkretno s ovlastima vlasničke grupe ove datoteke. U ovom primjeru to je grupa `tty` koja ima prava rada odnosno pristupa terminalima Linuxa, pa može poslati tekstualnu poruku na sve virtualne terminale i konzole svih logiranih korisnika na sustav. Izlistavanjem s naredbom `ls` vidljive su nam simboličke oznake ovlasti (`r`, `w`, `x`).
Osim simboličkih oznaka, moguće je mijenjati ih i na druge načine.

Usporedba drugih metoda označavanja ovlasti vidljiva je u ispisu u tablici:

| Binarna oznaka (rwx) | Ovlasti | Simbolička notacija | Oktalna notacija |
|---|---|---|---|
| `111` | Sve ovlasti (r+w+x) | **`-rwxrwxrwx`** | `0777` |
| `110` | Čitanje i pisanje (r+w) | **`-rw-rw-rw-`** | `0666` |
| `101` | Čitanje i izvršavanje (r+x) | **`-r-xr-xr-x`** | `0555` |
| `100` | Samo čitanje (r) | **`-r--r--r--`** | `0444` |
| `011` | Pisanje i izvršavanje (w+x) | **`-wx-wx-wx`** | `0333` |
| `010` | Samo pisanje (w) | **`-w--w--w-`** | `0222` |
| `001` | Samo izvršavanje (x) | **`--x--x--x`** | `0111` |
| `000` | Nema ovlasti (*prazno*) | **`----------`** | `0000` |

Obratite pažnju na *oktalne* oznake jer sve počinju s nula (0).
Ova početna nula označava *posebne ovlasti* o kojima smo govorili, a koje ćete dodatno upoznati kroz primjere koji slijede.

Kao *root* (administrator) kreirali smo datoteku `test` s naredbom: `touch test`. Pogledajmo ju s naredbom:

```
ls -al test
-rw-r--r-- 1 root root 24 2014-07-03 09:54 test
```

Vidimo kako vlasnik (`root`) ima pravo odnosno ovlasti: čitanja (`r`) i zapisivanja (`w`), a svi koji pripadaju grupi (`root`) imaju samo pravo čitanja (`r`) kao i svi ostali koji jedino imaju pravo čitati (`r`) ovu datoteku.

## Na koje načine možemo mijenjati ovlasti?

Ovlasti možemo mijenjati na sljedeće načine:

**1.Pogledajmo opisno definiranje ili mijenjanje ovlasti, koje koristi sljedeće ključne oznake**

- Prvo definiramo za koga ćemo mijenjamo ovlasti; pri tome:
  - `a` – označava sve: i vlasnik i grupa i svi ostali.
  - `u` – označava sâmo vlasnika (engl. *owner : user*).
  - `g` – označava grupu (engl. *group*).
  - `o` – označava sve ostale (engl. *other*).
- Potom slijedi operator; dodajemo li ili oduzimamo ovlasti;
  - `+` znači kako dodajemo ovlasti.
  - `-` znači kako oduzimamo ovlasti.
- Te slijedi naputak koje ovlasti mijenjamo:
  - `r` – ovlast čitanja (engl. *read*).
  - `w` – ovlast pisanja (engl. *write*).
  - `x` – ovlast izvršavanja/pokretanja (engl. *execute*)
- I konačno možemo koristiti i posebne ovlasti
  - `s` – označava SUID ili GUID: ovisno da li se nalazi pod "*user*" ili "*group*" dijelom.
  - `t` – označava takozvani *Sticky bit.*

**a.)** U sljedećem primjeru omogućimo svima ostalima (*other*) (ne odnosi se na vlasnika i vlasničku grupu) mogućnost izvršavanja odnosno pokretanja datoteke `test`. To ćemo postići pomoću naredbe `chmod` na sljedeći način:

```
chmod o+x test
```

Pogledajmo sada što smo dobili. Konkretno smo dobili oznaku `x` na kraju oznaka ovlasti, koja datoteci daje pravo izvršavanja:

```
ls -al

-rw-r--r-x 1 root root 24 2014-07-03 09:54 test*
```

2. **Oktalno definiranje ili mijenjanje ovlasti.** Prema tablici, za svakog korisnika, imamo brojeve koji označavaju: pravo čitanja: `r` (4), zapisivanja: `w` (2) ili izvršavanja: `x` (1).
   Stoga prema tablici imamo sljedeće ovlasti (opisno i brojčano), pa možete vidjeti vezu među njima:

| r | w | x |
|---|---|---|
| 4 | 2 | 1 |

**Pogledajmo i pretvaranje simboličkih u *oktalne* oznake, prema prethodnoj tablici:**

| **1** | **2** | **3** (2+1) | **4** | **5** (4+1) | **6** (4+2) | **7** (4+2+1) |
|---|---|---|---|---|---|---|
| x | w | w+x | r | r+x | r+w | r+w+x |

To znači da, ako želimo postaviti ovlast: **w**+**x** (*write i execute*), da će to biti vrijednost **3**; jer se zbrajaju: **2**(*w*) i **1**(*x*).

Tada se povezivanje odnosno mapiranje u oktalni prikaz označava za svakog od:

- Vlasnika (engl. *owner/user).*
- Grupe (engl. *group*).
- I svih ostalih koji nisu vlasnik ili unutar grupe (engl. *other users*).

Pri tome, primjerice `755` znači:

- `7` – (prvi broj) – za vlasnika (*owner*); što znači: `r+w+x`
- `5` – (drugi broj) – za grupu (*group*); što znači: `r+x`
- `5` – (treći broj) – za sve ostale (*other users*); što znači: `r+x`

Odnosno opisno `755` znači slijedeće:

- `7` – vlasnik (*owner*) ima prava čitanja (`r`), pisanja (`w`) i izvršavanja/pokretanja (`x`).
- `5` – grupa (*group*) ima prava čitanja (`r`) i izvršavanja/pokretanja (`x`).
- `5` – svi ostali (*other users*) imaju prava čitanja (`r`) i izvršavanja/pokretanja (`x`).

**Primjer:** Promijenimo za:

- Vlasnika (*owner*), da može čitati, zapisivati i izvršavati/pokretati: `r`+`w`+`x` (**7**).
- Grupu (*group*), da može samo čitati i pokretati: `r`+`x` (**5**).
- Za sve ostale koji nisu vlasnik ili unutar vlasničke grupe (*others*), da mogu samo čitati i pokretati: `r`+`x` (**5**).

Napravimo ovakvu promjenu ovlasti na datoteci `test` s naredbom `chmod` na slijedeći način:

```
chmod 755 test
```

Sada pogledajmo kako izgledaju novo napravljene ovlasti:

```
ls -al

-rwxr-xr-x 1 root root 24 2014-07-03 09:54 test
```

**Napredno**

Zapravo kod oktalnog dodjeljivanja ovlasti ispred prva tri broja uvijek stoji nula (0), koja označava posebne ovlasti.
Tablica prikazuje posebne ovlasti (koje su inače nula [**0**]) koje se izračunavaju slično kao i obične, prema sljedećem pravilu:

| Set UID | Set GID | Sticky bit |
|---|---|---|
| 4 | 2 | 1 |

Proširena tablica sa ovlastima na kraju izgleda ovako:

| **7** (4+2+1) | **6** (4+2) | **5** (4+1) | **4** | **3** (2+1) | **2** | **1** | **1** | **2** | **3** (2+1) | **4** | **5** (4+1) | **6** (4+2) | **7** (4+2+1) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Set UID + Set GID + Sticky bit | Set UID + Set GID | Sticky bit + Set UID | Set UID | Sticky bit + Set GID | Set GID | Sticky bit | x | w | w+x | r | r+x | r+w | r+w+x |

Dakle ako za ovlasti koje smo postavili u primjeru od gore (**755**) želimo dodati i *Set UID* tada će to oktalno biti: **4755**.

```
chmod 4755 test
```

A isto smo mogli napraviti i simbolički na sljedeći način:

```
chmod u+s test
```

Pogledajmo što smo sada napravili s gornjim naredbama:

```
ls -al
-rwsr-xr-x 1 root root 0 Mar 21 20:13 test
```

Vidljivo je da je treća ovlast za vlasnika iz `x` postala `s`. To znači kako je postavljen *Set UID* za navedenu datoteku, za vlasnika.

Osim toga, moguće je i oduzeti primjerice izvršni (*execute)* dio ovlasti za vlasnika (`u`=Engl. *User*), što ćemo napraviti sa:

```
chmod u-x test
```

Pogledajmo sada novo stanje iste datoteke:

```
ls -al

-rwSr-xr-x 1 root root 0 Mar 21 20:13 test
```

Sada se malo slovo `s` promijenilo u veliko `S`. Dakle sada imamo datoteku koja nije izvršna, ali ima postavljen ***SUID***.

Vratimo sve na početno stanje sa sljedećom naredbom:

```
chmod 0755 test
```

Pogledajmo i kako sve sada izgleda:

```
ls -al

-rwxr-xr-x 1 root root 0 Mar 21 20:13 test
```

Sada vlasnik ima `rxw` prava, grupa `r-x` prava te svi ostali imaju `r-x` prava.
Dakle i vlasnik i grupa i svi ostali mogu čitati i pokretati ovu datoteku (`r` i `x`), ali samo vlasnik ju može uređivati (editirati) odnosno mijenjati (`w`).

Postavimo i ***GUID*** prava; dakle prava koja se primjenjuju na grupu.

```
chmod g+s test
```

Pogledajmo kako to sada izgleda:

```
ls -al

-rwxr-sr-x 1 root root 0 Mar 21 20:13 test
```

Vidljiva je slična situacija kao kod ***SUID***a, ali na poziciji za grupu. Umjesto grupnih ovlasti `r-x` sada imamo `r-s`.

Dodatno i ovdje je moguće maknuti "*Execute*" dio ovlasti za grupu, ali da ostane ***GUID*** ovlast, na sljedeći način:

```
chmod g-x test
```

Pogledajmo kako i to izgleda:

```
ls -al

-rwxr-Sr-x 1 root root 0 Mar 21 20:13 test
```

Sada ponovno, prema ovlastima grupe imamo umjesto malog slova `s` veliko slovo `S`.

## Gdje se u praksi koriste posebne ovlasti?

U slučajevima kada obični korisnik, tijekom pokretanja nekog programa mora, sâmo za potrebe pokretanja tog programa, imati veće (snažnije) ovlasti; obično ovlasti *root* korisnika (UNIX/LINUX *Administrator*), koriste se ove (posebne) ovlasti.
Ova potreba postoji kod raznih mrežnih naredbi, koje "obični" korisnik ne bi mogao niti pokrenuti jer nema prava korištenja određenih resursa sustava. Najjednostavniji školski primjer je naredba `ping` za testiranje dostupnosti računala na mreži.

Pogledajmo ovlasti ove naredbe, koja se inače nalazi u direktoriju `/bin/`. Pogledajmo ju s naredbom `ls` na sljedeći način:

```
ls -al /bin/ping

-rwsr-xr-x 1 root root 36476 Jul 23  2015 /bin/ping
```

Vidljivo je kako je postavljen ***SUID*** (***X + SUID***) na trećoj poziciji za vlasnika (koji je u ovom slučaju korisnik *root*) stoji malo slovo `s`. Još kritičniji primjer je naredba `passwd` s kojom mijenjamo svoju korisničku lozinku (Engl. *password*).
Naime sve lozinke su pohranjene u datoteci `/etc/shadow` i sâmo ***root*** korisnik ima puna prava mijenjanja ove datoteke.



Za detalje pogledajte poglavlje:
**7.1.3 Datoteka /etc/shadow.**

Pošto svaki korisnik mora imati prava promijeniti svoju lozinku, to znači da bi svaki korisnik morao imati prava pristupa ovoj datoteci. To u praksi nije u potpunosti tako. Dakle samo ***root*** ima puna prava direktnog pristupa ovoj datoteci te pristupa preko naredbe koja je zadužena za promjenu lozinke (naredbi `passwd`) za pojedinog korisnika.
Ovoj naredbi su stoga dodijeljena ***Set UID*** prava. Pogledajmo kako ona izgleda:

```
ls -al /usr/bin/passwd

-rwsr-xr-x. 1 root root 25980 Feb 22  2012 /usr/bin/passwd
```

Ovdje vidimo oznaku/slovo `s` na poziciji vlasnika datoteke/programa (u ovom slučaju `root` korisnika).
Nadalje ova naredba je pisana tako da ona pri svakom pokretanju provjerava koji korisnik ju je pokrenuo. Tada ona omogućava korisniku koji ju je pokrenuo da može mijenjati datoteku `/etc/shadow` i to sâmo i isključivo onaj dio (redak) koji se tiče korisnika koji ju je pokrenuo. Konkretno on može mijenjati samo jedan redak u kojem je pohranjena lozinka za trenutnog korisnika, i to stupac u tom retku u kojem je stvarno pohranjena njegova lozinka.
To se upravo i događa kod pokretanja ove naredbe, koja se zbog postavljenog ***Set UID*** prava pokreće sa *root* pravima odnosno ovlastima administratora (*root*).

## Još "*posebniji*" slučajevi

Moguće je dodijeliti ***Set GID*** ovlasti i na direktorije (mape) s naredbom `chmod g+s`.
S ovime dobivamo mogućnost da novo kreirani direktoriji nasljeđuju ***Set GID*** postavljeni bît.

To znači kako će svaka novo kreirana datoteka ili pôddirektorij i sve datoteke u njemu, naslijediti mogućnost korištenja datoteka s posebnim ovlastima od grupe čiji je ***Set GID*** postavljen.
Ova mogućnost vrijedi sâmo za sve nove datoteke koje su kreirane nakon postavljanja ***Set GID*** na određeni direktorij.
U slučaju potrebe kada određenom postojećem direktoriju (i svim postojećim datotekama unutar njega, nasljeđivanjem) želimo postaviti ***Set GID*** ovlasti, potrebno je napraviti sljedeće:

**Primjer 1** - upotrebom naredbe: `find` s kojom u pretrazi tražimo direktorij (`-type d`) i potom mijenjamo ovlasti**:**

```
find /putanja-do-zeljenog-direktorija/direktorij -type d -exec chmod g+s '{}' \;
```

**Primjer 2** - direktno upotrebom naredbe `chmod` rekurzivno (prekidač `-R`)**:**

```
chmod -R g+s /putanja-do-zeljenog-direktorija/direktorij
```

## Sumirajmo priču o ovlastima odnosno sve do sada naučeno.

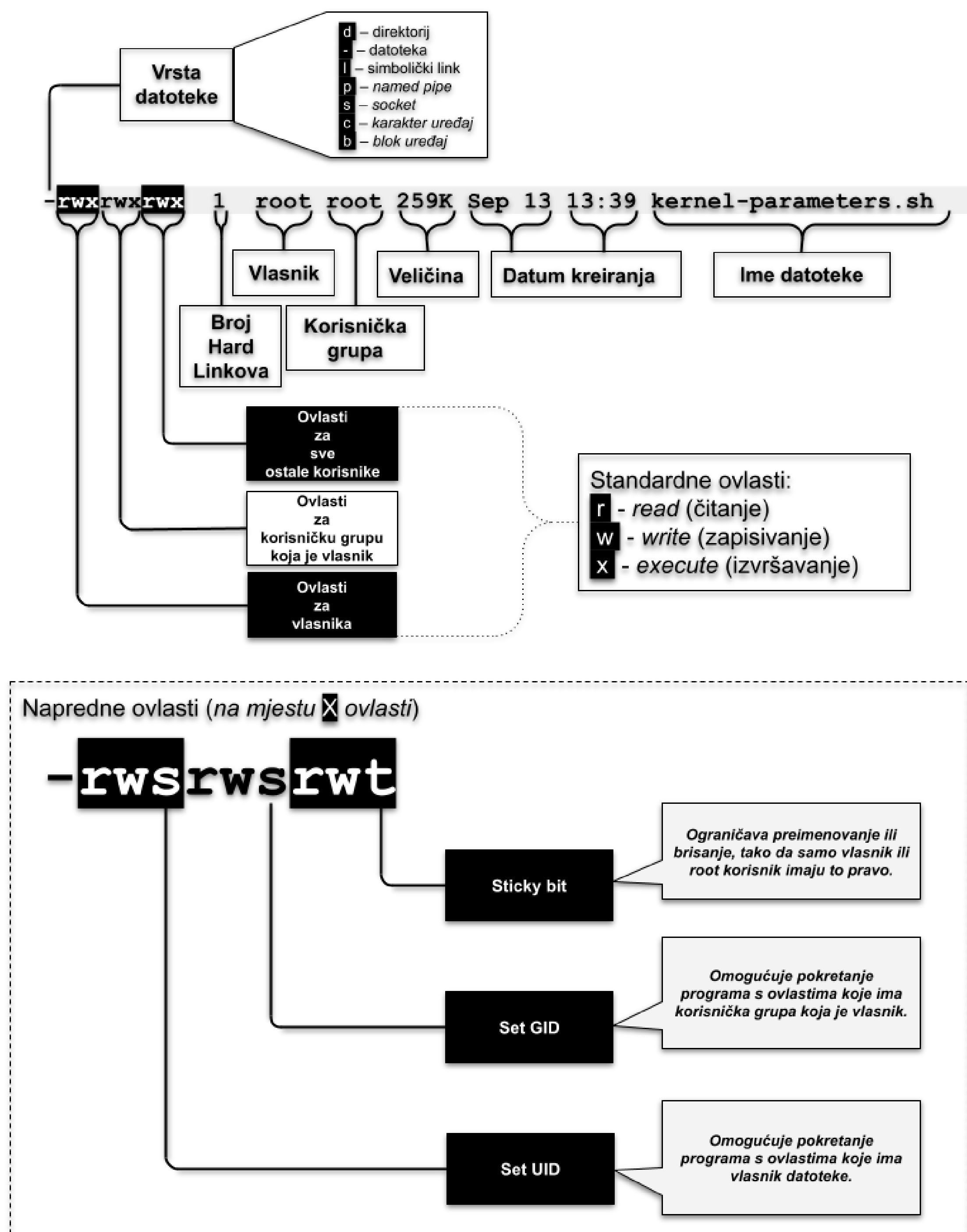
Pogledajmo ispis trenutnog direktorija, pomoću naredbe `ls` na sljedeći način:

```
ls -alh

total 1.2M
drwxr-xr-x    4 root root  168 Oct 30 11:36 .
dr-xr-x---. 22 root root 4.0K Jan 30 11:23 ..
-rw-r--r--    1 root root 2.9K Oct 30 11:36 GNU.projekti.txt
drwxr-xr-x    2 root root   23 Aug 30 09:55 PDF
-rwxrwxrwx    1 root root 259K Sep 13 13:39 kernel-parameters.sh
```

Malo smo povećali veličinu ispisa, kako bismo lakše uočili detalje.
Za cijeli sažetak pogledajte sliku 8.1. s naglaskom na zadnju datoteku iz gornjeg ispisa.

**Slika 8.1. Sažeti pogled na ovlasti.**

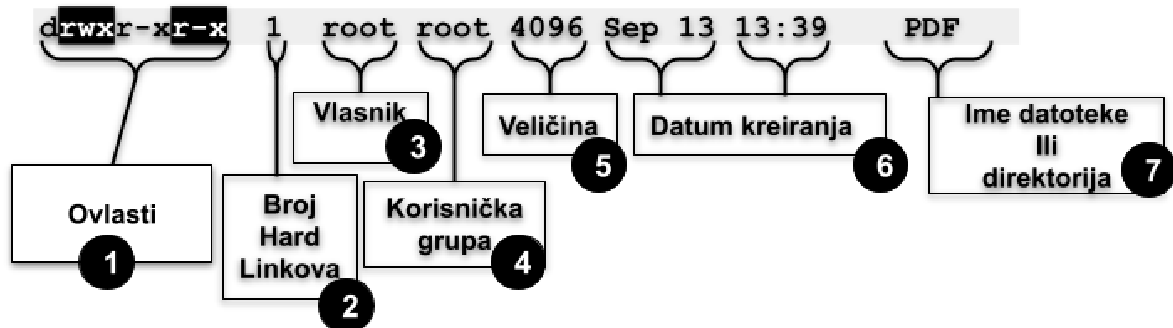



**Izvori informacija:** (107),(108),(109),(110),(K-1),(K-12), `man chmod`,`man find`,`man ls`,`man 5 passwd`.

# 4.4. Rad s direktorijima i datotekama detaljnije

U ovoj cjelini upoznat ćemo se s naprednim detaljima oko ovlasti (tzv. *atributima*) te mogućnostima izvršavanja datoteka i programa (tzv. *capabilities*). Za početak se prisjetimo ovlasti koje smo spominjali u prethodnoj cjelini, a saželi smo na prethodnoj stranici. Sada ćemo vidjeti kako izgledaju ovlasti, te što je još vidljivo na razini datoteka i direktorija (mapa) po tom pitanju.
Na slici 9. pogledajmo primjer izlistanja direktorija naziva: `PDF`.

**Slika 9. Ovlasti i druge oznake datoteka i direktorija (mapa).**




Što je ovdje vidljivo?

- (1) – u ovom dijelu su ovlasti, prethodno objašnjene; redom za: vlasnika (***rwx***), grupu (***r-x***) i sve ostale (***r-x***).
- (2) – ovo je brojčana vrijednost koja govori o broju *hard linkova* (o njima nešto kasnije):
  - Za datoteke je to broj *hard linkova* na tu datoteku, a ako nema drugih *hard linkova* na datoteku, tada je to **1**.
  - Za direktorije je to broj poddirektorija +2 ***(ako ih nema onda je 2, ako sadrži samo jedan poddirektorij onda je vrijednost 3)***.
- (3) – ovdje je zapisano tko je vlasnik datoteke ili direktorija (u ovom slučaju je to korisnik `root`).
- (4) – ovdje vidimo kojoj korisničkoj grupi pripada direktorij (u ovom slučaju je to korisnička grupa `server`).
- (5) – ovdje je vidljiva veličina:
  - Za *datoteke* ovo je veličina datoteke, standardno izražena u bajtima.
  - Za *direktorije*, ovo je veličina za *meta podatke* za datoteke unutar tog direktorija; obično je to standardna veličina bloka datotečnog sustava. Primjerice za datotečne sustave **ext2/3/4** je to uobičajeno **4096**, ali može i rasti.
- (6) – ovdje je vidljiv datum zadnje izmjene na datoteci ili direktoriju.
- (7) – i ovdje na samom kraju je vidljivo ime datoteke ili direktorija (mape).

U cjelinama koje slijede upoznat ćemo se i s drugim detaljima.

---

## 4.4.1. Ovlasti kod kreiranja nove datoteke ili direktorija i naredba *umask*

**Slijedi napredna cjelina!**
Kada kreiramo bilo koji novi direktorij (mapu) ili datoteku ona se kreira s određenim definiranim ovlastima prema *maski* ovlasti, a koje se postavljaju s naredbom `umask` čija standardna vrijednost je obično postavljena na: ***022***.

Zadane dozvole za stvaranje mogu se mijenjati pomoću uslužnog programa `umask`. Važno je razumjeti da *umask* utječe samo na trenutno okruženje ljuske. Postavke *umask* vrijednosti se mogu spremiti primjerice u datoteku `/etc/profile` koja se primjenjuje za sve korisnike sustava tijekom spajanja na sustav. Međutim, ako želite navesti neku drugu vrijednost za svakog pojedinog korisnika, uredite korisničke konfiguracijske datoteke (*bash*) ljuske kao što su `~/.bashrc` ili `~/.bash_profile`. Važno je znati da korisničke datoteke imaju prednost nad globalnim datotekama. Promijeniti trenutnu vrijednost *umask* sesije također možete pokretanjem *umask* naredbe nakon čega slijedi postavljanje željene vrijednosti. Trenutno postavljenu masku ovlasti (*umask*) za kreiranje datoteka i direktorija možemo vidjeti s pozivanjem naredbe `umask` bez argumenata:

```
umask

0022
```

← vidimo da je *umask* vrijednost postavljena na `0022`.
Odnosno ako želimo opisne oznake, možemo ju pozvati sa `umask -S`:

```
umask -S

u=rwx,g=rx,o=rx
```

U većini slučajeva je *umask* vrijednost za `root` i druge sistemske korisnike **`022`** dok je za obične korisnike ona postavljena na **`002`**. Vrijednost *umask* sadrži bitove dopuštenja koji NEĆE biti postavljeni na novostvorene datoteke i direktorije.
Ako maska ima bit postavljen na "1", to znači da će odgovarajuća početna dozvola za datoteku biti onemogućena.
Bit postavljen na "0" u maski znači da će odgovarajuće dopuštenje odrediti program i sustav.

**Kako se izračunava konačna (efektivna) maska ovlasti od postavljene *umask* vrijednosti?**
Prvo, koristi se standardna tablica za ovlasti, kod koje se vrijednost zbrajaju da bi se dobila konačna vrijednost(tablica dolje).
Ne zaboravimo da je u većini slučajeva *umask* vrijednost za `root` i druge sistemske korisnike **`022`** dok je za obične korisnike ona obično postavljena na **`002`**.

| | | |
|---|---|---|
| r | w | x |
| 4 | 2 | 1 |

Efektivna vrijednost ovlasti se dobiva oduzimanjem ***umaska*** od maksimalne vrijednosti i to za svaki broj zasebno, jer svaki od brojeva i predstavlja ovlasti za određenog korisnika (*vlasnik*, *grupa* i *ostali*):

- Za ***direktorije***: maksimalna vrijednosti za direktorije je `777`, pa je računica: `777-022=755`
- Za ***datoteke***: maksimalna vrijednosti za datoteke je `666`, pa je računica: `666-022=644`

Odatle je za prvi broj: 7-0=`7`; za drugi broj 7-2=`5` i za treći broj 7-2 jest isto `5`, pa imamo kao rezultat `755` za direktorije, a po istoj računici imamo vrijednost `644` za datoteke.

To znači da će kod svakog kreiranja novog direktorija, direktorij biti kreiran sa ovlastima `755` odnosno `rwxr-xr-x`, dok će novokreirane datoteke imati ovlasti `644` odnosno to će biti vidljivo kao: `rw-r--r--`.

**Pogledajmo primjere.**
Za datoteku `/root/podaci/test.sh`, tko god joj bio vlasnik, promijenimo joj vlasnika u `pero` i vlasničku grupu u `server` s naredbom `chown` (Engl. *Change ownership*) na sljedeći način:

```
chown pero:server /root/podaci/test.sh
```

Moguće je mijenjati vlasnika i vlasničku grupu i rekurzivno; za sve pôd direktorije unutar nekog vršnog direktorija, kao i za sve datoteke unutar njih, s prekidačem `-R` naredbe `chown`. Primjerice promijenimo vlasnika u `pero` i vlasničku grupu u `server` za sve direktorije i poddirektorije, kao i sve datoteke unutar njih, počevši od vršnog direktorija: `/root/podaci/`:

```
chown -R pero:server /root/podaci/
```



Postavljanje ***umask*** vrijednosti se radi i:

- Za sve sistemske servise, kako je definirano u datoteci: `/etc/init.d/functions`
- Kod kreiranja novih korisnika i njihovih *home* direktorija, kako je definirano u datoteci: `/etc/login.defs`

→ **Opisano u poglavlju: 7.1.6.2. Datoteka /etc/login.defs.**

Promijenimo ovlasti na datoteci `/root/podaci/test.sh` da bude i izvršna (*read*+*write*+*execute*), samo za vlasnika iste:

```
chmod 0744 /root/podaci/test.sh
```

Ako vam je jednostavnije tako, možemo zadavati i konfigurirati *umask* vrijednosti i opisno, pri tome imamo klase korisnika:

- `u` – označava vlasnika (engl. *owner*).
- `g` – označava korisničku grupu koja je vlasnik (engl. *group*).
- `o` – označava sve ostale (engl. *other*).
- `a` – označava sve ovlasti zajedno (engl. *all*) [`u`+`g`+`o`].

Dok operatori mogu biti neki od sljedećih:

- `+` – označava da se ovlast u masci dodaje.
- `-` – označava da se ovlast u masci oduzima (briše).
- `=` – označava da se ovlasti omoguće za navedene klase korisnika.

I na kraju slijede ovlasti koje koristimo su klasične ovlasti:

- `r` – označava ovlast za čitanje.
- `w` – označava ovlast za zapisivanje.
- `x` – označava ovlast za izvršavanje.

Sintaksa je: **umask** *KLASA-korisnika OPERATOR OVLAST*

Primjerice za *umask* masku 0**022** koja bi opisno bila: (`u=rwx,g=rx,o=rx`) , možemo ju zadati opisno i ovako:

```
umask u+w
```

Ova ovlast konkretno znači da se postavlja maska tako da kada se kreiraju datoteke, one imaju dopuštenja (ovlasti) koja dopuštaju pravo pisanja za korisnika (vlasnika datoteke). Ostatak dopuštenja za datoteku ne bi se mijenjao u odnosu na zadane postavke operativnog sustava.

Moguće je zadavati i složenije *umask* maske, primjerice **0130**:

```
umask u-x,g=r,o+w
```

Ova maska brani pravo izvršavanja (*execute*) za vlasnika (`u`) dok ne dira druge ovlasti za vlasnika. Za grupu (`g`) dozvoljava samo prava čitanja (`r`) dok brani pravo pisanja i izvršavanja. I na kraju za sve ostale (`o`) dodaje pravo zapisivanja ne dirajući (ne mijenjajući) ostale ovlasti.

***Umask vrijednost se može definirati i unutar bilo koje shell skripte, ako za to imamo potrebe.***



Vezano za promjene ovlasti s naredbom **`chmod`**, pogledajte i poglavlje:
**4.3. Ovlasti (*permissions* & *modes*).**

Prije nego što promijenite vrijednost *umask-a*, provjerite da nova vrijednost ne predstavlja potencijalni sigurnosni rizik. Vrijednosti koje su manje restriktivne od **022** trebaju se koristiti s velikim oprezom. Na primjer, *umask* **000** znači da svatko ima dopuštenja za čitanje, pisanje i izvršavanje svih novostvorenih datoteka.

Recimo da želimo postaviti restriktivnija dopuštenja za novostvorene datoteke i direktorije kako drugi ne bi mogli prelaziti u direktorije i čitati datoteke. Dozvole koje želimo su **750** za direktorije i **640** za datoteke. Navedena željena vrijednost *umask* predstavljena u numeričkom zapisu je tada **027**. Da biste trajno postavili novi sustav vrijednosti, otvorite datoteku `/etc/profile` u uređivaču teksta i na kraj dodajte:

```
umask 027
```

Da bi promjene stupile na snagu, pokrenite sljedeću naredbu (ili se odjavite i prijavite na sustav):

```
source /etc/profile
```

Drugi način postavljanja maske za stvaranje datoteke je korištenje simboličke notacije. Na primjer, `umask u=rwx,g=rx,o=` je isto što i **umask 027**.
Nadalje, u centralnoj datoteci `/etc/login.defs` moguće je definirati umask vrijednost, koja je standardno postavljena na:

```
UMASK           022
```

Moguće je postavke stavljati i u **PAM** podsustav, dakle u datoteke unutar direktorija `/etc/pam.d/`.
Primjerice dodavanjem unosa poput:

```
session optional pam_umask.so umask=0002
```

Međutim pazite na to da će, ukoliko postoje korisničke *umask* postavke (za interaktivno logiranje), pregaziti ove postavljene u **PAM** datoteci za pojedini servis (pr. `/etc/pam.d/sshd` za *ssh* servis).


Vezano za PAM sustav pogledajte:
**7.1.6.6. Linux PAM (Pluggable Authentication Modules).**

## Umask za servise

U određenim slučajevima možemo imati programe (procese) i servise koji stvaraju datoteke, a ako želimo upravljati ovlastima tih datoteka, najjednostavniji način je definicija *umaska*. Jedan od primjera je **apache** web (*httpd*) poslužitelj koji primjerice stvara log i druge datoteke, a nakon što su stvorene možda ih želimo pregledavati s našim korisničkim računom koji dijeli istu korisničku grupu (pr. grupu **apache**). Naime, ponekad imamo procese i servise koji stvaraju datoteke, a želimo upravljati dopuštenjima tih datoteka ili s nekim skriptama i nakon što su datoteke stvorene želimo izmijeniti ili čitati ove datoteke s našim korisničkim računom koji dijeli istu korisničku grupu (*apache* u konkretnom slučaju). Kako to tada učiniti?
Općenito, potrebno je staviti naredbu *umask* s definiranom željenom vrijednosti u skriptu koja se koristi za pokretanje servisa ili u drugu datoteku uključenu u pokretanje servisa. Za konkretan slučaj je to *umask* vrijednost **002**.

Za **Red Hat** do v.7. može se napraviti sljedeće:

```
echo "umask 002" >> /etc/sysconfig/httpd
```

Na taj način smo u inicijalizacijsku datoteku dodali potrebn željenii *umask*.
Međutim, kod novijih inačica **Red Hat** Linuxa (7+) za tu namjenu je zadužen ***systemd*** servis, pa je potrebno u datoteku: `/usr/lib/systemd/system/httpd.service`, dodati u sekciju `[Service]` naš *umask*, poput ovoga:

```
[Service]
UMask=0002
```

← (Početna **0** je oznaka *oktalnog* zapisa). Zatim je potrebno pokrenuti proceduru za osvježavanje sadržaja servisa:

```
systemctl daemon-reload
```

Tek sada možemo pokrenuti (ili restartati servis):

```
systemctl restart  httpd
```

Pogledajmo i vrlo skraćenu tablicu s *umask* postavkama odnosno pravima koja ona predstavljaju na datoteke i direktorije:

| Maska (umask) | Efektivne ovlasti za datoteke | | Efektivne ovlasti za direktorije | |
|---|---|---|---|---|
| 000 | 666 | rw-rw-rw- | 777 | rwxrwxrwx |
| 002 | 664 | rw-rw-r-- | 775 | rwxrwxr-x |
| 007 | 660 | rw-rw---- | 770 | rwxrwx--- |
| 022 | 644 | rw-r--r-- | 755 | rwxr-xr-x |
| 027 | 640 | rw-r----- | 750 | rwxr-x--- |
| 077 | 600 | rw------- | 700 | rwx------ |
| 277 | 400 | r-------- | 500 | r-x------ |

← **Puna tablica ima 777 mogućih vrijednosti!**

Potpunu tablicu *umask* vrijednost pogledajte na izvoru informacija **(1498)** ili na **poveznici: https://github.com/opensourceosijek/website/blob/master/static/slike/Umask-tablica.txt**.

**Izvori informacija: (109),(110),(1493),(1494),(1495),(1496),(1497),(1498),(K-12),** `man umask`, `man chmod`, `man chown`, `man pam_umask`.

### 4.4.2. Napredne ovlasti (atributi) te naredbe *lsattr* i *chattr*

**Slijedi napredna cjelina!.** Na većini modernih datotečnih sustava na Linuxu, u što se ubrajaju i ***ext2***, ***ext3*** i ***ext4***, ***xfs*** ali i mnogi drugi, osim što se pohranjuju standardne ovlasti (engl. *permissions & modes*) za datoteke ili direktorije (mape) moguće je koristiti i napredne atribute s kojima možemo postavljati neka dodatna ograničenja.
Za rad s ovim atributima možemo koristiti sljedeće naredbe:

- `lsattr` – za ispis odnosno izlistanje ovih naprednih atributa.
- `chattr` – za promjenu ovih naprednih atributa.

Važno je znati da su ovi atributi praktično nadogradnja na standardne ovlasti (`r`, `w` ili `x` za vlasnika, grupu ili ostale). Krenut ćemo s naredbom `chattr` i opcijama koje nam ona nudi, da bi vam bilo jasnije čemu služe ovi atributi. Sintaksa je sljedeća: `chattr` *prekidači operator atribut DATOTEKA/DIREKTORIJ*.
*Prekidači* ove naredbe su kako slijedi: za rekurzivni ispis datoteke i direktorija se koristi: `-R`, a za ispis (i) skrivenih datoteka se koristi `-a`. Nadalje samo za ispis direktorija bez datoteka, možemo koristiti prekidač `-d` (postoje i drugi prekidači ali ih nećemo spominjati).
Zatim slijede *operatori*:

- `+` – operator za dodavanje atributa.
- `-` – operator za brisanje atributa.
- `=` – operator s kojim postavljamo sâmo i isključivo određeni atribut koji slijedi

Atributi su (navodimo sam nekoliko njih):

- `A` – atribut koji označava da se vrijeme pristupa (*atime*) datoteci ili direktoriju neće moći mijenjati.
- `S` – atribut koji naznačava datotečnom sustavu da će se promjene na datoteci odmah snimiti na disk („*no cache*“).
- `a` – atribut koji označava da dozvoljavamo da se datoteka može samo dopunjavati (Engl. *Append*) ali ne i brisati.
- `i` – atribut (Engl. *Immutable*) koji označava da datoteku ne možemo nikako mijenjati (niti kao *root* korisnik).

Sada ćemo krenuti s par primjera za koje smo kreirali prazan direktorij imena `/test` i u njemu tekstualnu datoteku imena: `test.conf`. Prvo izlistajmo napredne atribute za sve datoteke unutar ovog direktorija, upotrebom naredbe `lsattr`

```
lsattr
--------------e---- ./test.conf
```

Vidimo samo standardno postavljen atribut `e`, koji kod Linux ***ext4*** datotečnog sustava označava način zapisivanja podataka (Engl. *Exents*). Ostali atributi nisu postavljeni, što je normalno jer se oni u pravilu **ne postavljaju**, osim u posebnim slučajevima. Sada na našoj datoteci `test.conf` postavimo atribut `i` koji brani ikakve promjene (ili brisanje) iste datoteke:

```
chattr +i test.conf
```

Pogledajmo koje atribute sada imamo postavljene:

```
lsattr test.conf
----i---------e---- test.conf
```

Probajmo sada nešto upisati u tu datoteku (upotrebom preusmjeravanja `>`) te pogledajmo što će se dogoditi (kao *root* korisnik):

```
echo "Test" > test.conf
-bash: test.conf: Operation not permitted
```

Vidimo da nemamo prava zapisivanja, pa ćemo obrisati ovaj atribut i staviti novi atribut, `a` s kojim ćemo moći sâmo dopisivati u datoteku, ali ju nećemo moći isprazniti ili popuniti od početka:

```
chattr -i test.conf
chattr +a test.conf
```

Probajmo popuniti ovu datoteku na način da izbrišemo sav sadržaj i stavimo novi od početka, upotrebom preusmjeravanja `>`

```
echo "Test" > test.conf
-bash: test.conf: Operation not permitted
```

Vidimo da to ne možemo, ali možemo nešto nadodati, sa upotrebom redirekcije `>>` za dopunjavanje sadržaja(ili na bilo koji drugi način)

```
echo "Test" >> test.conf
```

Pogledajmo sada nove atribute ove datoteke:

```
lsattr test.conf
-----a--------e---- test.conf
```



*Atributi su korisni, ako želimo zaštititi neku datoteku, da ju netko (pa i korisnik **root**) ne bi nehotice obrisao ili prepunio novim podacima i slično. To je korisno za neke primjene ili za podizanje razine sigurnosti na vrlo visoku razinu, za one datoteke za koje to možemo napraviti bez bojazni da ćemo poremetiti rad sustava.*
***Ako niste sigurni, bolje je ništa ne mijenjati.***

**Izvori informacija: (820),(821),(822),** `man chattr`,`man lsattr`,`man 7 attributes`, pogledajte animaciju:

## 4.4.3. To nije sve: mogućnosti izvršavanja datoteka i programa (*capabilities*)

**Slijedi napredna cjelina!**
U svrhu izvršavanja provjera dozvola izvršavanja programa (*procesa*), tradicionalna implementacija *UNIXa* (***POSIX 1003.1e, capabilities(7)***), ali i Linuxa kao njenog nasljednika, razlikuju dvije kategorije procesa:

- ***Privilegirani*** procesi čiji je efektivni korisnički ***UID*** nula, odnosno oni koje je pokrenuo takozvani super korisnik odnosno *root* (administrator).
- ***Neprivilegirani*** procesi čiji efektivni ***UID*** broj nije nula odnosno procesi koje pokreću svi ostali korisnici.

Privilegirani procesi zaobilaze posebne dodatne provjere dozvola sustava, dok nepovlašteni odnosno *neprivilegirani* procesi podliježu provjeri potpunih dopuštenja baziranih na vjerodajnicama procesa; obično: ***UID*** broju korisnika, pripadnosti grupi odnosno ***GID*** broju te pripadnosti dopunskim korisničkim grupama. Međutim počevši od kernela 2.2, Linux tradicionalno dijeli privilegije povezane i sa super korisnikom (*root*) u različite kategorije, poznate kao *sposobnosti* odnosno *mogućnosti* rada procesa (engl. *capabilities*) odnosno u konačnici i izvršnih datoteka, a koje se mogu omogućiti ili onemogućiti.

Naime izvršni programi pokretani s *root* ovlastima su glavna meta za hakere; ako mogu iskoristiti neki sigurnosni propust u njima, napadači mogu povećati razinu svojih privilegija na sustavu i samim time kompromitirati cijeli sustav. Navedene kategorije koje se dodjeljuju procesima odnosno inicijalno njihovim izvršnim datotekama su zapravo dodatni atributi koji se poput ovlasti i naprednih atributa zapisuju uz pojedine datoteke, a koje mogu i ne moraju biti postavljene, sve ovisno o namjeni izvršne datoteke odnosno programa. Ove *mogućnosti* (engl. *Capabilities*) nad izvršnim datotekama se zapisuju na datotečni sustav uz datoteke na koje se primjenjuju, u takozvanom polju proširenih atributa (***xattr***). Njih podržavaju svi važniji datotečni sustavi na Linuxu; poput: *Ext2*, *Ext3*, *Ext4*, *Btrfs*, *JFS*, *XFS*, *Reiserfs* i drugih. Međutim datotečni sustavi za posebne namjene poput *proc* i *sysfs* nemaju ovu podršku. Ove *mogućnosti/sposobnosti* primjerice zamjenjuju potrebu za upotrebom ***SETUID*** ovlasti, ali donose i više od toga. U osnovi je cilj *mogućnosti* razdijeliti sposobnosti *root* korisnika na određene kategorije privilegija, tako da, ako imamo proces (pokrenuti program) ili binarnu izvršnu datoteku koja ima jednu ili više ovih *mogućnosti*, potencijalna šteta je ograničena u usporedbi s istim procesom koji bi se izvršio s ovlastima *root* korisnika bez ovih kategorija privilegija.

Navedene *mogućnosti* se mogu postaviti na procese i binarne izvršne datoteke. Program koji pokrećemo (proces), a koji proizlazi iz izvršavanja izvršne datoteke s postavljenim *mogućnostima* (može) naslijediti *mogućnosti* te datoteke.

*Mogućnosti* implementirane na Linuxu brojne su, a mnoge su i dodane od njihovog izvornog izdanja. Neke od njih su sljedeće:

- `CAP_CHOWN` – omogućava promjene na *UID* (*User ID*) i *GID* (*Group ID*) oznake datoteke.
- `CAP_DAC_OVERRIDE` – zaobilazi tzv. *DAC* (*Discretionary Access Control*) sustav. Dakle zaobilazi provjeru standardnih: *read/write/execute* ovlasti.
- `CAP_KILL` – zaobilazi provjeru ovlasti za slanje signala procesima.
- `CAP_SYS_NICE` – koristi se za promjene *nice* vrijednosti procesa (prioriteta).
- `CAP_NET_BIND_SERVICE` – dozvoljava upotrebu portova manjih od 1024 (*sistemski portovi*) drugim korisnicima.
- `CAP_NET_ADMIN` – dozvoljava se rad s tablicama usmjeravanja, konfiguraciju mrežnih sučelja te vatrozida i sl.
- `CAP_NET_RAW` – dozvoljava se upotreba mrežnog *socketa* vrste: *packet* i *RAW*.

*Mogućnosti* se dodatno definiraju prema načinu rada, i to: "*permitted*", "*inheritable*", "*effective*" i "*ambient*" za procese i procesne nîti te "*permitted* (`p`)", "*inheritable* (`i`)" i "*effective* (`e`)" za binarne izvršne datoteke. Pomoću ovih definicija načina rada moguće je definirati vrlo kompleksno ponašanje efektivnih *mogućnosti* odnosno oni koji se u konačnici primjenjuju.

Pogledajmo naredbu `ping` (izvršna datoteka: `/usr/bin/ping`), koja je specifična po tome da su za nju potrebne posebne ovlasti za izvršavanje, za sve korisnike koji nisu administratori (tj. *root*). Pogledajmo ju s naredbom `getcap` ovako:

```
getcap /usr/bin/ping
/usr/bin/ping = cap_net_admin,cap_net_raw+p
```

Vidimo da naša naredba (odnosno njena binarna izvršna datoteka: `/usr/bin/ping`) ima postavljene već objašnjene *mogućnosti*: `CAP_NET_ADMIN` i `CAP_NET_RAW` ali i skup *mogućnosti* s oznakom: `+p`. Ova zadnja mogućnost (`+p`) je skup koji označava „*permitted”* stanje koje pojednostavljeno znači odobravanje navedenih *mogućnosti*.
Moguće je postaviti mogućnost da primjerice pokretanje određene datoteke traži ***SETUID*** ovlasti, ali ne želimo koristiti postavljenje ***SETUID*** ovlasti, već to želimo postići pomoću ovih *mogućnosti*, što je i preporučena (nova) metoda rada.

To možemo postići s naredbom: `setcap` primjerice za ***Apache*** web poslužitelj (***httpd***), na slijedeći način:

- *Kako bi* ***Apache*** *web poslužitelj mogli pokrenuti ali ne kao root korisnik, potrebne su i druge promjene na sustavu*(*osim ove*).

```
setcap cap_net_bind_service=+eip /usr/sbin/httpd
```

Naredbe `getcap` i `setcap` dolaze u ***libcap*** paketu pa ako ih nemate, instalirajte ovaj softverski paket sa:

```
yum -y install libcap
```



***SELinux*** *sustav se može ispreplitati odnosno eventualno i ometati ove mogućnosti (i obratno) pa treba biti oprezan.*



Navedene *mogućnosti* se automatski kopiraju tijekom kopiranja datoteka; primjerice s naredbom: **`cp -a`**.
Međutim neki programi trebaju posebni prekidač, kako bi se one kopirale; poput naredbe: **`rsync -X`**.

*Mogućnosti/sposobnosti* za svaki pokrenuti program odnosno proces, prema njegovom ***`PID`*** broju procesa, se upisuju u datoteku: `/proc/`***`PID`***`/status`, u „`Cap`" dio statistike; pr. za **`PID`** broj **`18081`** bi to vidjeli sa (uz naš komentar poslije znaka # ):

```
grep Cap /proc/18081/status
CapInh: 0000000000000000 → Mogućnosti koje dijete proces može naslijediti
CapPrm: 0000000000000000 → Mogućnosti koje su dozvoljene za upotrebu
CapEff: 0000000000000000 → Mogućnosti koje su efektivno u upotrebi
CapBnd: 0000001fffffffff → Mogućnosti preko kojih se ne može preći (gornja granica)
CapAmb: 0000000000000000 → Mogućnosti za sve koji nisu root (UID 0) korisnik
```

Dekodirati ove vrijednosti možemo s naredbom `capsh`; primjerice ovako: `capsh --decode=0000001fffffffff`.

**Izvori informacija: (935), `man capabilities`,`man getcap`,`man setcap`,`man capsh`.**

## 4.4.4. Niti to nije sve - Access control list (ACL)

Općenito govoreći, a vezano za računalnu sigurnost, pojam *access-control list* (**ACL**) odnosi se na popis kontrole pristupa odnosno svojevrsnih dozvola povezanih s određenim resursom sustava (objektom). Pri tome **ACL** određuje koji korisnici ili sistemski procesi (servisi) imaju pristup resursima, kao i koje su operacije dopuštene na danim resursima. Svaki unos u tipičnoj *ACL* listi definira predmet i operaciju. Primjerice: korisnik *root* ima prava čitanja određene datoteke, što se konkretno smatra definicijom ovlasti (engl. *permissions*) o kojima smo govorili ne početku poglavlja. S druge strane *ACL-om* se smatraju i liste pristupa na mrežne resurse poput: odobravanja ili blokiranja pristupa s mreže na određeni servis; primjerice zabrana pristupa na port 22 (SSH) s određenih IP adresa. Ipak, krenimo redom (s teorijom). Naime *ACL* u načelu mogu biti:

- **ACL na razini datotečnog sustava** – uobičajeno je to tablica koja sadrži unose koji određuju pojedinačna korisnička ili grupna prava na određene sistemske objekte kao što su programi, procesi ili datoteke odnosno direktoriji. Dakle privilegije (ovlasti) ili dozvole određuju određena prava pristupa, kao što je pravo može li korisnik čitati, zapisivati ili izvršavati određeni objekt (u ovom slučaju datoteku).
- **POSIX ACL** – radna skupina **POSIX 1003.1e/1003.2c** je opisala ACL-ove, što je rezultiralo onim što je danas poznato kao POSIX.1e, ACL ili jednostavno POSIX ACL. Većina Unixoidnih operativnih sustava (uključujući Linux) podržava ACL-ove prema POSIX.1e standardu. *Portable Operating System Interface* (**POSIX**) je skup standarda koji definira niz sučelja za operativne sustave, uključujući i upravljanje ovlastima datoteka. Na Linuxu, POSIX kompatibilni ACL su: standardne ovlasti (*permissions*) i mogućnosti izvršavanja (engl. *capabilities*) {*POSIX 1003.1e*}), dok (prošireni) atributi nisu. POSIX ACL-ovi se obično pohranjuju u atributima datoteke na datotečnom sustavu.
- **NFSv4 ACL** – znatno su detaljniji od POSIX ACL-ova, a njihova definicija nalazi se unutar *Network File Systm* (**NFS**) standarda. NFSv4 ACL-ove podržavaju mnogi operativni sustavi slični Unixu (pr. Linux) i sve varijante Unixa. NFSv4 ACL-ovi se također pohranjuju u atributima datoteke na datotečnom sustavu.
- **Mrežni ACL** – na mrežnim uređajima (pr. usmjerivačima, vatrozidima i slično), ali i na Linuxu općenito, popis kontrole pristupa (ACL) pruža i definira pravila koja se primjenjuju na mrežne portove ili protokole, IP adrese i druge mrežne resurse, kako bi se ograničio pristup prema njima. Ova previla mogu se definirati i prema drugim mrežnim detaljima poput: protokola, smjera komunikacije (dolazni/odlazni), ponašanja (prema određenom obrascu) i slično.

Klasični *ACL* pod Linuxom pruža dodatni, fleksibilniji mehanizam dozvola za datotečni sustav odnosno za datoteke i direktorija na njemu. ACL je dizajniran kao dodatak klasičnim ovlastima za datoteke. ACL vam pri tome omogućuje davanje dozvola za bilo kojeg korisnika ili grupu bilo kojem resursu diska.
Svi potrebni programi za ACL dolaze u programskom paketu imena `acl`. Ako ga nemate možemo ga instalirati sa:

```
yum -y install acl
```

Linux **ACL** je specifičan za datotečni sustav odnosno ovisi o datotečnom sustavu.

**Linux ext2/3/4 datotečni sustavi**
Za konfiguriranje parametara koji se koriste prilikom stvaranja (formatiranja) datotečnih sustava *ext2*, *ext3* i *ext4* koristi se datoteka `/etc/mke2fs.conf`. Neki od parametara koji se mogu konfigurirati u ovoj datoteci uključuju veličinu bloka datotečnog sustava, omjer rezerviranog prostora, opcije montiranja i navedena opcija `acl` (sada standardna) i slično.
Nadalje, na Linux *extended* datotečnim sustavima (*ext2/3/4*), potrebno je prilikom montiranja datotečnog sustava navesti da se ACL omogućuje. To je u novije vrijeme standardna opcija (opcija `acl`), pa se više ne mora navoditi u datoteci `/etc/fstab`.



*Opcija `acl` prilikom montiranja particije uključuje podršku za **ACL**, dok opcija `noacl` isključuje **ACL**.*
*Zadane opcije montiranih datotečnih sustava nisu vidljive u datoteci `/proc/mounts`.*

**XFS, ZFS i drugi moderniji datotečni sustavi**
Svi noviji datotečni sustavi na Linuxu, poput **XFS**, **ZFS** i drugih standardno imaju uključeni ACL.

**Rad s ACL-om na Linuxu**
Normalne datoteke u datotečnom sustavu na poziciji ovlasti imaju prikazano samo 10 znakova, a posljednjih 9 koristi se za ovlasti datoteka. Međutim, kada se koriste ACL kontrole pristupa datotekama, pojavljuje se i 11. znak. Ovaj znak je plus (`+`) i označava korištenje ACL-a datoteke (ili direktorija). Prvo pogledajmo jednu (normalnu) datoteku s naredbom `ls`:

```
-rw-r--r--   1 root root  6120257 Mar  1 18:56 Log.txt
```

Sada se fokusirajmo samo na dio s ovlastima: `-rw-r--r--`
Dakle konkretni početni minus (`-`) označava da se radi o datoteci, a preostale oznake su ostatak ovlasti (`rw-r--r--`).
U gornjem primjeru vidljivo je da su korisnik `root` i grupa `root` vlasnici datoteke `Log.txt`.
Međutim, želimo li dodati i ovlasti za druge korisnike za istu datoteku kao dodatak na postojeće ovlasti, i to je moguće upotrebom ACL-a. Primjerice želimo li dodati korisniku `hrvoje` pravo zapisivanja i čitanja ove datoteke, moramo koristiti naredbu `setfacl` na sljedeći način:

```
setfacl -m "u:hrvoje:rw" Log.txt
```

Pogledajmo sada ovu datoteku ponovno:

```
ls -al Log.txt
-rw-rw-r--+ 1 root root  6120257 Mar  1 19:52 Log.txt
```

Vidimo da se sada u ovlastima pojavio znak plus na kraju ovlasti (`+`), ali se standardne ovlasti uopće nisu promijenile !
Sada pogledajmo ACL s novom naredbom `getfacl`:

```
getfacl Log.txt
# file: Log.txt
# owner: root
# group: root
user::rw-
user:hrvoje:rw-
group::r--
other::r--
```

Ovdje je vidljivo sljedeće:

- `# owner: root i user::rw-` - ovlasti za vlasnika (`root`) se uopće nisu promijenile.
- `user:hrvoje:rw-` - pojavila se ACL za korisnika `hrvoje` odnosno dodana su mu `rw` prava.

A zatim primjerice korisniku `pero` dodajmo samo pravo čitanja (uz prethodno dodatnog korisnika i uz vlasnika datoteke):

```
setfacl -m "u:pero:r" Log.txt
```

Pogledajmo sada nova prava ove datoteke (skratili smo ispis):

```
getfacl Log.txt
# file: Log.txt
user::rw-
user:hrvoje:rw-
user:pero:r--
group::r--
other::r--
```

Ovdje je vidljivo sljedeće (uz sve prethodno dodano što nećemo opisivati):

- `user:pero:r--` - pojavila se ACL za korisnika `pero` odnosno dodano mu je `r` pravo.

Pogledajmo kako kopirati ACL iz jedne datoteke u drugu (primjerice iz datoteke `Log.txt` u datoteku `Log2.txt`)

```
getfacl Log.txt | setfacl --set-file=- Log2.txt
```

Pogledajmo i druge opcije i parametre naredbe `setfacl`:

- `-b` – brisanje svih ACL ovlasti.
- `-k` – brisanje svih standardno zadanih ACL ovlasti.
- `-R` – rekurzivno postavljanje ACL na sve datoteke i direktorije.
- `-u:IME ili UID:ACL` – postavljanje ACL za korisnika (`IME`) s ovlastima `ACL` .
- `-g:IME ili GID:ACL` - postavljanje ACL za grupu (`IME`) s ovlastima `ACL` .
- `-o:ACL` - postavljanje ACL za sve ostale s ovlastima `ACL` .



Na većini datotečnih sustava koji podržavaju ACL-ove, ograničen je maksimalni dopušteni broj ACL unosa na neki razuman broj. Za **ext2/3/4** je to **32**, za **XFS** je to **25**, dok je za **ReiserFS** to **8191**. Upotreba ACL-a, kod nekih datotečnih sustava poput **ext2/3/4** i **ReiserFS** usporava pristup datotekama, zbog dodatnog vremena potrebnog za čitanje ACL-a, dok se u novijim datotečnim sustavima ACL direktno upisuje u ***inode*** strukturu datoteke (ili direktorija) pa stoga nema dodatnog kašnjenja prilikom čitanja ACL lista. Međutim prilikom kopiranja velikog broja datoteka, provjera i kopiranje ACL struktura i u naprednim datotečnim sustavima (iz *inode-a*) unosi određeno kašnjenje, uobičajeno čak i nekoliko postotaka **(1597)**.

**Izvori informacija: (108),(1596),(1597),(1598), `man setfacl`,`man getfacl`,`man 5 acl`,`man mke2fs.conf`.**

# 4.5. Datotečni sustav detaljnije

***Slijedi napredna cjelina!***
Za razumijevanje detalja o datotečnom sustavu (engl. *file system*), potrebno je razumjeti i neke detalje o datotekama.
U ovoj cjelini prvo ćemo raspravljati o detaljima vezanim za datoteke i direktorije.



Prije više detalja o datotečnom sustavu i datotekama, pogledajte i poglavlja o stvaranju **particije**, **formatiranju** particije te konačno i **montiranju** datotečnog sustava. Naime to su koraci koji prethode upotrebi datotečnog sustava.

**13.6.1. Kreiranje particija.**
**13.6.2. Formatiranje particije.**
**13.7. Kako se montiraju particije u direktorije i zašto.**

Vezano za datoteke, svaka datoteka se na razini datotečnog sustava sastoji od dva osnovna dijela:

- Dijela s podacima (*data* dio) i pripadajućim *metapodacima* (*ovlasti*: *vlasnik* i *grupa*, *datum* kreiranja i pristupanja, …).
- Dijela sa sâmim imenom datoteke.

Na slici 10. vidimo kako pojednostavljeno izgleda datoteka na Linux ***ext2, ext3*** ili ***ext4*** datotečnim sustavima ili na primjerice Unix ***UFS*** datotečnom sustavu.

**Slika 10. Datoteka na Linux ext2/3/4 datotečnom sustavu**




Na nižoj razini rada sustava, osnovni identifikator svake datoteke na bilo kojem Linux datotečnom sustavu nije sâmo ime datoteke, već njen pripadajući **`inode`** broj. Na osnovu unosa u tom *inode* broju, koji je specifičan za svaku pojedinu datoteku, operativni sustav dobiva razne meta podatke (*vrsta datoteke, vlasnik, grupa, ovlasti, datum kreiranja ,…*) od sâme datoteke.
Nadalje, sâmi podaci unutar datoteke se upisuju u blokove čije pozicije se zapisuju u *inode* unose za konkretnu datoteku, kako ćete vidjeti kasnije. Nadalje, svaki puta kada operativni sustav želi pristupiti nekoj datoteci, on prvo traži njen *inode* broj. Važno je razumjeti kako postoji i određena struktura koja se zapisuje u *inode* tablice, a različita je za direktorije (mape) i datoteke.

***Inode*** **struktura direktorija (mapa)**
Prvo ćemo pogledati kako izgleda jedna ***inode*** tablica za određeni direktorij (mapu):

| *Direktorij s Inode* brojem `4459486` | |
|---|---|
| **Sadržaj** | **inode broj** |
| `.` | `4459486` |
| `..` | `4456449` |
| `file.dd` | `4459826` |
| `output.svg` | `4459825` |

Važno je razumjeti da se unutar direktorija zapravo ne nalaze sâme datoteke već samo indeksi koji pokazuju na te datoteke! Ti indeksi se zovu *inode* brojevi.
U Linuxu, direktorij je samo posebna datoteka koja sadrži popis unosa (konkretno nazive datoteka) i njihove pripadajuće indekse (*inode broj*).



U prethodnom ispisu (tablici) vidljiva točka unutar direktorija (`.`) predstavlja sâm direktorij i ona nosi svoj *inode* broj. Ovaj *inode* broj direktorija pokazuje na blokove diska koji sadrže strukture direktorija.
Dok dvotočka (`..`) predstavlja direktorij u hijerarhiji iznad njega, također sa svojim *inode* brojem.
Preostale datoteke imaju, svaka svoj *inode* broj.

Pogledajmo i kako to izgleda u izlistanju datoteka i direktorija, konkretno za direktorij `/root/test/`, pomoću naredbe `ls`:

```
ls -ali

total 17944
4459486 drwxr-xr-x  2 root root     4096 Feb 17 19:54 .
4456449 drwx------ 12 root root     4096 Feb 20 18:58 ..
4459826 -rw-r--r--  1 root root 10485760 Feb 17 19:54 file.dd
4459825 -rw-r--r--  1 root root  2290093 Feb 17 17:20 output.svg
```

S lijeve strane su *inode* brojevi, a s krajnje desne: datoteke i/ili direktoriji vezani za točno određeni *inode* broj.
Dakle *inode* struktura (tablica) kod direktorija sadrži poveznicu: *inode broj* → *ime datoteke ili direktorija*.

Kako smo već spomenuli: `.` predstavlja trenutni direktorij, a `..` direktorij iznad njega.

***Inode* struktura datoteka(e)**
U odnosu na strukturu direktorija (mapa), *inode* struktura datoteke izgleda malo drugačije:

| **Inode broj: pr. 4459826** |
|---|
| Mode |
| Owner info |
| Size |
| Time stamps |
| **DIRECT BLOCKS** |
| **Indirect blocks** |
| **Double Indirect** |
| **Triple Indirect** |

Objasnit ćemo polja unutar ***inode*** tablice za bilo koju pojedinu datoteku:

- `Inode broj` – s pripadajućim brojem *inode-a* je identifikator konkretne datoteke.
- `Mode` – ovdje se zapisuje nekoliko informacija za konkretni *inode* odnosno sâmu datoteku. I to redom:
    - Koje su ovlasti (*permissions*) nad tom datotekom postavljene.
    - Vrsta *inode*a, koja može biti:
        - ***Datoteka*** - u ovom slučaju.
        - ***Direktorij (mapa)*** - u prijašnjem slučaju.
        - ***Blok uređaj*** ili drugi uređaj (tzv. *device* datoteke).
- `Owner info` – podaci o vlasniku i grupi za ovu datoteku (tko je vlasnik i kojoj primarnoj grupi pripada).
- `Size` – ovdje se zapisuje veličina datoteke u *bajtima.*
- `Time stamps` – ovdje se zapisuju vremena: kada je *inode*/datoteka kreirana, modificirana i sl.
- `DIRECT BLOCKS` – sadrži prvi blok podataka, koji je uvijek veličine bloka/klastera koji je definiran u trenutku kreiranja datotečnog sustava, odnosno formatiranja particije s datotečnim sustavom. Obično je to 4KB.

## Napredni dio priče s blokovima i klasterima



Prvo pogledajte datelje u poglavlju:
**13.2 Geometrija diskova** i to blokove odnosno klastere.

Vezano za pohranjivanje odnosno zapisivanje podataka u datoteke, važno je razumjeti kako se podaci zapisuju u najmanjim jedinicama za pohranu odnosno u takozvani blok podataka. Jedan blok je ovdje najmanja moguća jedinica za zapisivanje podataka, koji se u konačnici i zapisuju unutar ovog bloka. Ako je potrebno zapisati više podataka nego što stane u jedan blok (pr. standardnih 4KB), tada se alociraju dodatni blokovi, sve do određene granice. Naime sve ovisno o datotečnom sustavu, unutar jednog *inode* unosa, moguće je alocirati maksimalno definiran broj direktnih blokova (*DIRECT BLOCKS*), direktno u jednom *inode unosu*. Tako je za ***ext2*** linux datotečni sustav u *DIRECT BLOCKS* pokazivače (*pointere*) na blokove, moguće spremiti do 12 blokova; obično od 4KB - što je ukupno 48KB prostora za spremanje podataka.

Ako je potrebno još prostora, na **ext2** datotečnom sustavu i njegovoj *inode* strukturi, tada je moguće aktivirati 13-ti blok, odnosno takozvani ***Indirect blocks***, koji može koristiti do sljedeća 1024 bloka, odnosno blok pokazivača (*engl. block pointer)*.

S time je moguće pohraniti do 4MB, ako su blokovi od 4KB. Ako je potrebno još više, na ***ext2*** datotečnom sustavu i njegovoj *inode* strukturi, tada je moguće aktivirati još jedan blok pokazivač koji se zove ***Double Indirect blocks***, koji nakon ovih 48KB + 4MB može adresirati još 1024 ***Indirect blocks*** pokazivača što znači 1024 x 4MB = 4GB. I na kraju, ako niti to nije dovoljno, na ***ext2*** datotečnom sustavu i njegovoj *inode* strukturi, tada je moguće aktivirati još jedan blok pokazivač koji se zove ***Triple Indirect blocks***, koji može adresirati do 1024 x ***Double Indirect blocks***, što je 4GB x 1024 = 4TB.
Samim time ova veličina od 4TB je maksimalna veličina datoteke na Linux ***ext2*** datotečnom sustavu.

Za neke druge Linux/Unix datotečne sustave ove veličine su malo drugačije, ali princip je isti. Zanimljivo je i to da se klasičnim Linux datotečnim sustavima ***ext2, ext3*** ili ***ext4***, tijekom formatiranja particije s datotečnim sustavom definira i gornja granica *inode* brojeva. To znači i kako je maksimalan broj svih datoteka ograničen tijekom formatiranja particije.

Ovo ograničenje je poprilično veliko, ali moguće ga je dostići.



Kada se dogodi slučaj da smo iskoristili sve ***inode*** unose, više nije moguće kreirati niti jednu novu datoteku!



Ne zaboravite na to kako su i *file deskriptori* posebne datoteke, koji također imaju svoj *inode* broj.



Za detalje oko *file deskriptora* pogledajte poglavlje:
**4.5.4. Opisnici datoteke (*File deskriptori*).**

Pogledajte i ograničenje broja *inode*ova za neke datotečne sustave koje imamo pod Linuxom:

- Za **ext2**, **ext3** i **ext4** broj *inodeova* je 32 bitni broj, što znači maksimalan broj datoteka $2^{32}$ = 4 milijarde.
- Za ***XFS***, **BTFRS** i **ZFS** broj *inodeova* je 64 bitni broj, što znači maksimalan broj datoteka $2^{64}$ (izračunajte sami).

Iskorištenost *inode* brojeva možete vidjeti sa naredbom `df` (engl. *Disk free*) upotrebom prekidača `-i` na sljedeći način:

```
df -hi


Filesystem                                        Inodes IUsed IFree IUse% Mounted on
--------------------------------------------------------------------------------------
rootfs                                              8.3M  133K  8.2M    2% /
udev                                                249K   366  249K    1% /dev
tmpfs                                               251K   350  251K    1% /run
/dev/sda1                                           117M  227K  117M    1% /export/Share1
/dev/sdb1                                            59M  2.4K   59M    1% /export/Share2
```

Broj koliko je *inode*ova na kojem datotečnom sustavu iskorišteno, vidimo u stupcu `IUsed` te u postotku iskorištenja, u stupcu `IUse%`, a koliko ih je još slobodno, u stupcu `IFree`.

Konkretno u ispisu vidimo sljedeće (skratili smo opis):

- Za *root* datotečni sustav (`/`) vidimo kako imamo alocirano 8.3 milijuna (`8.3M`) **inode** unosa (u stupcu `Inodes`) od kojih je u upotrebi 133.000 (`133K`) što vidimo u stupcu `IUsed`. Od tog broja je slobodno odnosno još dostupno 8,2 milijuna (`8.2M`) što vidimo iz stupca `IFree`.
- Za datotečni sustav montiran u (`/export/Share1`) vidimo kako imamo alocirano 117 milijuna (`117M`) **inode** unosa (u stupcu `Inodes`) od kojih je u upotrebi 227.000 (`227K`) što vidimo u stupcu `IUsed`. Od tog broja je slobodno odnosno još dostupno 117 milijuna (`117M`) što vidimo iz stupca `IFree`.
- Za datotečni sustav montiran u (`/export/Share2`) vidimo kako imamo alocirano 59 milijuna (`59M`) **inode** unosa (u stupcu `Inodes`) od kojih je u upotrebi 2,4 (`2.4K`) što vidimo u stupcu `IUsed`. Od tog broja je slobodno odnosno još dostupno 59 milijuna (`59M`) što vidimo iz stupca `IFree`.
- …

Dakle ovdje konkretno vidimo statistike za svaki datotečni sustav sa svojom točkom montiranja zasebno.



Za detalje oko upotrebe naredbe **`df`**, pogledajte poglavlje:
**13.10.2. Naredba *df*.**



Za druge detalje o *inode* unosima pogledajte i poglavlje:
**9.4.3 Slijedeći koraci: Unix i network socketi** i to cjeline pred kraj navedenog poglavlja.

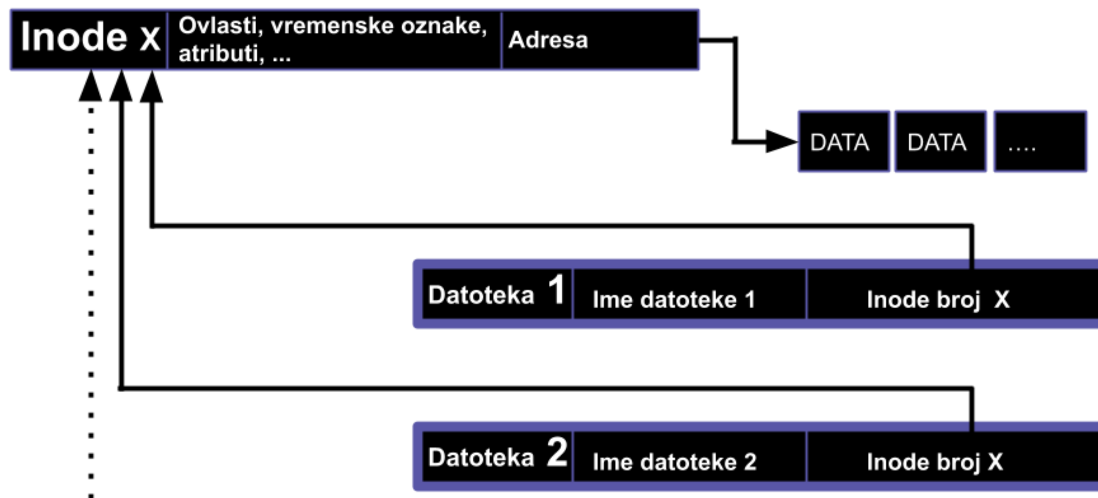
Pogledajte i sljedeća poglavlja:
**4.5.1. Hard Linkovi.**
**4.5.2. Soft linkovi.**

**Izvori informacija: (242),(243),(244),(245),(K-12), `man df`,`man 7 inode`.**

### 4.5.1. Hard Linkovi

Svaka datoteka odnosno njeno ime u Linux datotečnom sustavu je vezana za njen (pripadajući) *inode* broj. Stoga uvijek postoji jedna poveznica (*link*) između imena datoteke i njenog *inode-a*. Ova poveznica se zove *Hard link* odnosno *tvrda poveznica*. Dakle ova veza je u konačnici između naziva datoteke i stvarnih podataka pohranjenih u tu datoteku, u datotečnom sustavu. Podaci unutar svake datoteke se zapravo nalaze zapisani unutar ***inode*** *unosa*. Nadalje, moguće je kreirati i više *hard link* poveznica na istu datoteku. Takav *hard link* (**slika** 11.) nastaje tako da se u dijelu datoteke u kojem je upisan njen ***inode*** broj, koristi *inode* broj od neke druge datoteke, unutar kojeg su u konačnici i zapisani podaci te [druge] datoteke.
Kroz primjere ćete vidjeti i kako kreirati takozvanu *hard link* poveznicu na postojeću datoteku.

**Slika 11. Hard linkovi**




**Prednosti i mane Hard linkova**
Vezano za prednosti i mane *hard linkova*, važno je znati sljedeće:

- Mogu se kreirati samo unutar jedne particije jer ***inode*** unos postoji za svaku datoteku, ali unutar svake particije diska zasebno (konkretno unutar datotečnog sustava s kojim smo formatirali particiju).
- ***Inode*** broj izvorne datoteke i svih njenih hard linkova je isti (jer pokazuje na isti sadržaj datoteke).
- Brisanjem izvorne datoteke podaci ostaju sve dok postoji barem jedan *hard link*.
- Brisanjem *hard linkova* (poveznica) ne briše se sadržaj datoteke već samo broj *hard linkova*, sve dok postoji izvorna datoteka ili barem jedan *hard link*.
- Ne može se kreirati *hard link* na direktorij (mapu) kako bi se spriječila pojava rekurzivne petlje unutar datotečnog sustava. Naime tada bi bilo moguće kreirati *hard link* na roditeljski (vršni) direktorij, pa bi se ulaskom u njega ponovno ušlo u trenutni direktorij, što bi uzrokovalo da taj direktorij ima beskonačnu dubinu jer bi se ulaskom u njega ušlo u trenutni direktorij i tako dalje, beskonačno.
- Mogu se povezati (*linkati*) sâmo datoteke, ali ne i direktoriji (mape), uz posebne iznimke:
  - Kada se kreira svaki novi direktorij (primjerice naredbom `mkdir`) kreiraju se *hard linkovi* i to redom:
    - Jedan za roditeljski direktorij koji vidimo kao `..` i on ima isti *inode* broj kao roditelj (*parent*).
    - Jedan za trenutni direktorij koji vidimo kao `.` ali on ima novi *inode* broj.
- Ovlasti izvorne datoteke su iste na *hard link* datotekama.
  - Promjenom ovlasti na hard link datoteci mijenjaju se ovlasti na izvornoj datoteci i obratno.
- Teže ih je primijetiti i pratiti jer se mora gledati broj linkova i *inode* broj.

Kreirajmo datoteku imena `text.txt` pomoću naredbe `echo`:

```
echo "probni text" > text.txt
```

Napravimo takozvani *hard link* na nju, pomoću naredbe `ln` (engl. *Links*), a pri tome će se nova datoteka zvati `hard.link`:

```
ln text.txt hard.link
```

Provjerimo kako to sada izgleda, U primjeru nam prekidač (`-i`) naredbe `ls` prikazuje *Inode* brojeve:

```
ls -ali

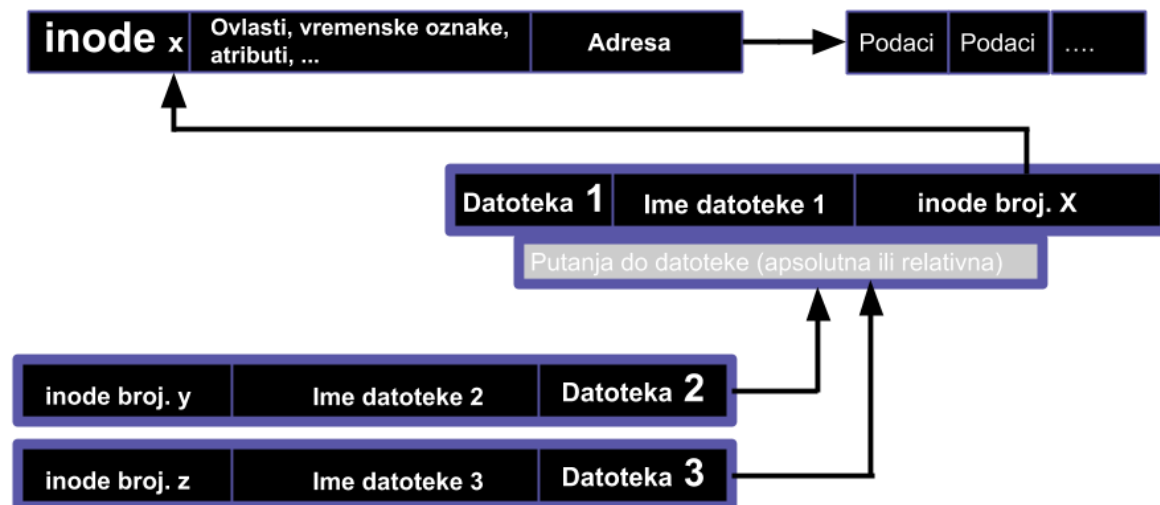
5636110 drwxr-xr-x  2 root root 4096 Nov 13 09:47 .
5636097 dr-xr-x---. 4 root root 4096 Nov 13 09:28 ..
5636111 -rw-r--r--  2 root root 5    Nov 13 21:29 text.txt
5636111 -rw-r--r--  2 root root 5    Nov 13 21:29 hard.link
```

Pogledajte gore: ***Inode*** brojeve (prvi stupac) od datoteka: `text.txt` i `hard.link`.
Dakle datoteka `text.txt` ima *inode* broj `5636111`, a datoteka `hard.link` ima isti *inode* broj jer se radi o *hard linkovima* odnosno zbog toga jer svi oni pokazuju na istu lokaciju (*inode* broj) na kojoj se nalazi sadržaj sâme datoteke. Obratite pažnju na treći stupac koji označava broj *hard linkova* na izvornu datoteku, koji ovdje ima vrijednost `2`. Naime pošto svaka datoteka ima barem jednu poveznicu: ime datoteke na svoj *inode* broj stoga, ako datoteka nema dodatnih *hard link* poveznica, ovdje ćemo uvijek imati vrijednost: `1`. Stoga ako imamo jedan dodatno kreiran *hard link* na nju, vrijednost će biti `2` i tako dalje.
**Izvori informacija: (K-1),(K-12), `man ls`, `man ln`.**

## 4.5.2. Soft linkovi

*Soft Linkovi* se zapravo referenciraju na već kreiranu datoteku ili direktorij s apsolutnom ili relativnom putanjom.
Dakle *soft link* datoteka zapravo pokazuje na ***inode*** broj od izvorne datoteke.
To znači kako se *soft link* direktno veže na sadržaj izvorne datoteke, pošto se njen sadržaj odnosno podaci nalaze u ***inode*** dijelu.
Logička shema *soft link* datoteke je vidljiva na slici 12.

**Slika 12. Pogled na *soft linkove***




**Prednosti i mane *Soft linkova***

- Fleksibilniji su jer pokazuju na putanju (*path*) do datoteke ili direktorija.
- Mogu se koristiti i između više datotečnih sustava (*filesystem*) odnosno diskova ili particija.

Slijedi nastavak na primjere od *hard linkova*. Pomoću naredbe `ln` napravimo *soft link* na izvorišnu datoteku `text.txt`, pri čemu će se nova datoteka zvati `soft.link`

```
ln -s text.txt soft.link
```

Provjerimo kako to sada izgleda

```
ls -ali

5636110 drwxr-xr-x  2 root root 4096 Nov 13 09:47 .
5636097 dr-xr-x---. 4 root root 4096 Nov 13 09:28 ..
5636111 -rw-r--r--  2 root root 5    Nov 13 21:29 text.txt
5636111 -rw-r--r--  2 root root 5    Nov 13 21:29 hard.link
5636112 lrwxrwxrwx  1 root root 8    Nov 13 09:34 soft.link -> text.txt
```

Uočimo razliku u ***inode*** broju *soft linka*: datoteka imena: `soft.link` ima potpuno novi ***inode*** broj (`5636112`), a ne isti kao izvorišna datoteka (na koju se referencira), kao što bi bilo u slučaju upotrebe *hard linka*.

## Ponašanje soft linkova

- ***Pokazivanje na nepostojeće datoteke ili brisanje izvorne datoteke***– ako izvorna datoteka ili direktorij koji je povezan *soft linkom* bude izbrisan, *soft link* postaje "prekinut" ili "mrtav" link (eng. *dangling link*). Kada pokušate pristupiti tom linku, dobivate poruku o pogrešci jer ne postoji stvarni sadržaj na koji pokazuje. Dakle brisanjem originala tj. izvorne datoteke (*source*), *soft link* ostaje, premda pokazuje na nepostojeću datoteku ili direktorij koji tada više ne postoji pa je efektivno neupotrebljiv
- ***Premještanje izvorne datoteke*** – ako se izvorna datoteka premjesti, *soft link* i dalje pokazuje na staru putanju, što može dovesti do prekinutih poveznica.
- ***Izmjene u soft linku*** – promjena sadržaja preko *soft linka* utječe na originalnu datoteku. No, ako izbrišete *soft link*, originalna datoteka ostaje netaknuta.

## Razlika između soft i hard linkova

- *Soft link* je pokazivač na drugu datoteku ili direktorij, dok je *hard link* direktna referenca na *inode* broj (strukturu koja pohranjuje podatke o datoteci u datotečnom sustavu).
- *Soft link* može pokazivati na datoteku ili direktorij, čak i na različitim datotečnim sustavima.
- *Hard link* mora pokazivati na datoteku unutar istog datotečnog sustava i ne može se primijeniti na direktorije.

**Izvori informacija: (K-1),(K-12), `man ls`, `man ln`, `man 7 symlink`.**

## 4.5.3. Oznake vremena (*timestamps*)

Tijekom svakog kreiranja direktorija (mape) ili datoteke, kao i pristupanjem ili mijenjanjem istih, zapisuje se vrijeme kada su se navedene operacije radile. Naime važno je razumjeti, da se za svaku mapu i datoteku zapisuju oznake vremena, ovisno o datotečnom sustavu na kojem se nalaze.

U pravilu se zapisuju sljedeće oznake vremena (engl. *Timestamps*):

- `access` – govori nam kada je pristupano datoteci ili mapi. Noviji datotečni sustavi ne moraju stalno mijenjati ovo vrijeme zbog optimizacije performansi, već mogu koristiti „*relatime*“ opciju kod koje se zapisuje (mijenja) vremenska oznaka samo, ako je ovo vrijeme starije od *modify* vremena.
- `modify` – govori nam kada je kreiran ili je mijenjan sadržaj mape ili datoteke.
- `change` – govori kada je promijenjen njen status (status ***inode*** unosa) ili njene ovlasti (engl. *Permissions & modes*).

**Slijede primjeri**

Nastavljamo s već kreiranim datotekama. Provjerimo kako izgledaju njihove oznake vremena, s izlistanjem:

```
ls -al

drwxr-xr-x  2 root root 4096 Nov 13 19:03 .
dr-xr-x---. 4 root root 4096 Nov 13 19:28 ..
-rw-r--r--  2 root root 5    Nov 13 19:29 hard.link
lrwxrwxrwx  1 root root 8    Nov 13 19:34 soft.link -> text.txt
-rw-r--r--  2 root root 5    Nov 13 19:29 text.txt
```

Pogledajmo **vremenske** unose odnosno oznake malo detaljnije, pomoću naredbe `stat` za datoteku: `text.txt`

```
stat text.txt

File: `text.txt';
Size: 5 ; Blocks: 8 ; IO Block: 4096 ; regular file
Device: 801h/2049d ; Inode: 5636111 ; Links: 2
Access: (0644/-rw-r--r--);Uid: ( 0/root) ; Gid: ( 0/root)
Access: 2013-11-13 19:47:34.519854892  +0000
Modify: 2013-11-13 19:29:17.359728861  +0000
Change: 2013-11-13 19:20:19.372604340  +0000
```

Vidimo navedena vremena: `Access/Modify/Change`, a koja su osjetljiva na vremensku zonu u kojoj se računalo nalazi. Lokalni vremenski odmak vremenske zone se vidi u zadnjem stupcu svakog retka; konkretno je to kod nas oznaka: `+0000`. Ovaj vremenski odmak (`+` ili `-`) odnosi se na odmak našeg lokalnog vremena od ***UTC*** vremenske zone.

Osim vremenskih oznaka, naredba `stat` nam prikazuje i ***inode*** broj datoteke (u primjeru: *`Inode: 5636111`*) te broj *hard linkova* na tu datoteku (u primjeru: *`Links: 2`*). Pri tome ova dvojka znači da ova datoteka ima *2 hard linka*.

Kako promijeniti vremena pristupa ili zadnje promijene datoteke pomoću naredbe `touch`:

- Promijenit ćemo: `access` (`-a`) i `modify` (`-m`) vrijeme u: godina: 2013, mjesec: 11, dan: 11, sat: 12, minuta: 01 i sekunda: 10.

```
touch -a -m -t 201311111201.10 text.txt
```

Provjerimo sada **vremena** odnosno vremenske oznake ove datoteke:

```
stat text.txt

File: `text.txt'
Size: 5 Blocks: 8 IO Block: 4096 regular file Device: 801h/2049d
Inode: 5636111 Links: 2 Access: (0644/-rw-r--r--)
Uid: ( 0/ root) Gid: ( 0/ root)

Access: 2013-11-11 12:01:10.000000000 +0000
Modify: 2013-11-11 12:01:10.000000000 +0000
Change: 2013-11-13 19:20:19.372604340 +0000
```

Vidimo da smo promijenili ono što smo i željeli. Kako još provjeriti vremena pristupanja/promjena ili pristupa datotekama?

Provjerimo vrijeme zadnje promjene datoteke u sekundama [*oznaka je za UNIX vrijeme (u sekundama od 1.1.1970)* ].

```
stat -c %Y file.sh

1458573410
```

Potom provjerimo isto vrijeme, ali s naredbom `date` na sljedeći način:

```
date -r file.sh +%s

1458573410
```

**Slijede napredni primjeri**
Pogledajmo i detaljniji ispis za našu datoteku, na razini datotečnog sustava (pr. ***ext3*** ili ***ext4***).

```
ls -ali file.sh

21062 -rw-r--r-- 1 root root 10 Mar 22 19:13 file.sh
```

Nas zanima ***inode*** broj od naše datoteke (u ovom slučaju je to `21062`). Sada ćemo izvući malo detaljniju statistiku na razini datotečnog sustava. Naša datoteka se nalazi u direktoriju `/root/test`.

Vršni direktorij `/` je *montiran* sa particije `/dev/sda1` i znamo da se koristi ***ext4*** datotečni sustav. Pogledajmo što će nam reći naredba `mount`:

```
mount

/dev/sda1 on / type ext4 (rw)
```

Sada ćemo u pretragu upisati naš INODE broj (`21062`) i particiju na kojoj se nalazi (sjetimo sa da ***INODE*** ovisi o particiji)

```
debugfs -R 'stat <21062>' /dev/sda1
debugfs 1.41.12 (17-May-2010)
Inode: 21062   Type: regular    Mode:  0644   Flags: 0x80000
Generation: 2152679827    Version: 0x00000000:00000001   User:     0
Group:     0   Size: 10     File ACL: 0    Directory ACL: 0 Links: 1   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x56f136ff:5705f0b8  -- Tue Mar 22 19:13:51 2016
 atime: 0x56f13681:db1b39b4  -- Tue Mar 22 19:11:45 2016
 mtime: 0x56f136ff:5705f0b8  -- Tue Mar 22 19:13:51 2016
 crtime: 0x56f1359a:bd10ae80 -- Tue Mar 22 18:07:54 2016
```

Ovdje vidimo još jedno polje, a to je `crtime`. Naime u slučaju ***ext3*** i ***ext4*** datotečnih sustava u ovo polje (`crtime`) se upisuje vrijeme kreiranja datoteke.

Ostala polja su standardna za sve Unix/Linux datotečne sustave: `ctime`, `atime` i `mtime`.

Naredba `debugfs` se koristi za *debugiranje* ***ext2***, ***ext3*** i ***ext4*** datotečnih sustava. Osim toga s njom je moguće i mijenjati određene parametre datotečnog sustava.

Za ***XFS*** datotečni sustav, za potrebe *debugiranja* koristi se naredba `xfs_io`.

U nastavku pogledajmo statistike za ***XFS*** datotečni sustav gdje imamo kreiranu datoteku `/root/test.txt`.
Provjerimo koje sve vremenske oznake datotečni sustav **XFS** pohranjuje (skratili smo ispis):

```
xfs_io
xfs_io> open /root/test.txt
xfs_io> stat -v


fd.path = "/root/test.txt"
fd.flags = non-sync,non-direct,read-write
stat.ino = 805348398
stat.type = regular file
stat.atime = Wed Mar 23 21:40:03 2016
stat.mtime = Wed Mar 23 19:48:11 2016
stat.ctime = Wed Mar 23 19:48:11 2016
fsxattr.xflags = 0x0 []
```

Vidljivo je da se pohranjuju: `atime`, `mtime` i `ctime`, međutim ne zapisuje se vrijeme kreiranja datoteke kako kod ***ext3/4***.



**Važno je razumjeti da pohranjivanje vremenskih oznaka ovisi o datotečnom sustavu.**
**Naime neki datotečni sustavi spremaju određene oznake, a drugi ne!**

**Izvor informacija: (K-12), `man stat`,`man debugfs`,`man date`,`man xfs_io`,`man mount`.**

## 4.5.4. Opisnici datoteke (*File deskriptori)*

***Slijedi napredna cjelina!***
Linux procesi (pokrenuti programi) komuniciraju s datotekama, *soketima* i drugim procesima putem takozvanih *file deskriptora*. Svaki proces u Linuxu ima svoj skup *file deskriptora*, koji pokazuju na otvorene datoteke, ***sockete*** ili druge resurse. Primjerice kada se otvori datoteka ili uspostavi veza putem ***socketa***, kernel dodjeljuje *file deskriptor* tom resursu, a procesu se potom pristupa kroz taj *file deskriptor*. Dakle, file deskriptori su način na koji procesi u Linuxu pristupaju datotekama, *socketima* i drugim resursima.

*File deskriptori* su znani i kao opisnici datoteke i oni su posebne datoteke, odnosno datoteke koje se nalaze u posebnom dijelu datotečnog sustava. Nadalje, oni se prema većini osnovnih parametara, gledano s razine datotečnog sustava, uopće ne razlikuju od drugih datoteka, jer i oni imaju: *ovlasti*, *datum kreiranja* i *inode* broj u datotečnom sustavu. Pohranjuju se u vršnom sloju datotečnog sustava koji se naziva virtualni datotečni sustav odnosno ***VFS*** *(Virtual File system)***,** koji se nalazi se u RAM memoriji.

*File deskriptora* postoje osnovne tri vrste, uz još tri rezervirane kategorije. Rezervirane ketagorije *file deskriptora* su:

- **`Stdin`** – koji predstavlja standardni ulaz, a njegov *file deskriptor* ima oznaku `0`.
- **`Stdout`** – koji predstavlja standardni izlaz, a njegov *file deskriptor* ima oznaku `1`.
- **`Stderror`** – koji predstavlja standardnu grešku, a njegov *file deskriptor* ima oznaku `2`.

Primjerice svaki puta kada otvorite neku datoteku operativni sustav kreira vezu (*stream*) prema toj datoteci. *File deskriptor* zapravo predstavlja vezu prema toj otvorenoj datoteci. U slučaju rada u terminalu, postoje minimalno gore navedene tri kategorije posebnih *file deskriptora*. Ova tri *file deskriptora*: `stdin`, `stdout` i `stderror` su obično spojena na terminal, a ne na datoteke ili druge poveznice, kao ostali *file deskriptori*.

To znači da kada nešto pišete u terminalu, sve što pišete ulazi na `stdin(FD0)`, a ono što vidite na ekranu odlazi zapravo na `stdout(FD1)`.

Zamislimo otvorene datoteke poput: `datoteka-1`, `datoteka-2` i `datoteka-3`. *File deskriptori* su indeksi odnosno pokazivači na te datoteke, koji se čuvaju u posebnoj tablici.

Za svaki pokrenuti program (*proces*), održava se po jedna ovakva tablica s otvorenim datotekama, poput tablice dolje:

| Index | **1** | **2** | **3** |
|---|---|---|---|
| Datoteka | `datoteka-1` | `datoteka-2` | `datoteka-3` |

Što se bilo kojeg pokrenutog programa, odnosno Linux procesa tiče, njega ne zanima (niti je svjestan) kako zapisivati podatke u datoteke, kako im uopće pristupiti i slično, o tome se brine sâm kernel, koji je svjestan toga na kojem datotečnom sustavu se nalaze datoteke, kao i drugih detalja.

Programi/procesi zapravo za rad i komunikaciju koriste *file deskriptore*, pa se tako može dogoditi da se otvore: po jedan *file deskriptor* na istu datoteku za čitanje, jedan za zapisivanje, i tako dalje, sve ovisno o namjeni.
Slična situacija je i kod mrežne komunikacije, u kojoj se za komunikaciju, otvaraju posebni *file deskriptori*, ovisno o vrsti komunikacije; primjerice: ***socket****(poglavlje:9.4.3.1)* ili ***pipe****(pogl.9.4.1)*.
Dakle poveznica između datoteka i *file deskriptora* nije **1:1**.

Prije navedena *file deskriptor* tablica zapravo izgleda ovako nekako:

| Index | **41** | **42** | **43** | **44** |
|---|---|---|---|---|
| Datoteka | `datoteka-1` | `datoteka-1` | `datoteka-2` | `datoteka-3` |
| Način pristupa | Čitanje (**`r`**) | Pisanje (**`w`**) | Dodavanje na kraj datoteke (***append***) | Čitanje (**`r`**) i pisanje (**`w`**) |

Tijekom pokretanja bilo kojeg programa (procesa), prvo se otvaraju prva tri osnovna definirana *file deskriptora*, koji moraju imati svoje pripadajuće brojeve (`0`, `1` i `2`). Potom se otvaraju i drugi po redu *file deskriptori*, kako se program učitava u RAM memoriju. Drugim riječima, programi odnosno procesi u Linuxu, za svako dohvaćanje vanjskih resursa, kao i za međusobnu i mrežnu komunikaciju, koriste *file deskriptore*, kako ćete vidjeti u poglavlju: **9.4. Komunikacija između procesa**.



Svakim otvaranjem datoteke; primjerice: tekstualne, binarne/izvršne, skripte, programskih biblioteka i slično, kao i svake mrežne veze, kreira se barem jedan *file deskriptor* koji predstavlja vezu prema toj datoteci ili mrežnoj vezi. Stoga se na sustavim s velikim brojem mrežnih veza ili velikim brojem otvorenih datoteka otvara još veći broj *file deskriptora*.
**Ne zaboravimo da se file deskriptori kreiraju u RAM memoriji!**

**Izvori informacija: (K-12),(673), `man bash`.**

## 4.5.5. File *deskriptori* detaljnije

***Slijedi napredno poglavlje (4.5.5.x).***
Iz sigurnosnih i razloga iskorištavanja resursa sustava, ograničen je maksimalan broj *file deskriptora* koji se može definirati za cijeli sustav, ali i svakog pojedinog korisnika. U svakodnevnom radu, posebno u radu servisa (zbog njihovih ograničenja), ponekad nam je potrebno saznati koliko *file deskriptora* su oni zauzeli.
Ne zaboravimo da u *Linuxu* svaki uređaj predstavlja neka (posebna) datoteka. Tako da svaki program pristupa prilično velikom broju datoteka, poput:

- Raznih biblioteka (engl. *library*) i drugih datoteka potrebnih za rad sâmog programa.
- Njegovih *log* datoteka u koje se zapisuju razni statusi rada i pristupanja programu i slično.
- Posebnih datoteka koje predstavljaju hardver ili neke funkcionalnosti sustava (tzv. `/dev/` datoteke).
- ***Unix*** ili ***network socket*** datoteka te drugih programa i njihovih datoteka.

Prvo što moramo saznati je koji proces ID (***PID** broj*) ima *servis* ili *program* koji pratimo. To možemo napraviti na nekoliko načina. Recimo da se radi o procesu imena `sshd` (to je SSH poslužitelj). Pogledajmo kako pronaći njegov ***PID*** broj.

**1. Način**: Upotrebom naredbe za izlistavanje aktivnih Linux procesa (*pokrenutih programa i servisa*) to jest `ps` na sljedeći način:

```
ps -aux | grep sshd

root 18670 0.0 0.0 66612 1228 ? Ss Feb22 0:00 /usr/sbin/sshd
```

***PID*** broj (ovog) programa/procesa se nalazi u drugom stupcu. U našem slučaju je to ***PID*** broj `18670`.

**2. Način:** Upotrebom naredbe `pidof` koja nam na osnovu imena procesa vraća ***PID*** broj tog (konkretnog) procesa:

```
pidof sshd

18670
```

Dakle imamo ***PID*** broj: `18670`. Sada moramo saznati koliko *file deskriptora* ima otvoren naš proces s ***PID*** brojem `18670`.

Za prebrojavanje *file deskriptora* (između njenih drugih mogućnosti), za određeni ***PID*** broj, zadužena je naredba `lsof`.
Listu svih otvorenih *file deskriptora* za naš program (po ***PID*** broju `18670`) ćemo dobiti ovako:

```
lsof -a -p 18670
```

… i to datoteku po datoteku (kojih će obično biti otvoren popriličan broj).

Pošto nam treba samo broj otvorenih *file deskriptora (odnosno veza prema otvorenim datotekama)*, koristit ćemo naredbu „*word count*" odnosno `wc -l`, koja će nam ih sâmo prebrojiti. Stoga ćemo pokrenuti sljedeći nîz naredbi:

```
lsof -a -p 18670 | wc -l

39
```

I konačno smo dobili rezultat: `39` otvorenih *file deskriptora,* kolikoj je otvorio naš program odnosno servis `sshd`.

Želimo li vidjeti koliko *file deskriptora* je u upotrebi na cijelom sustavu, to ćemo postići na sličan način:

```
lsof | wc -l

334
```

Dakle cijeli Linux (trenutno) na kojem smo pokrenuli ovu naredbu koristi 334 *file deskriptora*(*radi se o računalu koje baš i nije u upotrebi*).

Pogledajmo *file deskriptore* za `stdin`(0) `stdout`(1) i `stderr`(2). I to za recimo naš *Web* poslužitelj *Apache*.
Pronašli smo ***PID*** broj našeg *Apache Web* poslužitelja, koji je u ovom slučaju: `29377`.

Pogledajmo koje sve *file deskriptore* on ima otvorene (ispis je skraćen za potrebe prikaza):

```
lsof -p 29377

COMMAND   PID   USER    FD  TYPE   DEVICE  SIZE/OFF      NODE  NAME
httpd    29377 apache   0r  CHR     1,3       0t0      23086  /dev/null
httpd    29377 apache   1w  CHR     1,3       0t0      23086  /dev/null
httpd    29377 apache   2w  REG    0,52      2104   27263802  /var/log/httpd/error_log
```

U stupcu `FD` nalaze se *file deskriptori* u obliku: ***Broj****Stanje*. Dakle pod stupcem `FD` tražimo *file deskriptore:* `0`, `1` i `2`, pa imamo:

- `Stdin` odnosno *FD* `0` je u stanju čitanja (prema oznaci "`r`" (*READ*)), iz datoteke (stupac `NAME`): `/dev/null` što opet znači kako trenutno ne prima nikakav ulaz podataka.
- `Stdout` odnosno *FD* `1` je u stanju zapisivanja (prema oznaci "`w`" (*WRITE*)), prema datoteci (stupac `NAME`): `/dev/null`. To znači kako ništa trenutno ne šalje na "Ekran/MONITOR" jer sve završava u "*NULL*" datoteci.
- `Stderr` odnosno *FD* `2` je u stanju zapisivanja (prema oznaci "`w`" (*WRITE*)), prema datoteci (stupac `NAME`): `/var/log/httpd/error_log`. To znači kako se sve greške zapisuju odnosno *logiraju* u tu datoteku.

Isto smo još jednostavnije mogli postići izlistanjem datoteka u direktoriju unutar `/proc/`**`PID`**`/fd/` direktorija.
Pri čemu je **`PID`** zapravo ***PID*** broj našeg procesa (u ovom slučaju `29377`).

Na primjeru sa slike 13 smo pokrenuli naredbu:

```
ls -al /proc/29377/fd/
```

**Slika 13. File deskriptori (***veze prema otvorenim datotekama***) za pojedini proces (pokrenuti program)**

```
[root@OSOS-devel ~]# ls -al /proc/29377/fd/
total 0
dr-x------ 2 root   root    0 Jan 24 19:16 .
dr-xr-xr-x 7 apache apache  0 Jan 24 19:01 ..
lr-x------ 1 root   root   64 Jan 24 19:16 0 -> /dev/null
l-wx------ 1 root   root   64 Jan 24 19:16 1 -> /dev/null
l-wx------ 1 root   root   64 Jan 24 19:16 10 -> /var/log/httpd/ssl_request_log
lr-x------ 1 root   root   64 Jan 24 19:16 11 -> /dev/urandom
l-wx------ 1 root   root   64 Jan 24 19:16 12 -> /var/log/httpd/mod_jk.log
lrwx------ 1 root   root   64 Jan 24 19:16 13 -> /var/log/httpd/jk-runtime-status.1056
lrwx------ 1 root   root   64 Jan 24 19:16 14 -> /var/log/httpd/jk-runtime-status.1056.lock
lr-x------ 1 root   root   64 Jan 24 19:16 15 -> /usr/share/GeoIP/GeoLiteCountry.dat
lr-x------ 1 root   root   64 Jan 24 19:16 16 -> /usr/share/GeoIP/GeoLiteASNum.dat
lr-x------ 1 root   root   64 Jan 24 19:16 17 -> /usr/share/GeoIP/GeoLiteCity.dat
lrwx------ 1 root   root   64 Jan 24 19:16 18 -> [eventpoll]
l-wx------ 1 root   root   64 Jan 24 19:16 2 -> /var/log/httpd/error_log
lr-x------ 1 root   root   64 Jan 24 19:37 26 -> /etc/pki/nssdb/cert9.db
lr-x------ 1 root   root   64 Jan 24 19:37 27 -> /etc/pki/nssdb/key4.db
lrwx------ 1 root   root   64 Jan 24 19:16 3 -> socket:[21313227]
lrwx------ 1 root   root   64 Jan 24 19:16 4 -> socket:[21313229]
lr-x------ 1 root   root   64 Jan 24 19:16 5 -> pipe:[267079781]
l-wx------ 1 root   root   64 Jan 24 19:16 6 -> pipe:[267079781]
l-wx------ 1 root   root   64 Jan 24 19:16 7 -> /var/log/httpd/ssl_error_log
l-wx------ 1 root   root   64 Jan 24 19:16 8 -> /var/log/httpd/access_log
l-wx------ 1 root   root   64 Jan 24 19:16 9 -> /var/log/httpd/ssl_access_log
```

Opis: Uokvireni su `0`,`1` i `2` koji predstavljaju redom: `stdin`(standardni ulaz) (`0`), `stdout`(standardni izlaz) (`1`) i `stderr`(standardnu grešku) (`2`).



Za detalje oko naredbe `lsof` pogledajte poglavlje:
**25.7.6 Naredba list open files (*lsof*)**.

Osim osnovnih *file deskriptora* koje smo naveli, postoje i posebne vrste *file deskriptora*:

- `pipe` – odnosno *FIFO*, a oni se koriste za komunikaciju između procesa/programa. U stupcu `TYPE`, ispisa naredbe `lsof`, obično ima oznaku: **`FIFO`**.
- `unix socket` – koriste se za komunikaciju između procesa/programa. U stupcu `TYPE`, ispisa naredbe `lsof`, obično ima oznaku: **`unix`**.
- `network socket` – koriste se za mrežnu komunikaciju između procesa/programa. U stupcu `TYPE`, ispisa naredbe `lsof`, obično ima oznaku: **`IPv4`** ili **`IPv6`** ovisno radi li se o IP v.4. ili IP v.6. protokolu.



Za ***pipe***, ***unix*** *socket* i ***network*** *socket* vrste *file deskriptora*, pogledajte veće poglavlje:
**9.4 Komunikacija između procesa**, gdje ćete vidjeti malo detaljnije kako oni rade i čemu točno služe.

**Napredni primjeri**

Kako smo već govorili: *file deskriptor 0* predstavlja standardni ulaz, a *file deskriptor 1* standardni izlaz. To znači kako će se slanjem podataka (ili u ovom primjeru teksta) na *file deskriptor 1* za konkretni proces (program) taj tekst ispisati na standardni izlaz (terminal/ekran/monitor). Pronađimo ***PID*** broj naše trenutne ljuske. To možemo napraviti pomoću naredbe: `ps -ef`, te pretraživanjem ***PID*** broja naše ljuske ili sa sljedećom naredbom koja će nam dati ***PID*** broj trenutne ljuske (*shella)*:

```
echo $$
```

U svakom slučaju, saznali smo da je naš *shell/ljusku* (*bash*), u kojem radimo, pokrenut pod ***PID*** brojem `1180`. To znači da bi slanjem nekog teksta direktno na *file deskriptor* našeg *shella* zadužen za standardni izlaz (***1***), taj tekst bio poslao na naš ekran. Dakle ako napravimo preusmjeravanje odnosno redirekciju teksta „`Test`" u željeni *file deskriptor* (***1***) i to ovako:

```
echo "Test" > /proc/1180/fd/1
```

Dobit ćemo poruku u našoj ljusci (*shellu)* odnosno na našem terminalu/ekranu (naredbenom retku):

```
Test
```

To znači da istu stvar možemo napraviti i na nekom drugom *shellu* ili programu. Također na isti način možemo pokrenuti i redirekciju ispisivana bilo koje druge naredbe, u neku drugu ljusku (*shell)*, poput ispisivanja naredbe: `ls -al`:

```
ls -al  > /proc/1180/fd/1
```

Dodatno, postoji i naredba, s kojom možemo napraviti trajniju promjenu našeg standardnog izlaza (ili ulaza) prema negdje drugdje. Ova naredba je `exec`, a može se koristiti i za druge namjene (pr. zamjene *shella i* druga preusmjeravanja).
Zamislimo slučaj u kojem imamo dva korisnika spojena preko ***SSH*** protokola na udaljeni poslužitelj.

Prvi ima *bash shell (ljusku)* u kojoj radi, s ***PID*** brojem `1383`, a drugi smo mi, koji imamo našu radnu ljusku (*shell)* s ***PID*** brojem `1180`. Mi možemo trajno preusmjeriti svoj standardni izlaz *bash ljuske* (ono što vidimo na ekranu) na standardni izlaz druge *bash ljuske* drugog korisnika i to samo, ako smo *root* korisnik odnosno korisnik s dovoljno visokim pravima.

Iz naše *bash ljuske* pokrenimo naredbu `exec` na sljedeći način:

```
exec 1 > /proc/1383/fd/1
```

Od tog trena, sve što upišemo u našoj *bash* ljusci, će se ispisivati na udaljenoj *bash ljusci* od drugog korisnika (***PID*** `1383`). Ako primjerice pokrenemo naredbu: `ls -al` njen ispis neće više biti vidljiv u našoj *bash ljusci*, već u onoj od drugog korisnika odnosno one *bash* ljuske s ***PID*** brojem:`1383`.

Ako ipak sve želimo vratiti na terminal koji se prije koristio, moramo vratiti preusmjeravanje koje je bilo i prije, dakle: *1* → */dev/pts/0*, pa to i napravimo:

```
exec 1 > /dev/pts/0
```

U bilo kojem trenutku možemo vidjeti koji terminal koristimo u našoj ljusci (*shellu)*, s naredbom `tty` na sljedeći način:

```
tty
```

**Izvori informacija: (222),(673),(K-12), `man lsof`,`man 1p exec`,`man tty`.**

### 4.5.5.1. File deskriptori i naredba *lsof*

Pomoću naredbe `lsof` možemo provjeravati i kojim datotekama unutar željenih direktorija, pristupaju ili ih drže otvorenima, određeni programi ili servisi. Pogledajmo koji sve programi pristupaju datotekama unutar direktorija `/var/log` ne uključujući njene pôddirektorije, već sâmo datoteke koje su u njemu odnosno u ovom direktoriju (skratili smo ispis):

```
lsof +d /var/log/

COMMAND    PID  USER    FD   TYPE DEVICE SIZE/OFF     NODE  NAME
rsyslogd  734  root     2w   REG    0,54    38445 27264337  /var/log/messages
rsyslogd  734  root     4w   REG    0,54    75986 27263807  /var/log/cron
rsyslogd  734  root     5w   REG    0,54   165122 27264345  /var/log/secure
rsyslogd  734  root     6w   REG    0,54     5383 27264313  /var/log/maillog
mysqld   1139 mysql     1w   REG    0,54    70047 27264189  /var/log/mysqld.log
mysqld   1139 mysql     2w   REG    0,54    70047 27264189  /var/log/mysqld.log
```

S krajnje lijeve strane (pod stupcem `COMMAND`) vidimo koji program pristupa, a na krajnjoj desnoj strani (`NAME`) vidimo kojoj datoteci se pristupa. Ako želimo vidjeti neki drugi poddirektorij i datoteke kojima se pristupa unutar njega, poput direktorija: `/var/log/httpd/` koji koristi *Apache* server za zapisivanje log datoteka, to možemo postići na sljedeći način;

```
lsof +d /var/log/httpd/
```

Vidjet ćemo listu svih datoteka kojima netko trenutno pristupa. Možemo i provjeriti tko trenutno koristi (čita ili zapisuje) u točno određenu datoteku. U primjeru ćemo provjeriti tko koristi datoteku `/var/log/messages`, na sljedeći način (skratili smo ispis):

```
lsof /var/log/messages

COMMAND  PID USER    FD   TYPE DEVICE SIZE/OFF     NODE   NAME
rsyslogd 734 root     2w   REG    0,54    38445 27264337   /var/log/messages
```

Dakle program u stupcu `COMMAND` je `rsyslogd`. Prema tome, program `rsyslogd` trenutno koristi navedenu datoteku.



Za druge primjere upotrebe naredbe `lsof`, pogledajte i poglavlja:
**9.4.1. Unix Pipes.**
**9.4.3. Slijedeći koraci: *Unix* i *network socketi*.**
**25.7.6. Naredba *list open files* (*lsof*).**

**Izvori informacija: (222),(673),(K-12), `man lsof`.**

## 4.5.6. Ograničenja sustava i naredba *ulimit*

***Slijedi napredno poglavlje (4.5.6.x)!***

Naredba `ulimit` daje nam mogućnost definiranja raznih ograničenja sustava, od:

- Ograničenja broja procesa (programa) koji određeni korisnik može pokrenuti.
- Do ograničenja broja otvorenih datoteka i *file deskriptora*, kao i drugih ograničenja.

S druge strane naredba `prlimit` daje nam dodatne mogućnosti ograničenja korištenja resursa sustava.

**Izvori informacija:(589),(596),(597),(598), `man ulimit`,`man prlimit`,`man 5 limits.conf`.**

#### 4.5.6.1. Ograničavanje File deskriptora

Pošto smo maloprije pričali o *file deskriptorima*, pogledajmo koja ograničenja nam sustav daje na početku rada. Zanima nas koliko *file deskriptora* možemo imati, pri tome postoje; ***soft*** i ***hard*** ograničenja odnosno limiti. Ograničenje na *file deskriptore* znači koliko datoteka možemo imati istovremeno otvorenih, odnosno na koliko i s koliko njih istovremeno možemo raditi.
Pogledajmo ***soft*** *limit* na *file deskriptore*, što znači ograničenje do kojeg možemo ići bez ikakve poruke ili ograničenja sustava:

```
ulimit -Sn
1024
```

Pogledajmo ***hard*** *limit* na *file deskriptore*, što znači ograničenje preko kojeg ne možemo ići, jer nam sustav to neće dopustiti:

```
ulimit -Hn
4096
```

U slučaju kada nam je potrebno raditi s većim brojem datoteka i u konačnici *file deskriptora*, podignimo im *hard limit* ograničenje. Ovo je potrebno za neke velike baze podataka i slične napredne primjene, kod kojih koristimo (čitamo ih ili zapisujemo u njih) iznimno veliki broj datoteka. Mi ćemo trenutno podići ovo ograničenje (*limit*) na 8.192 *file deskriptora*.

```
ulimit -n 8192
```

Da bi ova promjena ostala stalna, potrebno ju je zapisati u posebnu datoteku `/etc/security/limits.conf` o kojoj ćemo govoriti kasnije. Što još možemo mijenjati s naredbom `ulimit` odnosno koji su njeni drugi prekidači i opcije:

- `-a` – prikazuje sva trenutna ograničenja sustava.
- `-c` – za maksimalnu veličinu *linux core* datoteke.
- `-d` – za maksimalnu veličinu podatkovnog (*data*) segmenta pojedinog procesa.
- `-e` – za maksimalni mogući prioritet procesa (***nice***).
- `-f` – za maksimalnu veličinu datoteka snimljenih od strane *shella* ili njegovih pod procesa.
- `-i` – za maksimalan broj signala poslanih prema procesima(pr. `sigkill`, `sigterm`, ...) koji su u stanju čekanja[3].
- `-l` – za maksimalnu veličinu memorije (kB) koja može biti u stanju „*memory lock*“ [1,2].(pogledajte poglavlje:**12.3.4.2.**).
- `-n` – za maksimalan broj otvorenih file deskriptora (pogledajte primjere gore, te poglavlje: **14.1.3.** te varijablu `/proc/sys/fs/file-max`).
- `-s` – za maksimalnu veličinu memorijskog stôga za svaki novi program (pogledajte poglavlje: **4.5.6.3.**).

**Kako izlistati procese po broju file deskriptora koje koriste, odnosno kako vidjeti koji proces ih koristi najviše?**
Pokrenimo sljedeću naredbu i dobiti ćemo listu procesa i pripadajućeg broja *file deskriptora* koje ima otvorene i to od najvećeg do najmanjeg broja (ovo je nîz naredbi koje treba izvršiti u jednom naredbenom retku):

```
lsof -n 2>/dev/null | awk '{print $1 " (PID " $2 ")"}' | sort | uniq -c | sort -nr |
head -25
```

**Izvori informacija: (589),(596),(597),(598), `man ulimit`,`man lsof`,`man limits.conf`.**

#### 4.5.6.2. Ograničenja procesa

U slučaju potrebe za ograničenjem maksimalnog broja programa (*procesa*) koje pojedini korisnik može pokrenuti, također možemo koristiti naredbu `ulimit` ili definirati ograničenja za pojedinog korisnika na razini cijelog sustava. Prvo pogledajmo koja su standardna ograničenja (*Soft* i/ili *Hard* limiti). Ispišimo trenutnu vrijednost *Soft limita* na broj procesa:

```
ulimit -Su
773215
```

Ispišimo trenutnu vrijednost *Hard limita* na broj procesa:

```
ulimit -Hu
773215
```

Sada ćemo za korisnika `ucenik1`, postaviti *Hard* i *Soft* limite na maksimalni broj pokrenutih procesa, i to:

- Maksimalno 100 procesa odnosno pokrenutih programa, kao *soft* limit.
- Maksimalno 500 procesa odnosno pokrenutih programa, kao *hard* limit.

Otvorimo datoteku `/etc/security/limits.conf` s programom `vi`.
Pri tome ćemo dodati sljedeće retke na kraj navedene datoteke:

```
ucenik1 soft nproc 100
ucenik1 hard nproc 500
```

Nakon što snimimo ove promjene i korisnik se ponovno logira, provjerimo njegovo stanje:

```
ulimit -Su
100
```

i

```
ulimit -Hu
500
```

Na primjerima vidimo kako su ograničenja aktivna, jer je prva vrijednost (*soft*) postavljena na 100, a druga (*hard*) na 500.
Na ovaj način smo ove promjene učili trajnima (permanentnima) odnosno otpornima na restart sustava.

**Baratanje s *file deskriptorima* u kernel memoriji**
Želimo li vidjeti više informacija o *file deskriptorima* na razini *kernela*, to možemo napraviti s naredbom `sysctl`:

```
sysctl fs.file-nr

fs.file-nr = 1184 0 9897041
```

Što to znači:

- `1184` – alociranih *file handel-ova* koji predstavljaju *file deskriptore* u kernel memoriji.
- `0` – nekorištenih, ali alociranih (težimo da je ovdje vrijednost 0).
- `9897041` – maksimalni broj *file handlera* (*file deskriptora*) za cijeli sustav tj. koliko ih kernel uopće može alocirati.

Postoji i drugi način provjere i konfiguriranja *file deskriptora*, pomoću direktne provjere ili upisa u kernel parametre. Podsjetimo se kako je `/proc` specifičan direktorij, koji sadrži parametre kernela.


Prvo provjerimo koliko kernel može maksimalno (sveukupno) alocirati *file deskriptora*.
**Dakle za cijeli sustav i sve programe na njemu**:

```
cat /proc/sys/fs/file-max

9897041
```

Na sličan način to možemo i promijeniti; primjerice smanjimo ih sa: `9.897.041` na `1.000.000`. Pogledajmo kako:

```
echo "1000000" > /proc/sys/fs/file-max
```

Ovdje smo proslijedili broj:`1000000` u datoteku `/proc/sys/fs/file-max`

Ako to stavimo u neku datoteku, koja se pokreće tijekom pokretanja sustava, to će postati permanentno (trajno).
Međutim, to se ipak preporučuje upotrebom ***sysctl*** metode. Pogledajte sljedeće poglavlje: **4.5.8 Naredba sysctl**.

**Dodatno je moguće ograničiti i broj *file deskriptora* za pojedine korisnike.**

Ograničit ćemo korisnika `ucenik` na maksimalan broj od 512 *file deskriptora*. Otvorimo datoteku: `/etc/security/limits.conf` s programom `vi`. Pri tome ćemo dodati sljedeće retke na kraj datoteke:

```
ucenik1 soft nofile 512
ucenik1 hard nofile 512
```

Kada se navedeni korisnik ponovno spoji na sustav (logira), imat će novo postavljeno ograničenje na broj *file deskriptora*.


Ova mogućnost se češće koristi za povećanje broja *file deskriptora*, obično za poslužiteljske servise kojima je potreban veliki broj istih. Primjerice vrlo velike i intenzivno korištene baze podataka kod kojih nije rijetko povećanje ovog broja do **65535** ili čak i više. Međutim, ovdje je važno ostati unutar granica vrijednosti postavljenih za cijeli operativni sustav.

**Izvori informacija: (589),(596),(597),(598), `man ulimit`,`man lsof`,`man limits.conf`,`man sysctl`.**

---

### 4.5.6.3. *Ulimit* - druga ograničenja

Postoji i posebno ograničenje koje se odnosi na ograničenje memorijskog *stôga* (engl. *Stack size*) koje se definira u kilobajtima (**kB**), a čiju maksimalno definiranu vrijednost možemo vidjeti s naredbom `ulimit` na sljedeći način:

```
ulimit -s

8192
```

Dakle standardno je dostupno 8MB memorije za memorijski stôg (*Stack*) za svaki proces. *Stôg* memorija je posebna memorija procesa (pokrenutog programa) koja se koristi kada se poziva neka funkcija; tada se rezervira blok podataka na vrhu memorijskog stôga za lokalne varijable i meta podatke. Kada se ta funkcija vrati programu, blok postaje neiskorišten i može se koristiti sljedeći puta kada se funkcija ponovno poziva. Međutim kada pozivamo neku funkciju, novi "*imenski prostor*" (engl. *Namespace*) je dodijeljen na stôg. Tako funkcije mogu imati lokalne varijable.
Kada funkcije pozivaju funkcije, koje naizmjenično pozivaju druge funkcije, zauzima se sve više i više prostora na *stôgu* kako bi se održala duboka hijerarhija prostora rada funkcija. Da bi se to ograničilo kako neki, a pogotovo zloćudni programi ne bi „*pojeli*" svu raspoloživu memoriju, sustav ograničava količinu memorije koja se može alocirati za ***stôg*** određenog programa (procesa). Osim povećanja ove memorije, moguće je i ukloniti sva ograničenja na veličinu memorije za *stôg*, sa sljedećom naredbom (ali **oprezno**):

```
ulimit -s unlimited
```

Međutim neki stariji programi ili specifični programi za stvarno posebne namjene, koji koriste ogroman broj rekurzivnih funkcija, mogu progutati ovu standardno postavljenu količinu memorije za *stôg*, pa ju je potrebno nešto povećati.
U normalnim uvjetima ova memorija se ne treba mijenjati.


Pogledajte i poglavlja:
**12.4 Anatomija programa u memoriji**
**6.2.2.1 Posebne sistemske varijable**
**5.9.5 Naredba *xargs* i ograničenje duljine argumenata**

**Izvori informacija: (589),(K-12), `man ulimit`,`man sysctl`.**

---

### 4.5.7. Datoteka `limits.conf` i druga ograničenja sustava

***Slijedi napredno poglavlje!***
Već smo spomenuli kako je moguće postaviti ograničenja za korisnike koja se postavljaju u konfiguracijskoj datoteci `/etc/security/limits.conf`. S njima je moguće postaviti ograničenja korisnika prema resursima sustava i prema procesima. Prvo ćemo pogledati važnije opcije koje možemo postaviti u konfiguracijskoj datoteci: `/etc/security/limits.conf` kao i zasebnim datotekama koje možemo kreirati kao dodatak na ovu konfiguracijsku datoteku. Naime ako imamo postavljene neke opcije u datoteci `/etc/security/limits.conf` i nismo konfigurirali dodatne konfiguracijske datoteke, tada će sustav čitati samo ovu datoteku i primijeniti pravila definirana u njoj.
Međutim moguće je definirati i zasebna pravila u zasebnim datotekama, a koje imaju veću težinu (važnost) kojom će pregaziti istu vrstu postavki iz datoteke `/etc/security/limits.conf`. Naime moguće je unutar direktorija: `/etc/security/limits.d/` kreirati datoteku (ili više njih), koje koriste istu sintaksu. Važno je da ove datoteke imaju ekstenziju `.conf`. Dakle ove datoteke možemo kreirati prema željenoj logici; primjerice za određene korisnike, grupe korisnika i slično. Primjer imena datoteke bi bio `/etc/security/limits.d/web-admini.conf`.

Pogledajmo što sve možemo definirati odnosno ograničavati u ovim datotekama, prema sljedećoj sintaksi:

***<domain>    <type>    <item>    <value>***

- **<domain>** može biti:
  - *USERNAME* – odnosno ime korisnika.
  - *GROUP name* – odnosno ime korisničke grupe sa sintaksom *@group*.
  - Posebni znak ***** koji označava standardnu postavku (*default*).
  - Posebni znak **%** koji označava ograničenje za „*maxlogin*“.
- **<type>** može biti:
  - *SOFT* – označava prvu razinu ograničenja odnosno *soft limit*.
  - *HARD* – označava krajnje (konačno) ograničenje odnosno *hard limit*.
- **<item>** može biti:
  - *AS* – definira ograničenje na adresni prostor u kB (`prlimit -v`).
  - *CORE* – definira ograničenje na ***core*** datoteku (*core file*) (`prlimit -c`).
  - *CPU* – definira ograničenje za izvršavanje programa/skripte (CPU vrijeme obrade) (`prlimit -t`).
  - *DATA* – definira ograničenje na maksimalnu veličinu podataka u kB (`prlimit -d`).
  - *FSIZE* – definira ograničenje na maksimalnu veličinu datoteka u kB (`prlimit -f`).
  - *LOCKS* – definira ograničenje na maksimalni broj zaključanih datoteka (*file lock*) korisnika ili grupe (`prlimit -x`).
  - *MEMLOCK* – definira ograničenje na maksimalni memorijski adresni prostor koji se može koristiti u kB (`prlimit -l`).
  - *MAXLOGINS* – definira ograničenje na maksimalni broj istovremenih (paralelnih) logiranja istog korisnika.
  - *MAXSYSLOGINS* – definira ograničenje na maksimalni broj istovremenih (paralelnih) logiranja istog korisnika direktno na sustav.
  - *MSGQUEUE* – definira maksimalnu količinu memorije koja se može koristiti za ***POSIX*** nîz za poruke (*message queue*) u bajtima (`prlimit -q`).
  - *NOFILE* – definira ograničenje na maksimalni broj otvorenih datoteka (`prlimit -n`).
  - *NICE* – definira ograničenje na maksimalnu vrijednost na koju se može povećati ***nice*** prioritet procesa.
  - *NPROC* – definira ograničenje na maksimalni broj pokrenutih procesa (`prlimit -u`).
  - *PRIORITY* – definira ograničenje na prioritet procesa s kojima korisnik/grupa mogu pokrenuti procese.
  - *RSS* – definira ograničenje na procese za upotrebu rezidentne memorije (*resident set size*) u kB (`prlimit -m`).
  - *RTPRIO* – definira ograničenje na maksimalnu vrijednost na koju se može povećati ***realtime*** prioritet procesa (`prlimit -r`).
  - *SIGPENDING* – definira ograničenje na maksimalni broj signala koji su u stanju čekanja (`prlimit -i`).
  - *STACK* – definira ograničenje na maksimalnu veličinu memorijskog stoga u kB (za svaki pojedini proces) (`prlimit -s`).
- **<value>** su vrijednosti koje definiramo u polju **<item>**, a postavljamo ovdje.

Postavljena ograničenja možemo vidjeti s naredbom `ulimit` na sljedeći način:

```
ulimit -a
```

Sva ovdje gore navedena dodatna ograničenja koja se tiču ograničenja svih novih procesa, možemo vidjeti i s naredbom:

```
prlimit

RESOURCE   DESCRIPTION                             SOFT      HARD  UNITS
AS         address space limit                unlimited unlimited  bytes
CORE       max core file size                         0 unlimited  blocks
CPU        CPU time                           unlimited unlimited  seconds
DATA       max data size                      unlimited unlimited  bytes
FSIZE      max file size                      unlimited unlimited  blocks
```

Slijedi nastavak ispisa:
```
LOCKS      max number of file locks held      unlimited  unlimited
MEMLOCK    max locked-in-memory address space     65536      65536  bytes
MSGQUEUE   max bytes in POSIX mqueues            819200     819200  bytes
NOFILE     max number of open files               32768      42768
NPROC      max number of processes                65535      65535
RSS        max resident set size              unlimited  unlimited  pages
RTPRIO     max real-time priority                     0          0
RTTIME     timeout for real-time tasks        unlimited  unlimited  microsecs
SIGPENDING max number of pending signals         224551     224551
STACK      max stack size                       9281536    9281536  bytes
```
Vidimo kako smo dobili trenutno postavljena ograničenja na cijeli sustav. Međutim, moguće je definirati i ograničenja na procese prema korisnicima ili korisničkim grupama koje ih pokreću, te dodatno za već pokrenute procese (programe).
Dakle i naredba `ulimit` i `prlimit` prije nego se ikoji proces pokrene, prvo provjerava ima li on neka ograničenja u:

- Datoteci: `/etc/security/limits.conf`.
- Ili u nekoj od datoteka unutar vršnog direktorija: `/etc/security/limits.d/`.

Ako su neka od ograničenja postavljena, ona će biti i primijenjena u trenutku pokretanja programa. Nakon što su ograničenja primijenjena, ona se zapisuju u datoteku: `/proc/PID/limits`, pri čemu ***PID*** broj označava ***PID*** broj pokrenutog programa (procesa). Pomoću naredbe `prlimit` možemo i vidjeti koja su efektivna ograničenja primijenjena na određeni pokrenuti program (proces). Ako primjerice naš program ima ***PID*** broj `1745`, tada ih možemo vidjeti s naredbom:
```
prlimit -p1745
```
Isto tako, moguće je i ručno promijeniti neka od ograničenja, na već pokrenuti program (proces). Ako se primjerice radi o istom ***PID*** broju i ako želimo tom procesu omogućiti povećanje broja otvorenih datoteka, što uključuje i *file deskriptore (na SOFT=*`32768` *te HARD=*`65535`), to možemo napraviti na slijedeći način. Prvo pogledajmo opis ove opcije:

- *NOFILE* – označava ograničenje na maksimalni broj otvorenih datoteka i postavlja se sa: `prlimit -n`

Navedenu promjenu ćemo stoga napraviti na slijedeći način:
```
prlimit -p1745 -n=32768:65535
```
U svakom slučaju sva ograničenja koja definiramo u direktoriju `/etc/security/*` primjenjuju se kada se program pokreće od strane nekog korisnika odnosno, ako program sâmi pokrećemo, tek nakon što se logiramo kao željeni korisnik. Ako ipak želimo iz našeg korisnika postati neki drugi korisnik, bez odlogiranja i ponovnog logiranja, to možemo izvesti pomoću naredbe `sudo` i to možemo napraviti, ako koristimo prekidače `-i -u`. Ako primjerice želimo postati korisnik `web`, ali da se postave i sva njegova ograničenja i sve postavke sustava (*ENV* varijable), moramo to napraviti na sljedeći način:
```
sudo -i -u web
```


***PRLIMIT*** funkcionalnost je dostupna tek od kernela 2.6.36, dakle tek na **RedHat**/**CentOS** 7.x ili novijima!

**Izvori informacija: (596),(597),(598),(K-14),** `man limits.conf`, `man prlimit`, `man sudo`.

## 4.5.8. Naredba *sysctl*

***Slijedi napredno poglavlje!***
***Sysctl*** je sučelje za baratanje s parametrima sustava na razini kernela, koje je bazirano na upotrebi *sysctl* varijabli. *Sysctl* varijable imaju i svoj pâr unutar **`/proc`** stabla direktorija (i datoteka). U *sysctl* postavke se uklapaju i primjeri s *file deskriptorima* iz prethodnog poglavlja, a koje možemo permanentno konfigurirati korištenjem *sysctl* metoda. U datoteci `/etc/sysctl.conf` se nalaze upisani parametri kernela koji će biti učitani tijekom svakog pokretanja sustava. Na novijim inačicama *RedHat/Rocky* Linuxa v.7.x/8+ moguće je koristiti i zasebne datoteke, svaku imena poput: `IME.conf` unutar direktorija: `/etc/sysctl.d/` (u najnovijim inačicama u `/usr/lib/sysctl.d/`) u koje je moguće upisivati iste vrijednosti kao i u datoteku `/etc/sysctl.conf`. Pri tome `IME` datoteke (`IME.conf`) može biti bilo koje ime ili ime s rednim brojem, pri čemu se prvo učitavaju datoteke s manjim početnim brojem; primjerice: `12-diskovi.conf` će se učitati prije `25-mreza.conf`.

**Slijede klasični primjeri uporabe.**
Dodajmo sljedeće retke (preporuča se i još jedan redak s opisom), u datoteku: `/etc/sysctl.conf`
```
# Maximum number of open files permited
fs.file-max = 1000000
```
Ovdje smo postavili maksimalnu vrijednost na 1.000.000 *file deskriptora* za sve programe (i mrežne konekcije) pokrenute na Linux sustavu. Za provjeru što se sve može konfigurirati preko `sysctl` sučelja trebamo pokrenuti sljedeću naredbu:
```
sysctl -a
```
Ako želimo ponovno postaviti vrijednosti koje smo definirali u `/etc/sysctl.conf`, tada pokrenimo sljedeću naredbu:
```
sysctl -p /etc/sysctl.conf
```


*Na (novijim) sustavima koji koriste **systemd** servis, za inicijalno čitanje i postavljanje **sysctl** varijabli koristi se servis:* ***systemd-sysctl****. Na tim sustavim se za ponovno učitavanje postavki upisanih u gore navedene datoteke može restartati ovaj servis:*
```
systemctl restart systemd-sysctl
```



Za druge **sysctl** primjere pogledajte i slijedeće cjeline:

- **4.5.6.2. Ograničenja procesa.**
- **9.3.1.1. Još detaljniji rad procesa i nîti.**
- **10.7.2.2. NUMA.**
- **12.3.2.1. Paging i dijeljenje memorije (Shared Memory).**

Pogledajte i druge cjeline u kojima se također spominju **sysctl** varijable**:**

- **12.3.4.2. Huge Pages.**
- **12.3.5. Zone RAM memorije.**
- **12.5. *Out of Memory* stanje rada i konfiguracija *OOM killera*.**
- **13.8.2.1. Malo više detalja o *swap* sustavu i particiji.**
- **14.1.2. Optimizacija *Page Cache* sloja** i poglavlja koja slijede te
- **20.7.1.2. Rad s ARP protokolom.**
- **20.7.1.3. Scenarij za prisilno osvježavanje ARP tablice (*Gratuitous ARP*) detaljnije.**
- **22.3.3. Multicast u upotrebi.**
- **23.2.1. Optimizacije vezane za fragmentaciju i MTU.**
- **24.2.2. Kako se uspostavlja TCP veza** i poglavlja koja slijede te
- **25.6.8. Ruting i optimizacija ruting parametara**.
- **25.7.1.1. ICMP poruka.**

Pogledajte i poglavlje o vatrozidu**: 26.7.2.1. Primjeri** te poglavlje:

- **28.3. Sigurnosne preporuke.**

Sve *sysctl* varijable grupirane su u nekoliko kategorija: **ABI**, **fs**, **kernel**, **net**, **sunrpc**, **user** i **vm**, a opisane su na izoru informacija **(1405)**: https://www.kernel.org/doc/Documentation/sysctl/. ***Sysctl*** naredba se koristi i na ***FreeBSD UNIXu***.

**Izvori informacija: (157),(221),(325),(340),(341),(478),(642),(1405),(K-12),(K-14), `man sysctl`,`man sysctl.d`, `man sysctl.conf`,`man systemd-sysctl`,`man 5 proc`.**

---

## 4.5.9. Naredba *fuser*

***Slijedi napredno poglavlje!***

Naredba `fuser`, koristi se, za razliku od naredbe `lsof` kada imamo potrebu provjeriti koji proces koristi određenu datoteku, direktorij ili mrežni *socket*. Dodatno, moguće je vidjeti i informacije o tom procesu, kao i vrsti pristupa na njega. Iako namjena ove naredbe nije direktno vezana za naredbu `lsof` kao niti za *file deskriptore*, njihova veza je indirektna, pa ju stoga spominjemo u ovom poglavlju.

Naredba `fuser` u svom ispisu, pod dijelom vrste pristupa (*ACCESS*), nudi nam slijedeće statistike o vrsti pristupa:

- `c` – označava trenutni direktorij (engl. *Current directory*).
- `e` – ovo je izvršna datoteka (engl. *Executable*) koja je pokrenuta.
- `f` – ovo je otvorena datoteka, slovo *f* se obično ne prikazuje u standardnom prikazu.
- `F` – ovo je otvorena datoteka za zapisivanje (*write*), slovo *F* se obično ne prikazuje u standardnom prikazu.
- `r` – ovo je *root* direktorij.
- `m` – ovo je memorijski mapirana (*mmap*) datoteka ili dijeljena biblioteka.

Česti prekidači ove naredbe su:

- `-c` – označava traženi *montiran* datotečni sustav.
- `-k` – ubij pronađene procese.
- `-i` – interaktivni rad: prvo pitaj prije nego što nešto radiš (pogotovo, ako se koristi `-k`).
- `-l` – lista signala koje možemo slati procesu kod ubijanja (koje navodimo u *-IME-SIGNALA* odnosno *-SIGNAL*).
- `-m` – označava sve procese koji pristupaju datotečnom sustavu (*mount point*) na kojemu se nalazi i datoteka koju gledamo.
- `-n` – označava mrežni prostor, koji je standardno datoteka (*file*), ali može biti i:
  - `-n tcp` – označava TCP transportni mrežni protokol, odnosno otvoreni TCP port.
  - `-n udp` – označava UDP transportni mrežni protokol, odnosno otvoreni UDP port.
  - `-4` – označava samo upotrebu IPv.4 protokola, dok `-6` označava samo upotrebu IPv.6 protokola.
- `-v` – označava detaljniji (*verbose*) ispis.
- `-SIGNAL` – vezano za `-k`. Ovo je oznaka signala koji će se poslati procesima koje se ubija s prekidačem `-k`.

Lista signala koje standardno možemo poslati procesima je sljedeća:

- *HUP INT QUIT ILL TRAP ABRT IOT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH IO PWR SYS*

**Slijede primjeri**

**1.** Pogledajmo koji procesi koriste datoteke unutar našeg trenutnog (radnog) direktorija (*/root*):

```
cd /root/
```

Te ih konačno provjerimo s naredbom `fuser`:

```
fuser -v ./

                     USER        PID ACCESS COMMAND
/root:               root      15125 ..c..  bash
```

Koristili smo i prekidač `-v` za detaljniji ispis. Vidimo kako je unutar direktorija */root* aktivan linux proces s ***PID*** brojem `15125`, koji je zapravo pokrenuta naredba `bash` koja je naša aktivna ljuska (*shell*), koja ima pristup na direktorij: oznaka `c` pod stupcem `ACCESS`.

**2.** Sada pogledajmo obratno: zanima nas koji proces koristi, odnosno koji proces je pokrenuo konkretnu naredbu (*bash*) koja se nalazi na putanji */bin/bash*:

```
fuser -v /bin/bash
                     USER        PID ACCESS COMMAND
/bin/bash:           root      15125 ...e. bash
```

Vidimo kako je korisnik *root* pokrenuo ovu naredbu, čiji je ***PID*** broj `15125`, a pristup je označen kao: *e* što znači kako se radi o izvršnoj datoteci.

**3.** U ovom primjeru, zanima nas tko je otvorio TCP mrežni port 22 (TCP port 22 koristi ***SSH*** poslužitelj za udaljeni pristup):

```
fuser -v -n tcp 22

                     USER        PID ACCESS COMMAND
22/tcp:              root       6796 F....  sshd
                     root      14418 f....  sshd
```

Isto smo mogli dobiti i sa:

```
fuser -v 22/tcp

                        USER     PID ACCESS COMMAND
22/tcp:              root       6796 F....  sshd
                     root      14418 f....  sshd
```

Vidljivo je da je TCP port broj `22` otvorio proces/naredba imena `sshd` što je zapravo *SSH* poslužiteljski servis (*daemon*) te vidimo sve ***PID*** brojeve od oba *SSH* procesa. Prvi u ovom slučaju ***PID*** (`6796`) je inicijalni SSH servis, a drugi (`14418`) je onaj koji se pokrenuo za naše spajanje (preko njega smo se udaljeno spojili na SSH).
Kada bi se još jedan SSH klijent spojio na ovaj poslužitelj imali bi sljedeće stanje:

```
fuser -v 22/tcp

                     USER        PID ACCESS COMMAND
22/tcp:              root       6796 F.... sshd
                     root      14418 f.... sshd
                     root      14770 f.... sshd
```

**4.** Pronađimo sve procese, prema ***PID*** brojevima, koji koriste *montirani* datotečni sustav koji je povezan s diskom */dev/sdb*:

```
fuser -c /dev/sdb


/dev/sdb:
344   1688  1689  3462  3480  3494  3787  3806  3814  3820  3834  3835  3836  3837  3838  3840
3853  3868  3906  3907  3919  4018  4050  4076  4077  4078  4080  4081  4084  4090  4145  4146
4168  4235  4241  4242  4269  4320  4321  4322  4323  4324  4325  5770 14545 14763 15123
```

Dobili smo listu svih ***PID*** brojeva, svih procesa koji su otvorili neku datoteku na navedenom disku (*/dev/sdb*).
Za više detalja smo mogli koristiti i prekidač *-v*, ali bi dobili predugačku listu (za našu potrebu prikaza).

**4.1** U slučaju kada se sustav nalazi u kritičnom stanju i ima greške na particiji, kao krajnja mjera bi bila ubijanje svih procesa koji drže bilo koje datoteke otvorene na navedenom disku, ako imamo potrebu *odmontirati* (*umount*) particiju pa ju testirati odnosno skenirati i popraviti eventualne greške na njoj. U ovom primjeru se radi o particiji `/dev/sdb`.


Oprez - nemojte ovo raditi jer ćete srušiti cijelo računalo odnosno Linux sustav!

```
fuser -c -i -k /dev/sdb

...
```

Isto tako smo mogli poubijati odnosno pozatvarati procese koje drže određene (pojedine) datoteke te krenuti dalje.

Ako naredba `fuser` nije instalirana, možete ju instalirati pomoću menadžera softverskih paketa, s naredbom `yum`:

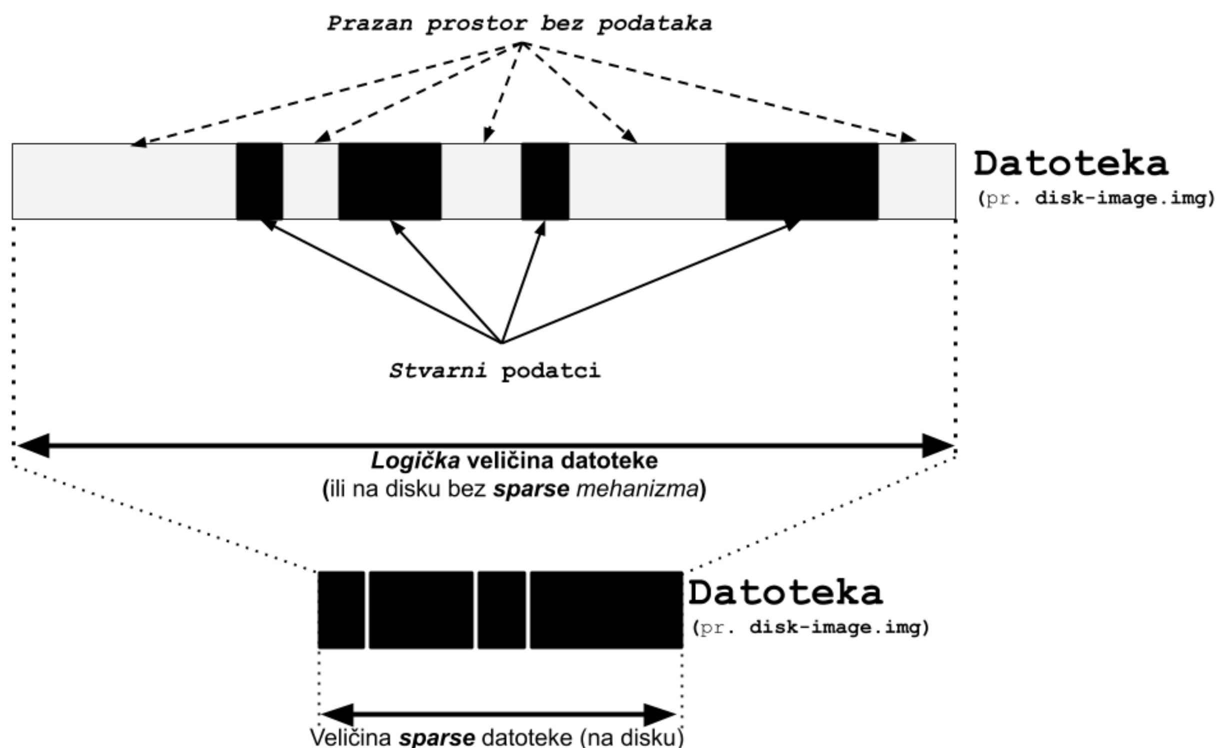
```
yum install psmisc
```

**Izvor informacija:** `man fuser`.

# 4.6. Prorijeđene (*Sparse*) datoteke

U ovom **naprednom** poglavlju pozabavit ćemo se s takozvanim prorijeđenim datotekama koje se nazivaju ***Sparse files*** (*Sparse* iz engleskog: rijetke ili prorijeđene [datoteke]). Naime u normalnom radu, svaka datoteka može imati određene praznine ili prazna područja odnosno područja koja sadrže nule. To možemo promatrati i ovako: zamislimo knjigu od stotinu stranica, na kojoj je svaka druga stranica prazna. Ovakva knjiga fizički zauzima stotinu stranica, ali efektivno ih je iskorišteno samo polovina. Isto ili barem u nekoj mjeri slično može biti i s datotekama jer se kod datoteka koje sadrže nule odnosno *praznine*, na disk zapisuju i one, pa se u konačnici prostor na disku zauzima i s praznim prostorom i naravno s konkretnim sadržajem odnosno podacima.
To se događa stoga što se na razini diskovnog sustava podaci alociraju (i zapisuju) u blokovima, bez obzira nalaze li se u njima stvarni podaci ili nule. Vezano za prorijeđene datoteke, i ovo su datoteke u osnovi baš kao i bilo koje druge, osim što blokovi podataka koji se zapravo zapisuju na disk, a koji sadrže samo nule odnosno po logici praznine, nisu stvarno i pohranjeni na disku. U slučaju upotrebe prorijeđenih datoteka, to znači da možete imati datoteku koja je naizgled velika **10 GB**, ali od tih **10 GB**, ako stvarnih podataka ima samo 1 GB, na disk će se zapisati samo tih **1 GB** (konkretnih) podataka.
Pogledajmo logički izgled prorijeđene datoteke na slici **14**.

**Slika 14. Logički pogled na prorijeđenu (*Sparse*) datoteku.**




Prepoznavanje i upotreba prorijeđenih datoteka može biti osobito korisna u situacijama u kojima se puni kapacitet datoteke možda nikada u potpunosti neće iskoristiti. Jedna od takvih situacija je virtualizacija odnosno diskovi virtualnih strojeva. Naime kod virtualnih strojeva (računala), disk virtualnog stroja je zapravo datoteka na fizičkom računalu odnosno na poslužitelju za virtualizaciju. Ta datoteka je virtualnom stroju vidljiva kao normalni disk, koji je prvo potrebno particionirati, a potom i formatirati. Normalno je da se u procesu formatiranja zauzima određeni postotak od ukupnog kapaciteta diska (*cca*. do 5%), a u standardnom procesu formatiranja cijela površina diska neće se popuniti s podacima, osim s onima potrebnim za sâmo formatiranje. Međutim niti nakon tog trenutka cijela površina diska nije popunjena s više od navedenih 5% zbog sâmog formatiranja. Prema tome nisu niti zapisane nule u prazna područja (osim u slučaju punog formatiranja). Dodatno, ako i dalje u radu virtualni stroj nikada ne popuni disk u potpunosti, to znači da određena količina diska nikada neće sadržavati ništa osim eventualno nula u njemu, što u konačnici omogućuje uštedu prostora na disku, pomoću tehnologije prorijeđenih datoteka.

**Što se događa u inicijalnom trenutku kreiranja ovakve datoteke?**
Operativni sustav koji podržava prorijeđene datoteke kreira ovakvu datoteku, koja je zapravo prazna, odnosno naznačeno je da je popunjena blokovima nula. Čim neki blok sadrži podatke, podaci se zapisuje na disk na odgovarajući način i datoteka na disku raste. Osim vrlo korisne upotrebe, naročito kod virtualizacije, postoje i neki potencijalni problemi s korištenjem prorijeđenih datoteka. Najočigledniji bi bio u slučaju kada *korisnički* program ili Linux naredba ne prepoznaje ili ne zna koristiti prorijeđene (*sparse*) datoteke. Rezultat ovakvog rada bi bio da u gornjem slučaju kada imamo prorijeđenu datoteku od 10GB, ali koja je iskorištena samo 1GB te nakon upotrebe našeg „problematičnog“ programa ili naredbe ona odjednom naraste na stvarnih 10GB. Čak i programi koji mogu i znaju raditi s prorijeđenim datotekama, to im uglavnom mora biti izričito rečeno.

Drugi problem je u tome što se sve vezano za upravljanjem diska temelji se na tome da se koriste sektori ili blokovi na disku, a veličina diska prijavljena za prorijeđenu datoteke je puna veličina datoteke, a ne stvarni broj blokova na disku. To također znači da će bilo koji program koji radi isključivo s blokovima na disku (npr. naredba `du`) prijaviti drugačije vrijednosti (veličine) od ostalih programa ili servisa. Još jedan problem je: prepoznaju ili *backup* programi i snimaju li ispravno u ili s prorijeđenim datotekama. *Backup* program koji ne prepoznaje (i nepravilno koristi) prorijeđenu datoteku za rezultat ima već navedeni primjer u kojemu mala prorijeđena datoteka od 1GB odjednom naraste na 10GB te za nju eventualno više nema prostora za pohranu.

Da biste pomoću naredbe `ls` dobili stvarnu veličinu prorijeđene datoteke umjesto samo veličine datoteke koja je prijavljena sustavu, potrebno je koristiti i prekidač `-s`. Pogledajmo kako:

```
ls -lsh

total  36K
 24K   -rw-r--r-- 1 root root 22K Jun  2 09:42 file.out.txt
4.0K   -rwxr-xr-x 1 root root  68 Jun  1 16:32 generator.sh
```

Blokovi koji se koriste u lijevom stupcu (`total`) su kilobajti (**KB**) korišteni za alociranje blokova na disku. Veličina sâmih datoteka, koja je prijavljena sustavu je vidljiva u šestom stupcu. Ako usporedimo veličinu alociranih blokova i prijavljenu veličinu datoteka vidimo kako se radi o normalnim datotekama odnosno ovdje nema prorijeđenih (*sparse)* datoteka.
Stvaranje *sparse* datoteka u osnovi je jednostavan proces za koji ćemo koristiti naredbu `dd`.

S naredbom `dd` kreirat ćemo datoteku, s generatorom nula `/dev/zero` kao ulazom (`if=`) te skačemo na poziciju 10 GB dalje (`seek=10G`). Na ovaj način je kreirana prazna datoteka, koja kreće s nulama, te skače na poziciju 10GB dalje. Zbog toga je njena vidljiva veličina +10GB stvarna jer Linux podržava prorijeđene datoteke, a u stvarnoj upotrebi je samo 8KB.
To znači kako ona zapravo zauzima samo dva bloka od 4KB na površini diska.



*Prisjetimo se da je standardna veličina bloka **4KB** (ovo je zapravo veličina klastera)!*

U primjeru koji slijedi kreirati ćemo navedenu *sparse* datoteku imena: `sparse-file` veličine 10GB, pomoću naredbe `dd`

```
dd if=/dev/zero of=sparse-file bs=1 count=1 seek=10G

1+0 records in
1+0 records out
1 byte (1 B) copied, 5.0734e-05 s, 19.7 kB/s
```

Pogledajmo što smo dobili:

```
ls -lsh

total 36K
 24K -rw-r--r-- 1 root root 22K Jun  2 09:42 file.out.txt
4.0K -rwxr-xr-x 1 root root  68 Jun  1 16:32 generator.sh
8.0K -rw-r--r-- 1 root root 11G Jun  2 09:43 sparse-file
```

Vidljivo je da je naša prorijeđena datoteka prijavljene veličine `11GB`, ali da u blokovima (prvi stupac lijevo) zauzima samo `8.0KB` jer je potpuno prazna. Dakle ona stvarno od datotečnog sustava alocira samo 8KB.
Usporedimo nekoliko Linux naredbi i njihovu "svjesnost" o prorijeđenim datotekama.

## Naredba du

Naredba `du` (engl. *disk usage*) nam daje podatak o veličini datoteka na disku; unutar nekog direktorija ili rekurzivno i za sve pôddirektorije unutar trenutnog direktorija (naravno i za sve datoteke). Sve inačice ove naredbe svjesne su prorijeđenih datoteka, ali je moguće vidjeti i veličine koje su prijavljene sustavu kao neprorijeđene. Pogledajmo kako to vidjeti:

```
du -hs

40K     .
```

Dakle zauzeće u trenutnom direktoriju s jednom prorijeđenom datotekom od 11GB je zapravo 40KB.

Sada pogledajmo kako se ovakva datoteka vidi inače na sustavu (prekidač `--apparent-size`), primjerice s naredbom `ls`:

```
du -hs --apparent-size

11G     .
```

Sada vidimo da je vidljiva veličina 11GB jer se sada u prikazu ne prepoznaju prorijeđene datoteke.

## Naredba tar

Naredba `tar` koristi se za kreiranje arhive, s time da ona ima mogućnost i komprimiranja iste arhive "u letu".



Za više informacija o programu/naredbi **`tar`** i njegovim mogućnostima pogledajte poglavlje o arhiviranju:
**8.1. Arhiviranje, komprimiranje i dekomprimiranje**.

Naredba `tar` je jedna od onih kojoj se mora reći da su u pitanju *sparse* datoteke, ako radimo s njima.
Izradimo nekomprimiranu *TAR* arhivu, u ovom slučaju naziva: `sparse-file.tar`, od naše *sparse* datoteke: `sparse-file`

```
tar -cv -f sparse-file.tar sparse-file
```

Kreiranje ovakve arhive koja nije svjesna *sparse* datoteka, će potrajati, kao i njeno kasnije ekstrahiranje; možda i satima, sve ovisno o brzini računala/poslužitelja. Pogledajmo što smo dobili?

```
ls -alhs

6.1G -rw-r--r-- 1 root root 6.1G Jun  2 14:16 sparse-file.tar
```

Dobili smo arhiviranu datoteku koja je veličine **6.1GB** i koja na disku zauzima također **6.1GB**. Dakle dokazali smo da se ovakvom primjenom ne koristi mehanizam za prorijeđene datoteke (*sparse files*).

Sada probajmo napraviti slično; ime arhive će sada biti: `sparse-file-S.tar`, ali dodajmo `tar`-u spoznaju kako se radi o *sparse* datoteci, pomoću dodatnog prekidača `-S`. Stoga napravimo ***tar*** arhivu na sljedeći način:

```
tar -cvS -f sparse-file-S.tar sparse-file
```

Pogledajmo i što smo sada dobili:

```
ls -alhs

12K -rw-r--r-- 1 root root  10K Jun  2 14:16 sparse-file-S-.tar
```

Ovdje imamo posve drugačiji rezultat; veličina u blokovima na disku je samo **12KB**, dok je vidljiva veličina **10KB**.
U svakom slučaju razlika je vidljiva i jasno je da se sada koristio *sparse* mehanizam.

Probajmo sada uključiti i `gzip` komprimiranje, pomoću prekidača `-z`, ali bez *sparse* podrške:

```
tar -czv -f sparse-file.tgz sparse-file
```

Rezultat će biti datoteka `sparse-file.tgz` koja je zapravo *GZIP* komprimirana *TAR* arhiva.

```
ls -alhs

10M -rw-r--r-- 1 root root  10M Jun  2 14:11 sparse-file.tgz
```

Sada smo dobili znatno manju arhivu, ali koja kada se bude ekstraktirala, biti će veličine **10GB**, što baš i ne želimo.

I na kraju napravimo slično kao u prethodnom primjeru, ali ponovno uključimo podršku za *sparse* (`-S`) na sljedeći način:

```
tar -czvS -f sparse-file-S.tgz sparse-file
```

Te pogledajmo kako to ovaj puta izgleda:

```
ls -alhs

4.0K -rw-r--r-- 1 root root  138 Jun  2 14:14 sparse-file-S.tgz
```

U ovom primjeru smo dobili malu arhivu, koja je svjesna prorijeđenih datoteka (**4KB** u blokovima i **138** bajta), te će ona, kada bude ekstraktirana (dekomprimirana) također zadržati *sparse* svojstva.

## Naredba `cp`

Linux naredba `cp` za kopiranje datoteka ili direktorija u većini slučajeva standardno može kopirati i prorijeđene datoteke.
Ona na osnovi heurističke analize datoteke, može zaključiti radi li se o običnoj ili prorijeđenoj datoteci.
Naime ona prije kopiranja prvo provjerava ima li datoteka dovoljno veliki nîz nula u sadržaju, te ako ima, smatra se prorijeđenom datotekom i kao takva se dalje koristi.

*Iako, da bi bili potpuno sigurni da će sparse datoteka biti ispravno prekopirana, možemo koristiti prekidač* `--sparse=always`

Pogledajmo kako to napraviti:

```
cp --sparse=always sparse-file /root/test/new-directory
```

Provjerimo je li se sve prekopiralo kako treba i kolika je stvarna veličina datoteke na disku (`-s` prekidač):

```
ls -alhs

8.0K -rw-r--r-- 1 root root 11G Jun  2 09:43 sparse-file
```

Dakle sve je u redu, jer je stvarno zauzeće diska (prvi stupac) samo `8 KB`.

## Naredba `rsync`

Naredba `rsync` odnosno program za izradu sigurnosnih kopija, između cijelog nîza mogućnosti, podržava i *sparse* datoteke.



Za primjere i više detalja o ovoj naredbi pogledajte poglavlje: **8.1.2 Naredba** *rsync*



Programi poput: `scp`, `ftp`, `sftp` i `pax` ne podržavaju *sparse* datoteke!.

Naredba `df` za prikaz zauzeća prostora na disku prema (montiranim) particijama prikazuje stvarno zauzeće blokova na disku odnosno prikazuje stvarno alociran prostor, što znači kako je svjesna *sparse* datoteka.



Za druge primjere upotrebe naredbe **`df`** pogledajte poglavlje: **13.10.2. Naredba *df*.**

**Izvori informacija: (139),(674),** **`man ls`**,**`man dd`**,**`man du`**,**`man tar`**,**`man cp`**,**`man rsync`**.

# 4.7. Format datoteka (*MIME type*)

Datoteke se osim po već navedenim posebnim vrstama, razlikuju i po unutarnjem formatu koji kategorizira unutarnja struktura podataka zapisanih u datoteku.
Na operativnim sustavima *Windows*, format datoteke definira njena ekstenzija; primjerice datoteka imena: `film.mp4` se identificira od strane sustava kao video datoteka u kojoj je video sadržaj komprimiran i (en)kodiran pomoću *MPEG*-4 algoritma (*codec-a*). Isto tako se primjerice datoteka s ekstenzijom `.mp3` identificira kao glazbena datoteka (*mp3*), a datoteka s ekstenzijom `.html`, kao HTML (web) dokument i tako dalje.

Međutim na operativnim sustavima *Unix* i Linux razlučivanje datoteka prema ekstenziji nije nužno i potrebno. *Unix* i *Linux* operativni sustavi prvo provjeravaju ekstenziju datoteke i uspoređuju ju s definicijom koja se nalazi u datoteci `/etc/mime.types`. Ova datoteka dolazi u softverskom paketu imena `mailcap`.

U navedenoj datoteci popisane su sve pripadnosti ekstenzija s pripadajućim kategorijama aplikacija.
U ovu datoteku se zapisuju identifikatori na sljedeći način: `kategorija/vrsta  ekstenzija`. Pogledajmo par unosa u njoj:

```
text/plain              txt asc text pm el c h cc hh cxx hxx f90 conf log
video/mp4               mp4 mpg4 m4v
application/gzip        gz tgz
```

Asocijacije s kojim programom se otvara koja kategorija ekstenzije se definiraju ovisno o (grafičkom) sučelju koje koristimo. Međutim u slučaju kada datoteka nema ekstenziju (ili nema ove ***mime*** datoteke), *Unix/Linux* radi sljedeće. Naime svaka od datoteka, ovisno o svom unutarnjem formatu podataka, odnosno pripadnosti, unutar sebe nosi zapis u kojemu se nalazi opis njenog unutarnjeg formata.

U grubo, postoje dvije kategorije datoteka:

- Obične tekstualne datoteke; poput *shell* i drugih skripti: koje u prvom retku (liniji) datoteke sadrže definiciju s kojim programom (ili ljuskom) se moraju pokrenuti.
- Binarne datoteke – one koje sadrže binarne podatke (sve osim standardnog *teksta*).



Svaka binarna datoteka (obično svaka ona koja nije tekstualna) u svom zaglavlju odnosno u barem prvih nekoliko blokova od obično osam (8) bajta, mora sadržavati identifikatore o kojem formatu odnosno vrsti internog sadržaja se radi.

Ako pogledamo navedenu binarnu datoteku imena: `film.mp4`, ali ne s uređivačem teksta već isključivo s nekim heksadecimalnim preglednikom; primjerice sa: `hexdump`, `od` ili `xxd` vidjet ćemo i njen stvarni (binarni) sadržaj.
Stoga pogledajmo sâmi početak datoteke i to prvih pet (5) redaka (ovdje je po 16 bajta u jednom retku), pomoću naredbe `xxd`:

```
xxd -l 0x50 film.mp4

00000000: 0000 0020 6674 7970 6973 6f6d 0000 0200   ... ftypisom....
00000010: 6973 6f6d 6973 6f32 6176 6331 6d70 3431   isomiso2avc1mp41
00000020: 0000 0008 6672 6565 0000 75f8 6d6f 6f76   ....free..u.moov
00000030: 0000 006c 6d76 6864 0000 0000 7c25 b080   ...lmvhd....|%..
00000040: 7c25 b080 0000 03e8 0000 6da9 0001 0000   |%........m.....

. . . slijedi sadržaj datoteke (MPEG-4 video) . . .
```

Već u prvom retku vidimo osnovne podatke o ovoj datoteci:

- `ftyp` (HEX: `66 74 79 70`) – znači kako se radi o „*QuickTime Container*“ formatu odnosno vrsti datoteke.
- `isom` (HEX: `69 73 6f 6d`) – znači kako se radi o pôd formatu(A): “*MP4 Base Media v1*(B)(C)“.
- `avc1` (HEX: `61 76 63 31`) – znači kako se radi o video datoteci koja je komprimirana i kodirana pomoću AVC1 algoritma, što se odnosi na *H.264* odnosno *MPEG-4 Part 10 format*.
- `mp41` (HEX: `6d 70 34 31`) – konkretno označava upravo *MPEG-4 Part 10* format zapisa.
- Ostatak nećemo dalje analizirati.

Ovdje je vidljivo da svaki heksadecimalni nîz od osam (8) bajta na određenoj poziciji (*offsetu*) identificira određeni dio formata odnosno, ako gledamo sâm početak datoteke, prvih nekoliko bajta definira format datoteke, kako je navedeno u izvoru informacija **(788)**.
Ovih prvih nekoliko bajta u zaglavlju svake binarne datoteke se zove čarobni broj (engl. *magic number*) datoteke.
Jasno je da svaki format datoteke mora imati točno definiranu strukturu kao i opisnik sadržaja datoteke u zaglavlju.

Kako bi se datoteke različitih formata; primjerice: *MP3, MP4, HTML, JPG, DOC i druge*, uopće mogle uredno zapisivati, ali i kasnije čitati, čak i neovisno o operativnom sustavu, format datoteke mora biti definiran nekim standardom.



Standardizacijsko tijelo koje definira standarde zapisa (formata) datoteka je ***IANA***.
Pogledajte izvor informacija: **(785).**

Na Linuxu, postoji datoteka u kojoj su pohranjeni „*magic numbers" oznake,* a to je datoteka: `/usr/share/misc/magic` koja dolazi u softverskom paketu imena; `file-libs`. U njoj su pohranjeni svi ovi „*čarobni brojevi*" uz pripadajuće opise.

Ako i sâmi želimo provjeriti o kojem se formatu datoteke radi, to možemo napraviti s naredbom `file`, koja (indirektno) kontaktira ovu datoteku.
Na sljedeći način možemo provjeriti kojeg formata je, već navedena datoteka imena: `film.mp4`

```
file film.mp4
```

I dobit ćemo lijepo formatirane informacije o ovoj konkretnoj *MP4* (*MPEG* 4) datoteci, koje smo već objasnili.



Vrsta odnosno format podataka unutar datoteke je prvi puta definiran kao takozvani „***MIME*** type" odnosno „*Multipurpose Internet Mail Extensions*", za potrebe ubacivanja privitaka u elektroničku poštu, ali je kasnije prihvaćen kao standard za definiciju formata datoteka. S time da se ovdje pod formatom misli na točnu definiciju zaglavlja te osnovnu definiciju ostatka koji slijedi; poput načina ***(en)kodiranja*** i zapisivanja podataka koji slijede nakon početnog zaglavlja datoteke.

Sve dalje od toga u datoteci (ono što od podataka slijedi) definira se na razini sâme aplikacije koja i treba koristiti te podatke.



Vezano za elektroničku poštu, pogledajte i poglavlje:
**25.8.8. Elektronička pošta (e-mail).**



Vezano za (en)kodiranje, pogledajte i poglavlje:
**10.3.2.1. Kodiranje i dekodiranje.**

**Izvori informacija:(780),(781),(782),(783),(784),(785),(786),(787),(788),(1135),(K-12), `man file`, `man magic`, RFC 6838**

## 4.7.1. CSV format datoteka

U ovoj cjelini upoznat ćemo se s jednim od često korištenih formata datoteka koji se naziva ***CSV*** (eng. *Comma-separated values*). U doslovnom prijevodu ovaj format naznačava kako se radi o tekstualnoj datoteci unutar koje se dijelovi teksta razgraničavaju odnosno *delimitiraju* (obično) pomoću znaka zarez (`,`). Ovdje se radi o tekstualnoj datoteci koja koristi zarez za odvajanje vrijednosti. Svaki redak ove datoteke predstavlja novi zapis podataka. Svaki se zapis sastoji od jednog ili više polja, odvojenih zarezom. Upotreba zareza kao separatora polja uzrok je naziva za ovaj format datoteke (engl. *comma*=zarez).

*CSV* datoteka obično pohranjuje tablične podatke (brojeve i tekst) u običnom tekstu, a u tom će slučaju svaki redak imati jednak broj polja. *CSV* format datoteke nije u potpunosti standardiziran.

Osnovna ideja razdvajanja polja teksta zarezom (`,`) je jasna, ali situacija se komplicira kada podaci polja sadrže i zareze ili ugrađene prijelome redaka (kraj retka) koje bi u pravilu trebali izbjegavati. Naime ovisno o implementaciji *CSV*-a, možda se neće obrađivati takvi podaci (s navedenih problematičnih polja) ili se mogu upotrijebiti navodnici za označavanje takvih polja.

Osim standardno definiranog znaka zarez kao separatora (`,`), moguće je koristiti i neki drugi znak, poput primjerice znaka točka zarez (`;`) ili nekog drugog (pr. `:`), sve ovisno o našim potrebama.

U ovom tekstualnom *CSV* formatu se zapisuju mnoge sistemske, ali i druge datoteke na sustavu, poput raznih log datoteka i mnogih tablica. Prvo pogledajmo kako izgleda jedna tablica u programu ***LibreOffice*** ***Calc***, na slici **14.1.**.

**Slika 14.1. Pogled na tablicu s ocjenama u programu LibreOffice Calc**

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Ime | Prezime | Ocjena | | | | |
| 2 | Ana | Anić | 4 | | | | |
| 3 | Marina | Horvat | 4 | | | | |
| 4 | Pero | Perić | 4 | | | | |
| 5 | Luka | Lukić | 2 | | | | |

U ovoj tablici s ocjenama za predmet *Osnove Linuxa* vidimo stupce koji redom predstavljaju: ime, prezime i ocjenu, a ispod njih u svakom novom redu se nalaze ime, prezime i ocjena koju je svaki pojedini učenik dobio za zalaganje na predmetu.

Međutim tablični kalkulatori poput programa ***LibreOffice Calc*** ili ***Microsoft Excel*** osim ovih vidljivih podataka, odnosno uz sâmu tablicu s podacima, mogu zapisivati (i zapisuju) i prateće podatke, poput:

- Fonta teksta te veličine fonta, kao i boje s kojom je obojan svaki red, stupac ili pojedino polje.
- Formula koje koristimo; primjerice za razne izračune, poput prosjeka ocjena i slično, te mnoge druge meta podatke.

U konačnici, jednostavna tablica poput ove se pri snimanju u format specifičan za svaki program, primjerice: za ***LibreOffice Calc*** je to format ***ODS***, dok je za ***MS Excel*** to ***XLS*** ili ***XLSX*** format, zbog snimanja i mnogih pratećih odnosno takozvanih meta podataka, pretvara u teže čitljiv format od običnog *CSV*-a.
Ipak, moguće je i iz navedenih programa ***LibreOffice Calc*** i ***MS Excel*** izvesti (snimiti) tablice u običnom *CSV* formatu, ali ćemo pri tome izgubiti sve navedene prateće odnosno meta podatke.

Na kraju će u konačnoj *CSV* datoteci ostati samo osnovna tablica s vidljivim stupcima i redovima, bez formula, fontova, boja i slično. Sada pogledajmo istu tablicu u ***CSV*** (tekstualnom) formatu:

```
Ime,Prezime,Ocjena
Ana,Anić,4
Marina,Horvat,4
Pero,Perić,4
Luka,Lukić,2
```

Važno je razumjeti kako je sa cijelim nîzom programa koje standardni imamo dostupne u Linuxu, ekstremno lakše raditi upravo s ovakvim *CSV* tekstualnim formatom, pa ga je moguće pretraživati, filtrirati, mijenjati ili raditi bilo što drugo.

Samo neke od sistemskih Linux datoteka koje koriste ***CSV*** format su:

- `/etc/passwd` – sadrži informacije o korisničkim računaim na sustavu .
- `/etc/shadow` – sadrži hash vrijednost lozinki za korisničke račune.
- `/etc/group` – sadrži informacije o korisničkim grupama.
- … i mnoge druge, kao i sve log datoteke.



***CSV*** format datoteka je definiran u **RFC4180**.



Za primjere rada s ***CSV*** datotekama, pogledajte slijedeća poglavlja:
**5.7.1. Naredba *grep*.**
**5.7.2. Naredba *cut*.**
**5.7.3. Naredba *awk*.**
**5.7.7. Sortiranje sadržaja: naredba *sort*.**

**Izvori informacija: (975), RFC4180.**

# 5. Osnove rada u Linuxu

U ovom poglavlju upoznat ćemo se s nekim od osnovnih naredbi za rad u Linuxu (i *Unixu*)

## 5.1. Rad s nekim od osnovnih Linux/Unix naredbi

U ovoj cjelini upoznati ćemo se s nekim od osnovnih naredbi u Linuxu, kako slijedi:

- `cd` – za promjenu direktorija (mape) (engl. *Change the working directory*).
- `ls` – za ispis direktorija i datoteka (engl. *List directory contents*).
- `pwd` – za ispis trenutnog direktorija u kojem se nalazimo (engl. *Print Working Directory*).
- `date` – za ispis datuma i sata (engl. *Date*).
- `cal` – za ispis kalendara (engl. *Calendar*).
- `man` – za ispis uputa (engl. *Manual*) za neku naredbu (primjerice: `man date`) [*objašnjeno u zasebnom poglavlju*].

Pogledajmo i primjere njihove upotrebe, kroz koje ćemo ih naučiti koristiti. Trenutno imamo slijedeću strukturu direktorija i poddirektorija koja nas zanima: `/home/ucenik/` dakle imamo vršni direktorij `/home` i unutar njega direktorij `ucenik`.

**1.** Uđimo u direktorij (mapu) `/home`, a potom u poddirektorij `/home/ucenik` s naredbom `cd` na sljedeći način:

```
cd /home

cd ucenik
```

**1.1** Vratimo se iz direktorija `/home/ucenik/` u `/home/`. Dakle jedan direktorij iznad odnosno jedan direktorij unazad. Pri tome nas upravo naredba `cd ..` vraća jedan direktorij unazad (`..` preko *inode* broja označava direktorij iznad/unazad).

```
cd ..
```

**2.** Pogledajmo gdje se trenutno nalazimo u stablu direktorija (mapa), s naredbom `pwd`

```
pwd

/home
```

Vidimo da se nalazimo u direktoriju (mapi): `/home`.

**3.** Izlistajmo sadržaj direktorija (mape) s naredbom `ls` te sa njenim prekidačem `-al` na sljedeći način:

```
ls -al

total 12
drwxr-xr-x.  3 root   root   4096 Jul 22 21:30 .
dr-xr-xr-x. 21 root   root   4096 Jul 24 16:50 ..
drwx------   6 ucenik ucenik 4096 Jul 24 18:49 ucenik
```

**4.** Provjerimo trenutni datum i vrijeme (naredba `date`) koji je postavljen na sustavu i koristi se na računalu na kojem radimo:

```
date

Thu Jul 24 19:03:49 CEST 2014
```

**5.** Ispišimo kalendar za tekući mjesec (naredba `cal`) na sljedeći način:

```
cal

      July 2014
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 ''24''  25 26
27 28 29 30 31
```

**6.** Ispišimo upute (manual [`man`]) za naredbu `cal` koju smo malo prije pozvali odnosno pokrenuli:

```
man cal

CAL(1)                    BSD General Commands Manual                   CAL(1)

NAME
     cal - displays a calendar

SYNOPSIS
     cal [-smjy13] [[[day] month] year]

DESCRIPTION
Cal displays a simple calendar.  If arguments are not specified, the current month is displayed.
```

**Izvori informacija: (K-12), `man cd`,`man ls`,`man pwd`,`man date`,`man cal`,`man man`.**

# 5.2. Naredbe za rad s direktorijima i datotekama

U radu s naredbama koje slijede malo ćemo se dublje upoznati s već naučenim naredbama te ćemo naučiti i neke nove.

## 5.2.1. Naredbe: *cd*, *ls*, *pwd*, *mkdir*, *cp*, *mv* i *rmdir*

U ovom poglavlju upoznati ćemo se s naredbama za rad s mapama (direktorijima) i datotekama:

- `cd` – za promjenu trenutnog radnog direktorija odnosno mape (engl. *Change directory*).
- `cp` – za kopiranje datoteke ili direktorija odnosno mape (engl. *Copy*) [***za primjere pogledajte poglavlje 5.5.***].
- `ls` – za izlistanje sadržaja trenutnog (radnog) direktorija odnosno mape (engl. *List directory contents*).
- `pwd` – za prikaz direktorija (mape) u kojoj se trenutno nalazimo (engl. *Print working directory*).
- `mkdir IME` – za kreiranje direktorija (mape) imena `IME` (engl. *Make directory*).
- `mv IME1 IME2` – za prebacivanje direktorija (mape) ili datoteku `IME1` u `IME2` (engl. *Move*).
- `rmdir IME` – za brisanje direktorija (mape) imena `IME` (engl. *Remove directory*), uz napomene:
  - Ne može se brisati direktorij (mapa) u kojem se nalazimo.
  - Ne može se brisati direktorij (mapa) koji je pun.
    - Za rekurzivno brisanje punih direktorija (mapa) koristiti `rm -rf IME`.

Navedene naredbe dolaze u softverskom (*RPM*) paketu imena: ***coreutils***.

Za dublje upoznavanje s gore navedenim naredbama, proći ćemo kroz primjere.
Kreirajmo direktorij (mapu) sa poddirektorijem (podmapom) u njemu, sve u jednom koraku (`/root/test/test1`)

```
mkdir -p /root/test/test1
```

Prekidač `-p` je naredbi `mkdir` naznačio da je osim glavnog direktorija (`/root/test`) potrebno kreirati i poddirektorij imena `test1` unutar njega.
Kreirajmo datoteku imena `proba.txt` unutar direktorija `/root/test/test1/` upotrebom naredbe `touch`:

```
touch /root/test/test1/proba.txt
```

Probajmo sada ući u direktorij `/root/test`

```
cd /root/test
```

Moguće je i relativno kretanje po direktorijima. To znači da ćemo iz trenutnog direktorija `/root/test/` ući u direktorij `/root/test/test1`, ali ne dajući apsolutnu putanju do tog direktorija (`/root/test/test1`) već relativnu. Relativna putanja znači putanja u odnosu na našu trenutnu lokaciju (jer se trenutno nalazimo u direktoriju (`/root/test/`):

```
cd test1
```

Provjerimo gdje se sada nalazimo u stablu direktorija, pomoću naredbe `pwd`

```
pwd

/root/test/test1
```

Vratimo se jedan direktorij unazad. Za ovo se koristi naredba `cd`. Pri tome `..` označavaju direktorij iznad trenutnog.

```
cd ..
```

Pogledajmo gdje se sada nalazimo, ponovno pomoću naredbe `pwd`

```
pwd

/root/test
```

Izlistajmo sadržaj trenutnog direktorija s naredbom `ls` i njenim prekidačem `-al` koji nam daje detaljan ispis.

- `-a` – daje nam sve detalje i skrivene datoteke (ili direktorije).
- `-l` – daje nam listu dodatnih informacija: vrijeme, veličina te ovlasti datoteka i direktorija.

```
ls -al

drwxr-xr-x   3 root root 4096 Jun  1 13:34 .
dr-xr-x---. 10 root root 4096 Jun  1 13:27 ..
-rw-r--r--   1 root root   16 Jun  1 13:29 lista.txt
-rw-r--r--   1 root root   14 Jun  1 13:30 popis-polaznika.txt
drwxr-xr-x   2 root root 4096 Jun  1 13:34 test1
```

Filtrirat ćemo samo zadnji desni stupac s imenima datoteka iz ispisa od gore:

| **Ime datoteke ili direktorija** | **Opis** |
|---|---|
| `.` | `.` označava trenutni direktorij (mapu). |
| `..` | `..` označava jedan direktorij ispred/iznad (prije ovoga u kojem se nalazimo) |
| `lista.txt` | Ovo je datoteka imena: `lista.txt` |
| `popis-polaznika.txt` | Ovo je datoteka imena: `popis-polaznika.txt` |
| `test1` | Ovo je direktorij (mapa) imena: `test1` |

Uđimo u poddirektorij `/root/test/test1`, ako se trenutno nalazimo u direktoriju `/root/test/`:

```
cd test1
```

Sada ćemo izlistati sadržaj direktorija ispred, ali korištenjem dvije točke odnosno `..` koje označavaju jedan direktorij ispred:

```
ls -al ../

total 20
drwxr-xr-x   3 root root 4096 Jun  1 13:34 .
dr-xr-x---. 10 root root 4096 Jun  1 13:27 ..
-rw-r--r--   1 root root   16 Jun  1 13:29 lista.txt
-rw-r--r--   1 root root   14 Jun  1 13:30 popis-polaznika.txt
drwxr-xr-x   2 root root 4096 Jun  1 13:34 test1
```

Na isti način (radom s relativnom putanjom) možemo izlistati i još jedan direktorij iznad odnosno ispred njega:

```
ls -al ../..

dr-xr-x---. 10 root root  4096 Jun  1 13:27 .
dr-xr-xr-x. 23 root root  4096 Jun  1 13:26 ..
-rw-------.  1 root root  1094 Nov  2  2014 anaconda-ks.cfg
-rw-------.  1 root root 22008 Jun  1 13:28 .bash_history
-rw-r--r--.  1 root root    18 May 20  2009 .bash_logout
-rw-r--r--.  1 root root   176 May 20  2009 .bash_profile
-rw-r--r--.  1 root root   207 Jan 30 10:37 .bashrc
-rw-r--r--.  1 root root   100 Sep 23  2004 .cshrc
-rw-r--r--   1 root root     0 Mar 21 16:16 file.sh
```

Isto smo mogli napraviti i s upotrebom apsolutne putanje, poput: `/root/`

```
ls -al /root/
```

Vratimo se u direktorij `/root/test/`

```
cd /root/test
```

Izlistajmo sadržaj direktorija, ali osnovno samo s naredbom `ls` bez prekidača, na sljedeći način:

```
ls

lista.txt  popis-polaznika.txt  test1
```

Vidimo listu i datoteka i direktorija (mapa), sve popisane bez detaljnih informacija o njima.
A sada probajmo vidjeti listu datoteka, ali sa svakim unosom u novom redu, upotrebom prekidača `-1` na sljedeći način:

```
ls -1

lista.txt
popis-polaznika.txt
test1
```

Izlistajmo **samo** datoteke i direktorije (`-l` prekidač naredbe `ls` daje nam detaljno izlistanje).

```
ls -l

total 12
-rw-r--r-- 1 root root   16 Jun  1 13:29 lista.txt
-rw-r--r-- 1 root root   14 Jun  1 13:30 popis-polaznika.txt
drwxr-xr-x 2 root root 4096 Jun  1 13:34 test1
```

Izlistajmo ponovno sve datoteke i direktorije, ali poredane po zadnjem vremenu njihove promjene.
Za ovu potrebu koristiti ćemo prekidač `-t` (*time*):

```
ls -lt

total 12
-rw-r--r-- 1 root root   19 Jun  1 14:36 popis-polaznika.txt
drwxr-xr-x 2 root root 4096 Jun  1 13:34 test1
-rw-r--r-- 1 root root   16 Jun  1 13:29 lista.txt
```

Vidimo kako je na vrhu datoteka `popis-polaznika.txt` koju smo zadnju mijenjali (snimali). Možemo i koristiti prekidač za unatrag:  `-r` (Engl. *Reverse*) u kombinaciju s vremenom (`-t`).

Tada ćemo dobiti izlistanje prema vremenu promjene, ali počevši od starijih prema novijima (suprotno od prethodnog primjera) jer smo naznačili da želimo ispis unatrag:

```
ls -ltr
total 12
-rw-r--r-- 1 root root   16 Jun  1 13:29 lista.txt
drwxr-xr-x 2 root root 4096 Jun  1 13:34 test1
-rw-r--r-- 1 root root   19 Jun  1 14:36 popis-polaznika.txt
```

Sada je pri vrhu datoteka koja je zadnja mijenjana.

Možemo koristiti i rekurzivni ispis odnosno ispisati sve datoteke i direktorije unutar postojećeg direktorija i svih direktorija unutar njih, koliko god ih bilo u dubinu. Prekidač za rekurzivan ispis je veliko slovo **R** odnosno `-R`, pogledajmo kako:

```
ls -lR

-rw-r--r-- 1 root root   16 Jun  1 13:29 lista.txt
-rw-r--r-- 1 root root   19 Jun  1 14:36 popis-polaznika.txt
drwxr-xr-x 2 root root 4096 Jun  1 13:34 test1

./test1:
-rw-r--r-- 1 root root 0 Jun  1 13:34 proba.txt
```

Ovdje vidimo sadržaj trenutnog direktorija te direktorija unutar njega (`test1`). Naredba `ls` ima još cijeli nîz prekidača koje ćemo koristiti u naprednim poglavljima, a sada ćemo ih samo spomenuti. Naredba `ls` i neki od njenih prekidača su:

- `-a` – ispisuje i skrivene datoteke.
- `-h` – daje nam ispis veličine datoteka u KB, MB ili GB umjesto u bajtima (engl. *Human readable*).
- `-i` – daje nam i podatak o *i-nodeu* (Pogledajte poglavlje: **4.5. Datotečni sustav detaljnije**).
- `-l` – daje nam detaljan ispis (ovlasti, vrijeme promjene, vlasnika, grupu, …).
- `-n` – dobivamo numerički ispis vlasnika i grupe umjesto imena (tj. UID i GID).
- `-r` – reverzno odnosno unatrag (daje obrnuti redoslijed ispisa prethodnih prekidača).
- `-R` – dobivamo rekurzivan ispis svih datoteka i direktorija unutar postojećeg direktorija.
- `-t` – dobivamo ispis prema vremenu promjene datoteke/direktorija.
- `-s` – ispiše stvarno zauzeće datoteke na disku (koristiti uz `-h`). Ovo vrijedi samo za takozvane ***sparse*** datoteke.
- `-S` – *(veliko slovo s)* - sortira prema veličini datoteke.
- `-X` – *(veliko slovo x)* - sortira prema ekstenziji datoteke.

Promijenimo ime datoteke `lista.txt` u `nova-lista.txt` na slijedeći način (upotrebom naredbe `mv`):

```
mv lista.txt nova-lista.txt
```

Sada imamo sljedeće stanje:

```
ls -l

-rw-r--r-- 1 root root   16 Jun  1 13:29 nova-lista.txt
-rw-r--r-- 1 root root   19 Jun  1 14:36 popis-polaznika.txt
drwxr-xr-x 2 root root 4096 Jun  1 16:47 test1
```

Promijenimo ime direktorija (mape) `test1` unutar kojeg se nalaze datoteke, u novi direktorij: `novi-test`

```
mv test1 novi-test
```

Pogledajmo ih sada:

```
ls -l

-rw-r--r-- 1 root root   16 Jun  1 13:29 nova-lista.txt
drwxr-xr-x 2 root root 4096 Jun  1 16:47 novi-test
-rw-r--r-- 1 root root   19 Jun  1 14:36 popis-polaznika.txt
```

Sada prebacimo datoteku `nova-lista.txt` iz našeg trenutnog (radnog) direktorija u direktorij koji smo malo prije preimenovali u `novi-test`. To ćemo napraviti na sljedeći način:

```
mv nova-lista.txt novi-test
```

 Naredba `mv` tijekom prebacivanja (ili kopiranja) čuva postavljene ovlasti nad datotekama i direktorijima (mapama).

I sada je vidljivo da datoteke `nova-lista.txt` više nema u trenutnom direktoriju, pozivanjem naredbe `ls -l`.

```
drwxr-xr-x 2 root root 4096 Jun  1 17:28 novi-test
-rw-r--r-- 1 root root   19 Jun  1 14:36 popis-polaznika.txt
```

Već se ona nalazi u poddirektoriju `novi-test`, stoga ispišimo sadržaj ovog direktorija:

```
ls -l novi-test/

-rw-r--r-- 1 root root      16 Jun  1 13:29 nova-lista.txt
-rw-r--r-- 1 root root       0 Jun  1 13:34 proba.txt
```

Rekurzivno obrišimo (! oprez!) direktorij `novi-test` i sve datoteke u njemu kao i sve njegove poddirektorije i datoteke u njima (ako ih ima). Prekidači koje koristimo su:

- `-r` – rekurzivno brisanje svôg sadržaja (svih datoteka i direktorija) počevši od navedenog pa sve unutar njega.
- `-f` – odnosno nasilno brisanje, bez pitanja i upozorenja (engl. *force*).

```
rm -rf novi-test
```

U navedenom primjeru bit će obrisano sve unutar direktorija `novi-test`, koliko god se u njemu nalazilo datoteka ili poddirektorija, jer smo koristili prekidač `-r`. Dakle briše se cijela grana direktorija (mapa) i svih datoteka u njima, počevši od one vršno navedene.

 U slučaju kada neki program (aplikacija) drži otvorenu datoteku, a mi ju želimo obrisati, to neće biti moguće sve dok ne zatvorimo taj program. Da biste pronašli program koji drži otvorenu određenu datoteku, proučite naredbu `lsof`.



Vezano za naredbu `lsof` proučite poglavlje:
**4.5.5.1. File deskriptori i naredba *lsof*.**
Što se tiče detalja oko naredbi: `cp`, `rm` i `mv`, pogledajte poglavlje:
**5.3.1. Napredno kreiranje i brisanje (sadržaja) datoteka.**

**Izvori informacija**: **(K-12), `man cd`,`man ls`,`man pwd`,`man mkdir`,`man mv`,`man rmdir`.**

## 5.3. Kreiranje datoteka

Za kreiranje datoteka, možemo koristiti slijedeće naredbe:

- `touch IME` – za kreiranje prazne datoteke imena `IME` (engl. *Touch*(dotakni/kreiraj)) .
- `cat > IME` – za kreiranje datoteke imena `IME` (upisati sadržaj; `CTRL d` za kraj i snimanje) (engl. *Concatenate*).
- `vi IME` – za kreiranje datoteke imena `IME` i uređivanje iste s `vi` uređivačem teksta (engl. *Visual editor*).
- `echo > IME` – za kreiranje prazne datoteke imena `IME` (engl. *Echo*(jeka)).
- `dd` – koristi se i za druge namjene, ali je s njom moguće i kreirati datoteke sa željenim sadržajem (engl. *Disk dump/Convert and copy*).

**Slijede primjeri**
**1.** Kreirajmo praznu datoteku imena `prazno1.txt` na sljedeći način, s naredbom `touch`:
```
touch prazno1.txt
```

**2.** Kreirajmo drugu praznu datoteku `prazno2.txt`. Nakon pozivanja ove metode kreiranja datoteke za kraj stisnuti `CTRL d`
```
cat > prazno2.txt
CTRL d
```

**3.** Kreirajmo datoteku imena `prazno3.txt` pomoću `vi` uređivača teksta. O `vi` uređivaču teksta, više u poglavlju: **5.6.**
```
vi prazno3.txt
```

**4.** Kreirajmo praznu datoteku imena `prazno4.txt` upotrebom naredbe `echo` upotrebom redirekcije (Pogledajte poglavlje: **5.10.**)
```
echo > prazno4.txt
```

**5.** Upišimo tekst "*TEST*" u datoteku `prazno1.txt`, pomoću naredbe `echo`, također upotrebom jednostruke redirekcije `>`
```
echo "TEST" > prazno1.txt
```



*Naredba `dd` se može koristiti i za kreiranje datoteka određene veličine, kao i datoteka popunjenih sa željenim sadržajem (podacima) ili za druge posebne namjene!.*



Vezano za naredbu **`dd`**, pogledajte sljedeća poglavlja:
**9.4.2 Named pipe.**
**4.6 Prorijeđene (*Sparse*) datoteke.**

Novu datoteku možemo kreirati i primjerice s programom `vi`, primjerice:
```
vi datoteka.txt
```



Vezano za program **`vi`**, pogledajte poglavlje:
**5.6. Uređivač teksta *vi*.**

**Izvori informacija: (K-12), `man touch`,`man cat`,`man vi`,`man echo`,`man dd`,`man vi`.**

### 5.3.1. Napredno kreiranje i brisanje (sadržaja) datoteka

U ovoj cjelini upoznat ćemo se s naprednim načinima kreiranja i/ili pražnjenja sadržaja datoteka.

## Kreiranje datoteka (nastavak na prethodno poglavlje)

Krenut ćemo s kreiranjem velikih datoteka pomoću nekoliko naredbi, kako bi uočili razlike.

**Naredba fallocate**

S naredbom `fallocate` moguće je iznimno brzo kreirati velike datoteke, stoga što ova naredba koristi sistemski poziv (*fallocate*). Kreirajmo jednu ovakvu datoteku veličine 1GB:

```
fallocate -l 1G large-file
```

Pogledajmo sada ovu datoteku s dodatnim pogledom na njenu Sparse veličinu:

```
ls -alhs large-file

1.0G -rw-r--r-- 1 root root 1.0G Feb  6 09:36 large-file
```

← Vidimo da je njena veličina prijavljena kao 1GB te da na disku zauzima također `1.0G`, što znači kako ona nije takozvana prorijeđena odnosno *sparse* datoteka.

**Naredba truncate**

Naredba `truncate` u Linuxu omogućuje povećanje (ili smanjivanje) datoteka na određenu veličinu.
Ova naredba koristi sistemski poziv (*truncate*) s kojim je moguće smanjiti ili povećati datoteku na željenu veličinu iznimno brzo. Kreirajmo datoteku veličine 100MB:

```
truncate -s 100M large-file
```

Sada pogledajmo njenu veličinu:

```
ls -alhs large-file

0 -rw-r--r-- 1 root root 100M Feb  6 10:19 large-file
```

← Vidimo da je ovo *sparse* datoteka, jer iako je prividne veličine 100MB, na disku nije alociran prostor za njen sadržaj (prva oznaka je `0`).

**Naredba dd**

Naredba `dd` (engl. *disk dump*) radi tako da se podaci s ulaza u naredbu (što mogu predstavljati i posebne datoteke) koriste za zapisivanje u izlaznu datoteku, u blokovima podataka. Ona podržava mnoge načine i metode takvog rada.
U prvom primjeru kao ulaz ćemo koristiti posebnu datoteku koja je generator nula (`/dev/zero`), a dodatno ćemo definirati veličinu blokova podataka (`bs`) u koji će se zapisivati, uz definiran broj ponavljanja (`count`).
Pri tome će izlazna datoteka biti veličine 100MB, i kreirat će se blok po blok (konkretno po 1MB)

```
dd if=/dev/zero of=large-file bs=1M count=100
```

Ovakav način kreiranja datoteke također ne kreira sparse datoteke:

```
ls -alhs large-file

100M -rw-r--r-- 1 root root 100M Feb  6 09:57 large-file
```

← Vidimo da su obije veličine 100MB

Sada ćemo isto upotrebom naredbe `dd` kreirati *sparse* datoteke od 100MB:

```
dd if=/dev/zero of=large-file bs=1M count=0 seek=100
```

← ovdje smo koristili (`seek`) koji skače na krajnju poziciju (100 blokova od 1MB) .

Isto smo mogli napraviti i ovako:

```
dd if=/dev/zero of=large-file bs=1 count=0 seek=100M
```

Pogledajmo ovu datoteku:

```
ls -alhs large-file

0 -rw-r--r-- 1 root root 100M Feb  6 09:59 large-file
```

← Vidimo da je ovo *sparse* datoteka, jer iako je prividne veličine 100MB, na disku nije alociran prostor za njen sadržaj (prva oznaka s lijeve strane je `0`).
U svakom slučaju kreiranje ovakve *sparse* datoteke je iznimno brzo.

**Naredba seq**

Naredba `seq` inače se koristi za ispisivanje niza brojeva, ali se može koristiti za preusmjeravanje (*redirekciju*) tog niza brojeva u datoteku, ali vrlo sporo. Kreirajmo niz od 13.107.200 brojeva (oko 102MB)

```
seq 13107200 > large-file
```

← ova operacija će potrajati jer je potrebno neko vrijeme da se kreira ovoliki niz brojeva i zapiše u datoteku.



Vezano za sistemske pozive, pogledajte poglavlja:
**9.2.1. Napredno o procesima i signalima.**
**11.1. Kernel.**

## Pražnjenje (brisanje) sadržaja datoteka

Prvi najjednostavniji način pražnjenja (sadržaja) datoteke je koristeći naredbeni redak tako da preusmjerimo tzv. *null* (nepostojeći objekat) u datoteku, kao u nastavku:

```
> large-file
```

**Upotreba naredbi cat, cp i dd**

Prisjetimo se kako posebna datoteka `/dev/null` zapravo uklanja (briše) svaki ulaz u nju (ili izlaz iz nje).
U tu svrhu možemo ju iskoristiti tako da njen izlaz preusmjerimo u datoteku, što možemo napraviti:

```
cat /dev/null > large-file
```

Ili čak kopiranjem njenog sadržaja u datoteku:

```
cp /dev/null large-file
```

Ili korištenjem nje kao ulaza u naredbu dd te izlazom u željenu datoteku čiji sadržaj želimo obrisati::

```
dd if=/dev/null of=large-file
```

**Upotreba naredbe echo**

Naredba `echo` se koristi za ispis sadržaja (teksta, vrijednosti varijabli i slično) na terminal, ali i nju možemo koristiti uz preusmjeravanje (redirekciju), za brisanje sadržaja datoteke:

```
echo "" > large-file
```

Ili direktnom *redirekcijom*:

```
echo > large-file
```

Gornja dva primjerna u odredišnu datoteku će preusmjeravanjem (*redirekcijom*) dodati prazan redak i to sa znakom za kraj retka (heksadecimalno `0A` – tzv. *Line Feed*).
Stoga pogledajmo ovu datoteku nakon ovakvog pražnjenja njenog sadržaja, pomoću heksadecimalnog preglednika odnosno pomoću naredbe `hexdump`, filtrirano s naredbom `head`. Dakle unutar ovakve datoteke biti će vidljivo sljedeće:

```
hexdump large-file | head
0000000 000a
0000001
```

← Jasno je vidljivo da je u prvi redak datoteke dodana heksadecimalna oznaka `0A`.

Dodano je moguće naložiti naredbi `echo` da šalje prazan izlaz (bez oznake za novi redak):

```
echo -n "" > large-file
```

Ako ponovno pogledajmo ovu datoteku nakon ovakvog pražnjenja njenog sadržaja, pomoću heksadecimalnog preglednika odnosno pomoću naredbe `hexdump`, unutar ove datoteke sada će biti vidljivo samo sljedeće:

```
hexdump large-file | head
0000000
```

← Naime ovdje je sada vidljivo da je prvi redak datoteke potpuno prazan.

**Upotreba naredbe truncate**

Naredba `truncate` u Linuxu omogućuje smanjenje (ili povećanje) datoteka na određenu veličinu.
Ova naredba koristi sistemski poziv (*truncate*) s kojim je moguće smanjiti ili povećati datoteku na željenu veličinu iznimno brzo.
Ako koristite *truncate* s izlaznom veličinom 0 (`-s0`), možete potpuno isprazniti datoteku, kao u primjeru:

```
truncate -s 0 > large-file
```



Vezano za prorijeđene (sparse) datoteke, pogledajte poglavlje:
**4.6 Prorijeđene (*Sparse*) datoteke.**

**Izvori informacija:** `man 2 fallocate`, `man fallocate`, `man 2 truncate`, `man truncate`, `man dd`, `man seq`, `man hexdump`, `man head`.

# 5.4. Naredbe za izlistanje sadržaja datoteka

U ovoj cjelini proći ćemo kroz primjere upotrebe naredbi za izlistanje odnosno prikaz sadržaja datoteka.

Pri tome ćemo se upoznati sa slijedećim naredbama:

- `cat IME` – za prikaz sadržaja datoteke (*stdout*) ili povezivanje više datoteka u jednu (engl. *Concatenate*).
- `more IME` – za prikaz sadržaja datoteke na standardni izlaz, ekran po ekran (engl. *More* (više)).
- `less IME` – naprednija inačica naredbe `more`, s više mogućnosti i opcija (engl. *Less*(manje)).
- `head IME` – za prikaz samo početka sadržaja datoteke (*stdout*) (engl. *Head*(glava/zagljavlje/početak)).
- `tail IME` – za prikaz samo kraja sadržaja datoteke (*stdout*) (engl. *Tail*(rep/kraj)).

**Pogledajmo nekoliko primjera**

**1.** Ispišimo sadržaj datoteke `prazno1.txt` (iz prethodnog poglavlja), pomoću naredbe `cat` na sljedeći način:

```
cat prazno1.txt
TEST
```

**1.1.** Spojimo više datoteka (`1.txt`, `2.txt`, `3.txt`) u jednu `izlaz.txt`, tako da ona sadrži sve što i navedene tri datoteke i to sadržajno tako da će se u nju prvo zapisati sadržaj prve datoteke, a potom druge pa treće ili koliko ih već navedemo:

```
cat 1.txt 2.txt 3.txt > izlaz.txt
```

**2.** Ispišimo sadržaj datoteke (`/etc/profile`) pomoću naredbi `more` i `less`. Njihova puna funkcionalnost vidljiva je tek kod ispisa dužeg teksta jer tek tada možemo koristiti njene pune mogućnosti (pretraživanje, skakanje na početak/kraj i sl.).

```
more /etc/profile
```



Naredba `more` prikazuje tekst, koji je moguće skrolati (gore/dolje) s kursorskim tipkama. U slučaju izlistanja dužeg teksta (na više stranica) za izlazak iz ispisa se koristi tipka odnosno slovo `Q`.

Sada pogledajmo ispis sadržaja datoteke (`/etc/profile`) pomoću naredbe `less`.

```
less /etc/profile
```



Naredba `less` prikazuje tekst, koji je isto moguće skrolati (gore/dolje) ali i pretraživati (stiskanjem znaka **/** nakon kojeg upisujemo tekst odnosno pojam koji želimo tražiti). Osim toga moguće je skočiti na početak teksta (slovo **g**) ili na kraj (slovo **G**) ali i koristiti i druge napredne funkcije. Izlazak iz programa je također tipka odnosno slovo `Q`

**3.** Ispis pomoću naredbe `head` se koristi kada želimo ispisati samo početak, odnosno samo prvih nekoliko redaka neke datoteke. Stoga ako želimo vidjeti samo prvih nekoliko redaka datoteke `/etc/profile` trebamo napraviti sljedeće:

```
head /etc/profile
```

Pogledajte neke korisne prekidače naredbe `head`:

- `-n` ***x*** - ispiši samo prvih ***x*** redaka datoteke.
- `-v` - (*verbose*) na početku ispisa ispiši i naziv datoteke (korisno ako se otvara više datoteka istovremeno).

**4.** Ispis pomoću naredbe `tail` se koristi kada želimo ispisati samo kraj neke datoteke; isti primjer datoteke `/etc/profile`.

```
tail /etc/profile
```

Moguće je koristiti i prekidač `-f`; poput `tail -f IMEDATOTEKE`, a tada će se prikazana otvorena datoteka (ako ju netko konstantno dopunjava) i prikazati osvježavana s novo dodanim redovima odnosno sadržajem. To je vrlo korisno za gledanje log datoteka koje se stalno dopunjavaju s novim podacima.

Pogledajte korisne prekidače naredbe `tail`:

- `-f` – dopunjavaj ispis uživo kako otvorena datoteka raste (ako ju neki drugi program dopunjava).
- `-n` ***x*** – ispiši samo zadnjih ***x*** redaka datoteke.
- `-c` ***x*** – ispiši samo zadnjih ***x*** bajta datoteke (mogu se koristiti i prefiksi **k** [kB], **M** [MB] i drugi).
- `-v` – (*verbose*) na početku ispisa ispiši i naziv datoteke, što je korisno ako se otvara više datoteka istovremeno.

**Izvori informacija: (K-12), `man cat`, `man more`, `man less`, `man head`, `man tail`.**

## 5.5. Rad s datotekama

U ovoj cjelini proći ćemo primjere rada s datotekama odnosno upoznati se s naredbama, pomoću kojih možemo kopirati, brisati, premještati datoteke iz direktorija u direktorij i slično. Pogledajmo koje su to naredbe.

Naredba `mv IME1 IME2` (engl. *Move*(*makni/prebaci*)) – služi za prebacivanje datoteke iz direktorija u direktorij ili za preimenovanje datoteke ili direktorija (mape):

- `mv ime-datoteke ime-dir` – koristi se za prebacivanje datoteke u željeni direktorij (mapu) (engl. *Move*).
- `mv staro-ime novo-ime` – koristi se za preimenovanje imena datoteke u novo ime datoteke.

Naredba `rm` – koristi se za brisanje datoteka ili direktorija, prema principu (engl. *Remove*(*makni/obriši*)):

- `rm ime-datoteke` – koristi se za brisanje datoteka (ili direktorija).

Naredba `cp` – koristi se za kopiranje datoteka ili direktorija (engl. *Copy*(*kopiraj*)), prema principu `cp` ***`izvor odredište`*** ili:

- `cp izvor1 izvor2 odredište` – pri ovoj operaciji kopiramo više datoteka istovremeno te ih kopiramo u odredišni direktorij.
- `cp -r izvor odredište` – pri ovoj operaciji, rekurzivno kopiramo sve s izvora u odredišni direktorij (`odredište`). **Kod kopiranja možemo koristiti i prekidač `-p` kod potrebe za kopiranjem izvornih [ovlasti](#) i vremenskih oznaka**. Prekidač `-r` koristimo, ako trebamo rekurzivno kopiranje sadržaja (poddirektorija/datoteka).



Navedene naredbe dolaze u softverskom (*RPM*) paketu imena: ***coreutils***.

**Slijede primjeri**.
Izlistajmo prvo direktorije i datoteke u trenutnom direktoriju u kojem se nalazimo:

```
ls -al

drwxrwxr-x 3 ucenik ucenik 4096 Jul 21 21:57 .
drwx------ 6 ucenik ucenik 4096 Jul 21 21:45 ..
-rw-rw-r-- 1 ucenik ucenik    5 Jul 21 21:34 prazno1.txt
-rw-rw-r-- 1 ucenik ucenik    0 Jul 21 21:33 prazno2.txt
-rw-rw-r-- 1 ucenik ucenik    0 Jul 21 21:33 prazno3.txt
-rw-rw-r-- 1 ucenik ucenik    0 Jul 21 21:34 prazno4.txt
-rw-rw-r-- 1 ucenik ucenik    0 Jul 21 21:27 prazno.txt
drwxrwxr-x 3 ucenik ucenik 4096 Jul 21 21:57 test
```

**1.** Prebacimo datoteku `prazno1.txt` u direktorij imena `test` pomoću naredbe `mv`:

```
mv prazno1.txt test
```

**2.** Pogledajmo sada sadržaj direktorija `test` te preimenujmo datoteku `prazno1.txt` u `puno1.txt`

```
ls -al test

drwxrwxr-x 3 ucenik ucenik 4096 Jul 21 22:00 .
drwxrwxr-x 3 ucenik ucenik 4096 Jul 21 22:00 ..
-rw-rw-r-- 1 ucenik ucenik    5 Jul 21 21:34 prazno1.txt
drwxrwxr-x 2 ucenik ucenik 4096 Jul 21 21:57 test1
```

A zatim preabacimo (u ovom slučaju preimenujmo) datoteku `prazno1.txt` u `puno1.txt` pomoću naredbe `mv`:

```
cd test
mv prazno1.txt puno1.txt
```

Sada pogledajmo sadržaj trenutnog direktorija (mape):

```
ls -al

-rw-rw-r-- 1 ucenik ucenik    5 Jul 21 21:34 puno1.txt
drwxrwxr-x 2 ucenik ucenik 4096 Jul 21 21:57 test1
```

**3.** Kopirajmo datoteku `puno1.txt` u novu datoteku imena `prazno1.txt`, pomoću naredbe `cp` na sljedeći način:

```
cp puno1.txt prazno1.txt
```



Kod kopiranja s naredbom **`cp`**, ako moramo zadržati izvorne **[ovlasti](#)** i vremenske oznake, moramo koristiti prekidač **`-p`**.

Ponovno pogledajmo sadržaj direktorija (mape) u kojem s trenutno nalazimo (skratili smo ispis sadržaja direktorija):

```
ls -al

-rw-rw-r-- 1 ucenik ucenik    5 Jul 21 22:08 prazno1.txt
-rw-rw-r-- 1 ucenik ucenik    5 Jul 21 21:34 puno1.txt
drwxrwxr-x 2 ucenik ucenik 4096 Jul 21 21:57 test1
```

**4.** Sada obrišimo datoteku `puno1.txt`, pomoću naredbe `rm` na sljedeći način:

```
rm puno1.txt
```

Pogledajmo ponovno sadržaj direktorija (mape) u kojem s trenutno nalazimo (skratili smo ispis sadržaja direktorija):

```
ls -al

-rw-rw-r-- 1 ucenik ucenik    5 Jul 21 22:08 prazno1.txt
drwxrwxr-x 2 ucenik ucenik 4096 Jul 21 21:57 test1
```

Važno je znati da naredba `rm` uklanja (briše) datoteke bez traženja potvrde te ne postoji način za poništavanje brisanja jer se uklonjene datoteke ne premještaju u smeće. Međutim naredba `rm` ima opciju `-i` (interaktivnu) koja traži potvrdu prije uklanjanja datoteka. Ipak, u slučaju kada datoteka nema postavljenu ovlast zapisivanja (tj. ovlast **w** - *write*) što znači da je zaštićena od zapisivanja (engl. *write protected*), a pokušamo ju obrisati, ona neće automatski biti obrisana.
U slučaju potrebe za uklanjanjem (brisanjem) ovakvih datoteka, možemo koristiti prekidač `-f` (engl. *force*), s kojim će i ovakve (zaštićene) datoteke automatski biti obrisane.

Pogledajte neke korisne prekidače naredbe `rm`:

- `-i` – traži potvrdu za brisanje svake pojedine datoteke.
- `-I` – traži potvrdu za brisanje više od tri datoteke ili rekurzivno brisanje.
- `-r` – brisanje direktorija i njihovog sadržaja (datoteka) rekurzivno.
- `-f` – nasilno brisanje i datoteka koje su zaštićene od zapisivanja (one bez w ovlasti).
- `-v` – kod svakog brisanja ispisuje detalje o brisanju.
- `--` – ako brišemo datoteke čije ime počinje sa znakom minus (`-`) potrebno je prvo navesti ovaj prekidač (*minus minus*).
- `--no-preserve-root` – ako želimo (nasilno) brisati vršni *root* direktorij (**/**) cijelog sustava, onda moramo koristiti ovaj prekidač.

**5.** Nastavljamo s kopiranjem datoteka ili direktorija (mapa) upotrebom naredbe `cp`. Prvo kreirajmo direktorij `/backup`:

```
mkdir /backup
```

Sada kopirajmo datoteku `/etc/passwd` u ovaj direktorij:

```
cp /etc/passwd /backup
```

Kopirajmo više datoteka (`/etc/passwd`, `/etc/group` i `/etc/shadow`) u isti odredišni direktorij (mapu):

```
cp /etc/passwd /etc/group /etc/shadow /backup/
```

Moguće je kopirati i rekurzivno [`-r`] odnosno direktorije i poddirektorije te datoteke unutar njih. Stoga kopirajmo sve datoteke i direktorije te sve poddirektorije unutar vršnog direktorija `/home/hrvoje/` u odredišni direktorij `/backup`.

```
cp -r /home/hrvoje/ /backup/
```

Pogledajmo i druge korisne prekidače naredbe `cp`:

- `-f` – forsiraj kopiranje, bez pitanja o prepisivanju odredišne datoteke.
- `-l` – ne kopiraj datoteku na odredište, već na odredištu kreiraj tvrdu poveznicu (engl. *hard link*) na nju.
- `-p` – tijekom kopiranja, sačuvaj sve ovlasti izvorišne datoteke na odredištu.
- `-P` – *(veliko slovo P)* tijekom kopiranja, nemoj pratiti simboličke veze s izvorišta datoteke na odredištu.
  ***Standardno ponašanje je da se prilikom kopiranja prate simboličke poveznice (engl. symbolic link).***
- `-s` – ne kopiraj datoteku na odredište, već na odredištu kreiraj simboličku vezu (engl. *symbolic link*) na nju.
- `--sparse=always` – tijekom kopiranja, ako se kopiraju ***sparse datoteke***, kao takve ih i kopiraj na odredište.
- `-u` – ne kopiraj datoteku, ako izvorišna datoteka nije novija od odredišne.

Pogledajte i korisne prekidače naredbe `mv`:

- `-i` - traži potvrdu za prebacivanje svake pojedine datoteke.
- `-f` - nasilno prebacivanje i datoteka koje su zaštićene od zapisivanja (one bez ***w*** ovlasti).
- `-n` - kod prebacivanja datoteke u drugi direktorij, ako na odredištu već postoji ista datoteka, ona se tada ne prebacuje.
- `-u` - (*update*) premjestiti datoteku, samo kada je izvorna datoteka novija od odredišne datoteke ili kada odredišna datoteka nedostaje.
- `-b` - (*backup*) ako odredišna datoteka postoji, kreiraj backup datoteku u odredišnom direktoriju.

**Izvori informacija: (K-12), `man mv`, `man cp`, `man rm`, `man ls`.**

## 5.6. Uređivač teksta *vi*

Prije nego krenemo dalje, morat ćemo se upoznati s osnovnim uređivačem odnosno *editorom* teksta. Program `vi` je uređivač teksta i on je inicijalno razvijan za prve inačice *UNIXa,* negdje tijekom 1976. godine, a od tada je dostupan na svim inačicama *Unixa* i *Linuxa* koje postoje i koriste se do danas. On je jedini uređivač teksta koji ćemo sigurno naći na bilo kojem *UNIX* ili *Linux* baziranom operativnom sustavu pa je važno poznavati barem neke njegove osnove. Koliko god da je netko navikao na neki drugi uređivač teksta, u praksi nikada ne možemo biti sigurni koji od njih ćemo zateći na nekom *UNIX* ili *Linux* sustavu, osim `vi` uređivača teksta koji se nalazi na svima njima.

Važno je razumjeti da `vi` uređivač teksta ima dva osnovna načina rada:

- *Takozvani **insert*** način rada za upisivanje sâmog teksta.
- ***Naredbeni*** način rada za izvršavanje naredbi koje želimo primijeniti nad tekstom.


Ne zaboravite da se kod upisa naredbi, kao i cijelog Linuxa razlikuju velika i mala slova!.

Pošto je ovaj program razvijen za rad u naredbenom retku odnosno ljusci, u njemu ne postoji grafičko sučelje iz kojeg možemo mijenjati opcije ili način rada. To znači kako on radi u terminalu, a naziva ga se i uređivačem teksta naredbenog retka. Međutim on nije niti program za obradu i procesiranje teksta poput: *MS Worda*, *Libre Offfice Writera* i slično, već je sličniji programima poput: ***MS Notepad*** ili ***Notepad++.*** Pokretanjem programa `vi`, prvo ulazimo u *naredbeni* način rada, što znači kako u tom trenutku još ne možemo unositi tekst odnosno još uvijek ništa ne možemo pisati, osim unositi njegove naredbe.

**Prvo upoznajmo njegove osnovne naredbe, podijeljene u sljedeće logičke cjeline (**koje se pozivaju **isključivo** iz naredbenog načina rada**) :**
Naredbe za ubacivanje (engl. *insert*) i nadodavanje teksta (engl. *append*) te prikaz teksta:

- `i` – ubaci tekst (ispred) (engl. *Insert*).
- `a` – ubaci tekst iza odnosno na kraj retka (engl. *Append*).
- `o` – *(malo slovo **o**)* - otvori novi redak ispod trenutnog u kojem se nalazimo.
- `O` – *(veliko slovo **O**)* - otvori novi redak iznad onoga u kojem se trenutno nalazimo.
- `I` – ubaci tekst na početak retka (u kojem se trenutno nalazimo).
- `A` – ubaci tekst na kraj retka (u kojem se trenutno nalazimo).
- `:set number` - naznačava programu da prikazuje brojeve redaka (linija), ili da ih ne prikazuje (`set nonumber`).
- tipka `ESC` - vraća nas u naredbeni način rada.

Ako ne rade kursorske tipke, možemo koristiti i njihovu zamjenu, a to su slova koja ih predstavljaju:

- `h` – pomjeranje lijevo.
- `l` – pomjeranje desno.
- `j` – pomjeranje dolje.
- `k` – pomjeranje gore.
- tipka `ESC` – vraća nas u naredbeni način rada.

Naredbe za brisanje slova/znakova upisanog teksta:

- `x` – za brisanje: znak po znak (jedno stiskanje slova/tipke **x** briše jedno slovo).
- `dd` – za brisanje cijelog retka teksta.

Poništavanje uređivanja (engl. *Undo*):

- `u` – *(malo slovo **u**)* - poništi prvu promjenu (vraća se na prvu promjenu prije) u trenutnom redu (liniji).
- `U` – *(veliko slovo **U**)* - poništava sve promjene koje smo radili, unutar trenutnog retka (linije).

Ponavljanje naredbe:

- `.` – *(točka)* - ponavlja zadnju naredbu.

Kopiranje (engl. pojam *Yank*):

- `yw` – kopiraj trenutnu riječ.
- `2yw` – kopiraj dvije (2) riječi.
- `yy` ili `Y` – kopiraj trenutni redak teksta (cijeli redak teksta).
- `2yy` – kopiraj dva (2) retka teksta.

Lijepljenje (zalijepi) (*Engl: Paste*)

- `p` – zalijepi prethodno kopirani tekst (engl. *Paste*).

Traženje teksta, pojma ili ključne riječi unutar teksta odnosno cijelog dokumenta:

- `/`**`pojam`** – traži riječ: **`pojam`**.
- `n` – skoči na sljedeći pronađeni pojam.
- `N` – skoči na prethodni pronađeni pojam.

Pronađi i zamjeni:

- `%s/prvo/drugo/g` – zamjeni riječ/pojam unutar cijelog teksta/dokumenta: `prvo` sa pojmom `drugo`

Kretanje po radnoj datoteci:

- `g` – *(malo slovo G)* - skoči na početak datoteke.
- `50G` – skoči na 50-ti red u datoteci.
- `G` – *(veliko slovo **G**)* - skoči na kraj datoteke.

Snimanje uređenog teksta (moramo doći do naredbenog način rada da nam se pojavi znak `:` u donjem lijevom kutu, sa `ESC` `:`)

- `:w` – snimi radni dokument [*dokument na kojem radimo*] (engl. *Write*).
- `:wq` – snimi radni dokument i izađi iz programa `vi` (engl. *Write* + *Quit*).
- `:wq!` – snimi i izađi iz programa `vi`, čak i ako je otvorena datoteka zaključana za zapisivanje (ako je „*read-only*“).

**Osnovni rad u programu `vi`:**

Iz Linux ljuske otvorimo novu tekstualnu datoteku, koja će se zvati: `dokument.txt`

```
vi dokument.txt
```

Ovim potezom smo pokrenuli program `vi` i s njime stvorili (ako je nije bilo) i otvorili datoteku: `dokument.txt`

Nalazimo se na početku naše novo stvorene datoteke; u njenom prvom redu, na prvoj poziciji vidljivoj kao **~**

```
~
~
~
~
"dokument.txt" [New File]
```

Međutim nalazimo se u naredbenoj načinu rada te sada moramo odabrati što želimo raditi. Pošto želimo početi upisivati tekst, moramo stisnuti slovo to jest tipku `i`, jer želimo unositi tekst [`i`=insert=ubaci tekst (počni s upisivanjem teksta)].

Nakon što smo stisnuli tipku `i`, bit će vidljivo sljedeće:

```

~
~
~
"dokument.txt" [New File]
```

Potom u prvom redu vidimo kursor ( ) i možemo početi upisivati tekst, zalijepiti (*paste*) ga iz nekog drugog dokumenta i slično. Mi ćemo početi pisati, pa upišimo primjerice: `ovo je proba`

```
ovo je proba
~
~
~
"dokument.txt" [New File]
```

Za sada ćemo samo sve snimiti i izaći iz programa `vi`. Kako bismo to napravili, stisnimo tipku `ESC`, a potom stisnimo tipke: `SHIFT` i `Q` te ENTER, a potom će nam se pokazati slijedeći ekran. Ako ne vidimo `:` stisnimo tipku `ESC` pa potom znak `:`

```
ovo je proba
~
~
~
Entering Ex mode.  Type "visual" to go to Normal mode.
:
```

Vidimo kako smo ušli u način rada u kojem možemo zadati željene naredbe programu **vi**, u donjem lijevom kutu.

Potom ćemo upisati: `wq` te stisnuti tipku ENTER, poput:

```
:wq
```

Čim smo stisnuli tipku ENTER, `vi` je snimio tekst u našu datoteku `dokument.txt` i izašao u naredbeni redak (*shell*).

Ponovno učitajmo našu datoteku:

```
vi dokument.txt
```

I na početku se nalazimo u prvom redu, na prvom slovu našeg teksta; u ovom slučaju je to slovo `o`:

```
ovo je proba
~
~
~
"dokument.txt" 1L, 13C
```

Sada, kad se još nalazimo u naredbenom načinu rada, možemo koristiti bilo koju naredbu, od onih koje smo popisali. Mi ćemo sada skočiti na kraj retka, tako što ćemo stisnuti veliko slovo `A`. Sada se dogodilo to da smo skočili na kraj retka, na poziciju

```
ovo je proba
~
~
~
--INSERT --
```

I na kraju dokumenta vidimo kako smo prešli u `--INSERT--` način rada, te sada možemo upisivati željeni tekst. Nakon što smo upisali što smo željeli, ponovno prvo stisnimo tipku `ESC`, a potom stisnimo tipke: `SHIFT` i `Q` te ENTER, a potom snimimo dokument, upisivanjem `wq` i stiskanjem tipke ENTER. Toliko o osnovama programa `vi`.



Standardni uređivač teksta u Linuxu se definira u sistemskoj varijabli **`VISUAL`**.

Sistemsku varijablu `VISUAL` možemo postaviti na **vi** uređivač teksta na sljedeći način:

```
export VISUAL=/usr/bin/vi
```



Pogledajte i poglavlje*:*
*6.2.2.* **Sistemske (*Environment*) varijable i postavke terminala.**



**Pogledajte (za PDF dokument) i kratku animaciju rada u programu `vi` na ovoj poveznici.**

**Izvor informacija: (K-12), `man vi`.**

# 5.7. Rad sa sadržajem datoteka

U ovom poglavlju upoznat ćemo se s radom sa sadržajem datoteka. I to u radu sa sadržajem datoteka tekstualnog formata.

Pri ovim radnjama najčešće koristimo nekoliko osnovnih programa odnosno naredbi:

- `grep` – pronalazi traženi pojam odnosno ključnu riječ unutar željene datoteke.
- `cut` – pronalazi tražene dijelove teksta unutar datoteke [pr. polje podataka, stupac ili sl.] (engl. *Cut*(odreži/izreži)).
- `awk` – programski jezik i naredba za pronalaženje, obradu i rad s uzorcima teksta.
- `sed` – koristi se za razne transformacije teksta (engl. *Stream Line Editor*).

Osim navedenih, postoji i veći broj pomoćnih programa odnosno naredbi, od kojih ćemo spomenuti:

- `wc` – koristi se za prebrojavanje riječi u datoteci (engl. *Word Count*(*prebroji riječi*)).
- `uniq` – za izbacivanje riječi odnosno redova koji su identični tj. ponavljaju se (engl. *Uniq*(jedinstven)).
- `sort` – za sortiranje riječi po abecedi, broju ili sl. (engl. *Sort*(*sortiraj*)) odnosno naredba `tsort` za topološko sortiranje

S navedenim naredbama upoznat ćemo se u cjelinama koje slijede!

**Izvori informacija: (K-12), `man grep`, `man cut`, `man awk`, `man sed`, `man wc`, `man uniq`, `man sort`.**

### 5.7.1. Naredba *grep*

Naredba `grep` je nastala u vrijeme nastanka prvih UNIX-a (1973.g.). Koristi se za traženje teksta odnosno pojma ili riječi, unutar tekstualnih podataka ili datoteka. Pretraživanje `grep` odrađuje pregledavajući redak po redak u datoteci unutar koje se radi pretraživanje. Standardno `grep` prikazuje sve retke (linije) u kojima je pronađen traženi tekst odnosno traženi pojam ili riječ. Postoji nekoliko važnijih varijanti naredbe `grep` od kojih je svaka razvijena za svoju specifičnu namjenu:

- `grep` – ovo je standardna naredba odnosno program.
- `egrep` – ovo je nadogradnja na osnovnu naredbu, koji dodaje nove meta karaktere za regularne izraze: **+**, **?**, **|**, **(** , **)** a njegova funkcionalnost se može pozvati i sa `grep -E`.
- `fgrep` – je varijanta programa optimizirana za ekstremno brzo pretraživanje, ali ne podržava regularne izraze. Može se pozvati i sa `grep -F`.

**Česti prekidači naredbe** `grep` su:

- `-c` – ispiši samo broj pronađenih pojmova (broj ponavljanja pojma).
- `-i` – ignoriraj veliko ili malo slovo (pretraži i jednu i drugu varijantu).
- `-n` – ispiši i broj reda (linije teksta) u kojem je pronađen traženi pojam.
- `-r` – rekurzivno pretraživanje svih direktorija i poddirektorija ispod trenutnog, te naravno datoteka unutar njih.
- `-l` – samo ispiši datoteku u kojoj je pronađen traženi pojam.
- `-m` – nakon koliko pronađenih istih pojmova se prekida pretraživanje. Primjerice upotrebom prekidača `-m1` će se već nakon prvog pronađenog pojma prestati s daljim pretraživanjem.
- `-o` – ispiši samo traženi pojam koji je pronađen.
- `-v` – ispiši sve one retke (linije) koje ne sadrže pojam. Ovo je tzv. *inverzna* pretraga.

Dodatni operatori za ispis su:

- `-A` **n** – uz pronađeni pojam ispiši još **n** redaka nakon onoga koji je pronađen.
- `-B` **n** – uz pronađeni pojam ispiši još **n** redaka prije onoga koji je pronađen.

**Primjeri**: Iz tabličnog kalkulatora (*Open Office Calc* ili *Microsoft Excell*) smo izvezli (eksportirali) tablicu s ocjenama studenata za predmete: *Uvod u Linux* (`uvod-u-linux.csv`) i Linux Napredno(`linux-napredno.csv`) i to u ***CSV*** formatu.
Pri tome postoje tri stupca: `Ime`, `Prezime` i `Ocjena`, odvojena sa znakom `;`. Dakle znak `;` predstavlja granicu stupca.
Tablice trenutno izgledaju ovako (ako ih gledamo s naredbom `cat`) na sljedeći način:

```
cat uvod-u-linux.csv

Ime;Prezime;Ocjena
Ana;Anic;4
Marina;Horvat;5
Luka;Lukic;2
Pero;Peric;4
Kristijan;Kristic;5
Maja;Anic;2
```

Dok druga tablica izgleda ovako:

```
cat linux-napredno.csv

Ime;Prezime;Ocjena
Ana;Anic;2
Marina;Horvat;4
Luka;Lukic;2
Pero;Peric;4
Kristijan;Kristic;3
Maja;Anic;2
```

**1.** Pomoću naredbe `grep` unutar datoteke `uvod-u-linux.csv` pronađimo cijeli unos (red) od studenta imena `Ana`

```
grep Ana uvod-u-linux.csv

Ana;Anic;4
```

**1.1.** Ispišimo i broj reda u kojem je pronađen student imena `Ana` na sljedeći način (`-n` = ispisuje i broj reda):

```
grep -n Ana uvod-u-linux.csv

2:Ana;Anic;4
```

**2.** Sada nas zanima i kako izgleda unos u obje tablice (datoteke), za istu osobu (`Ana`). Ovdje ćemo koristiti regularni izraz `*` koji označava bilo koji znak, pošto obje datoteke u imenu imaju riječ `linux` i ekstenziju odnosno nastavak u nazivu datoteke koja završava sa `.csv`. To ćemo napraviti na sljedeći način:

```
grep Ana *linux*.csv

linux-napredno.csv:Ana;Anic;2
uvod-u-linux.csv:Ana;Anic;4
```

**3.** Probajmo ponoviti istu pretragu, ali da se ignoriraju velika i mala slova (`-i`) pa ćemo sada tražiti ime `ana` (malim slovima).

```
grep -i ana *linux*.csv

linux-napredno.csv:Ana;Anic;2
uvod-u-linux.csv:Ana;Anic;4
```

**4.** Pretražimo sve direktorije i pod direktorije, rekurzivno (`-r`) počevši od trenutnog u kojem se nalazimo (`.`) i naravno sve datoteke unutar njih koje sadrže ključnu riječ `ana`

```
grep -r -i ana .

./uvod-u-linux.csv:Ana;Anic;4
./linux-napredno.csv:Ana;Anic;2
```

**4.1.** Ispišimo sâmo datoteke u kojima je pronađeno ime `Ana` ili `ana` (`-l` = samo ispis imena datoteka koje zadovoljavaju kriterije pretrage). Dakle želimo samo naziv datoteka u kojima je pronađena ključna odnosno tražena riječ:

```
grep -r -i ana -l .

./uvod-u-linux.csv
./linux-napredno.csv
```

**5.** Ispišimo sve ostale, osim studenta `ana` korištenjem prekidača `-v` (obrni logiku [sve osim]- *invert match*) iz datoteke: `linux-napredno.csv` na sljedeći način:

```
grep -v -i ana linux-napredno.csv

Ime;Prezime;Ocjena
Marina;Horvat;4
Luka;Lukic;2
Pero;Peric;4
Kristijan;Kristic;3
Maja;Anic;2
```

**6.** Prebrojimo koliko studenata se preziva `Anic`, u istoj datoteci (`-c` = prebrojavanje)

```
grep -c Anic linux-napredno.csv

2
```

**7.** Pronađimo više unosa istovremeno te pri tome pronađimo prezimena i `Anic` i `Lukic`.
**7.1.** Pomoću naredbe `grep` (potrebno je odvajati pojmove sa pipe ( **|** ), ali ga moramo pozivati sa kontrolnom (*escape*) sekvencom **\|** (**pogledajte poglavlje: 5.12.**) da bi se ispravno interpretirao ovaj (**|**) *pipe* znak:

```
grep 'Anic\|Lukic' uvod-u-linux.csv

Ana;Anic;4
Luka;Lukic;2
Maja;Anic;2
```

**7.2.** Pomoću naredbe `egrep` (ovdje ne moramo koristiti *escape* **** karakter odnosno kontrolnu sekvencu za *pipe* znak **|** ):

```
egrep 'Anic|Lukic' uvod-u-linux.csv

Ana;Anic;4
Luka;Lukic;2
Maja;Anic;2
```

**7.3.** U datoteci `/proc/cpuinfo` koja sadrži podatke o procesoru (CPU), ispišimo sve zastavice (engl. *flags*) koje nam daju listu svih funkcionalnosti *CPUa*. Dakle prvo pretražujemo pojam *`flags`*. Potom pronađimo funkcionalnost `aes` jer nas zanima ima li naš CPU podršku za hardversku akceleraciju *AES* algoritma za kriptiranje i dekriptiranje. Ovo ćemo napraviti u dva koraka (da bude razumljivije). Pogledajmo kako; slijedi prvi korak (s nîzom naredbi koje treba izvršiti u jednom retku):

```
grep -m1 flags /proc/cpuinfo

flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf cpuid_faulting pni pclmulqdq
dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic
movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm ida arat epb pln pts
dtherm tpr_shadow vnmi flexpriority ept vpid fsgsbase bmi1 avx2 smep bmi2 erms invpcid xsaveopt
```

Na izlistanju smo dobili samo prvo spominjanje pojma `flags` (pomoću prekidača `-m1`) jer se on pojavljuje u ispisu za svaku CPU jezgru pa nam ne treba isti podatak koji se ponavlja isti broj puta koliko imamo jezgri procesora. Stoga jer se inače za svaku CPU jezgru ispisuju sve zastavice (*flags*) koje ona podržava. I sada ćemo ponoviti sve prethodno uz dodatak, da želimo ispisati i sâm pojam `aes`, ako ga uopće ima u popisu zastavica:

```
grep -m1 flags /proc/cpuinfo  | grep -o aes

aes
```

Dobili smo i potvrdu kako naš CPU hardverski podržava instrukcije `aes`.
Odnosno dobili smo sâmo ispis tražene ključne riječi `aes` koja to potvrđuje.

**Izvori informacija: (K-12), `man grep`.**

## 5.7.2. Naredba *cut*

Naredba `cut` se koristi za raščlanjivanje (*parsiranje*) odnosno pronalaženje dijelova teksta, poput određenog stupca teksta ili slično. Pri tome `cut` procesira redak po redak odnosno liniju po liniju (datoteke), a koji može raščlaniti odnosno segmentirati prema nekom od željenih separatora. Pri tome separatori odnosno filteri za rad ove naredbe mogu biti:

- `-c` – omogućava separaciju prema karakterima (engl. *Character*) tj. "slovima/znakovima". Primjer ovakvog rada bi bio ispis svakog drugog slova iz svakog novog reda teksta (u datoteci), s prekidačem `-c2`.
- `-d` – omogućava separaciju prema određenom polju koje predstavlja određeni stupac podataka. Primjer bi bila ***CSV*** (engl. *comma-separated values*) datoteka iz nekog tabličnog kalkulatora (poput *MS Excel* ili *OpenOffice Calc*) programa, koja je snimljena u ***CSV*** formatu u kojemu je *delimiter* primjerice znak `;`. U tom slučaju, sve što se nalazi između dva znaka `;` predstavlja jedan stupac unutar tablice. To bi onda mogli napraviti s prekidačem: `-d";"`.
- `-b` – omogućava separaciju prema broju *bajta*. Primjerice uzimanje prva četiri *bajta* iz svakog novog reda sa: `-b1-4`.

Pogledajmo nekoliko primjera.
**1.** Imamo datoteku u kojoj su sadržani podaci o korisnički računima `/etc/passwd`, a u kojoj se nalaze sljedeći podaci, poput:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
```

Trenutno nećemo definirati *delimiter* koji je u konkretnoj datoteci znak `:` s kojim se odvajaju stupci unosa, već želimo izvući samo drugi stupac na način da to bude drugo slovo svake riječi u novom retku. To ćemo napraviti na sljedeći način:

```
cut -c2 /etc/passwd

o
a
i
y
y
a
```

Dobili smo ispisano samo drugo slovo iz svakog retka teksta, kako smo i htjeli.

**2**. Želimo izvući sve što se nalazi u prvom stupcu iste datoteke od gore, a što je zapravo lista imena korisničkih računa.
Naime u toj ***CSV*** datoteci *delimiter* (oznaka stupca) je znak `:` pa ćemo ga i upotrijebiti. Pogledajmo i kako:

```
cut -d':' -f1 /etc/passwd

root
daemon
bin
sys
sync
games
```

Prekidač `-d` koristimo kako bismo rekli naredbi da je *delimiter* odnosno znak ograničenja/razgraničenja u ovoj ***CSV*** datoteci znak `:`, a prekidač `-f` kako bismo joj rekli da želimo samo prvi stupac. To bi konkretno bilo: `-f1`.

**3.** Želimo izvući sve što se nalazi u prvom (korisničko ime) i sedmom (*shell* od tog korisnika) stupcu iste datoteke od gore, te zamijeniti *delimiter* sa znakom `;` jer nam to treba za novu ***CSV*** datoteku u kojoj će *delimiter* biti upravo znak `;`

```
cut -d':' -f1,7 --output-delimiter=';' /etc/passwd

root;/bin/bash
daemon;/bin/sh
bin;/bin/sh
sys;/bin/sh
sync;/bin/sync
games;/bin/sh
```

Vidimo kako smo dobili ispisani prvi i sedmi stupac te je delimiter zamijenjen sa znakom `;` (kako smo i htjeli).
Sada tu datoteku eventualno možemo uvesti tj. učitati u neki od programa, poput *OpenOffice Calc* ili *Microsoft Excel*.



Za detalje o datoteci **`/etc/passwd`** pogledate poglavlje:
**7.1.2. Datoteka */etc/passwd*.**

**Izvori informacija: (K-12), `man cut`.**

## 5.7.3. Naredba *awk*

Naredba odnosno program `awk` je nastao 1970 u tvrtki ***Bell Labs*** od strane tri autora: *Alfred* ***A****ho*, *Peter* ***W****einberger* i *Brian* ***K****ernighan*, koji su kombinirali funkcionalnosti programa:

- `egrep` – naredbe za pretraživanje.
- `ed` – uređivača teksta.
- `snobol` – programski jezik *(StriNg Oriented and symBOlic Language)*.
- `c` – programski jezik **C**.

`awk` je interpreterski programski jezik, što znači kako se direktno izvršava bez *kompiliranja*, a koristi se za:

- Procesiranje teksta.
- Izvlačenje podataka.
- Izvještaje.
- Provjeru i manipulacije s tekstom.
- …

Naredbe se kod upotrebe `awk`-a nižu prema principu: `awk "uvjet" {"akcija/radnja"}`
Dalje u tekstu upoznat ćemo se s osnovnim dijelovima programskog jezika `awk`.


Možda niste znali, ali `awk` je i programski jezik i istoimena naredba.

### 5.7.3.1. *Awk* print naredba

Naredba `awk` za početak posjeduje vrlo važnu funkcionalnost ispisa, odnosno prikaza rezultata njegove obrade.
Opis njegovih osnovnih internih naredbi za ispis je sljedeći:

- `{ print }` – ispisuje cijeli sadržaj teksta ili datoteke (ovisno kako se koristi ili poziva).
- `{ print $0}` – ispisuje cijeli sadržaj teksta ili datoteke (`$0` označava sve).
- `{ print $1}` – ispisuje prvi stupac iz cijelog sadržaja teksta ili datoteke.
- `{ print $2}` – ispisuje drugi stupac iz cijelog sadržaja teksta ili datoteke.
- `{ print $1,$2}` – ispisuje prvi i drugi stupac iz cijelog sadržaja teksta ili datoteke.

**Primjeri ispisa**
Kreirali smo tekstualnu datoteku `awk.test.txt`. Ova datoteka predstavlja *log* datoteku u koju se zapisuju računala koja pristupaju sadržaju na internetu, prema imenu računala (PC-1, PC-2, PC-3, …), za svaku odredišnu IP adresu na internetu.
Ova datoteka je slijedećeg sadržaja:

```
1 PC-1 100 121.23.45.67
3 PC-3 150 26.34.78.98
2 PC-1 20 172.156.34.22
6 PC-2 45 121.23.45.67
5 PC-1 234 26.34.78.98
4 PC-2 124 172.156.34.22
7 PC-2 23 172.156.34.22
```

Dakle sadržaj datoteke predstavlja pojednostavljeni dio *log* datoteke u koju se snima statistika izlaza na internet za svako pojedino računalo u našoj lokalnoj mreži. Krećemo s primjerima.

**1.** Iz datoteke `awk.test.txt` želimo prikazati sve (sadržaj cijele datoteke):

```
awk '{print}' awk.test.txt

1 PC-1 100 121.23.45.67
3 PC-3 150 26.34.78.98
2 PC-1 20 172.156.34.22
6 PC-2 45 121.23.45.67
5 PC-1 234 26.34.78.98
4 PC-2 124 172.156.34.22
7 PC-2 23 172.156.34.22
```

**2.** Sada želimo prikazati samo drugi stupac odnosno čije računalo (*hostname* [PC-1, PC-2, …]) je pristupalo na internet:

```
awk '{print $2}' awk.test.txt

PC-1
PC-3
PC-1
PC-2
PC-1
PC-2
PC-2
```

**3.** Ovdje želimo prikazati drugi i treći stupac s čime bi dobili podatak: koja/čija računala prema imenu računala *(hostname*) su, koliko puta pristupala određenim web stranicama. Broj pristupanja, koliko puta se vidi u statistici u trećem stupcu u datoteci.

```
awk '{print $2,$3}' awk.test.txt

PC-1 100
PC-3 150
PC-1 20
PC-2 45
PC-1 234
PC-2 124
PC-2 23
```

Dakle dobili smo ispis drugog (što označava `$2`) i trećeg (što označava `$3`) stupca unutar navedene datoteke.

#### 5.7.3.2. *Awk* varijable

***Slijede napredne cjeline rada u programu awk!.*** U programskom jeziku `awk` moguće je koristiti i varijable.
Naime ovdje je moguće koristiti sistemske varijable ili definirati svoje vlastite.

### Sistemske awk varijable

Opis češće korištenih sistemskih varijabli ugrađenih u sam `awk` jezik, koje odmah možemo koristiti je sljedeći:

- `NR:` – prebrojava broj ulaznih zapisa.
- `NF:` – čuva broj prebrojanih polja u ulaznom zapisu. Pri tome se zadnje polje može pozvati sa: `$NF`.
- `FILENAME:` – sadrži ime trenutno otvorene datoteke (kao ulazne datoteke koja se obrađuje).
- `FS:` – sadrži separator polja (engl. *field separator*). Standardno je to razmak, koji podrazumijeva razmak ili „*tab razmak*“.
- `RS:` – sprema trenutni separator zapisa (engl. *record separator character*). Standardno je to oznaka za novi redak (engl. *Newline*).
- `OFS:` – sprema izlazni separator polja (engl. *output field separator*)”, koji odvaja polja kada ih *awk* ispisuje. Standardna vrijednost je razmak (engl. *space*).
- `ORS:` – sprema izlazni separator zapisa (engl. *output record separator*), koji odvaja izlazne zapise, kada ih *awk* ispisuje. Standardno je to znak za novi redak (engl. *Newline*).
- `OFMT:` – sprema format za numerički izlaz. Standardni format je ”***`%.6g`***“.

**Primjeri**

**1.** Ispišimo brojeve linija (redova) za našu datoteku `awk.test.txt`

```
awk '{print NR}'  awk.test.txt

1
2
3
4
5
6
7
```

**2.** Ispišimo ukupan broj redova unutar naše datoteke `awk.test.txt`

```
awk 'END {print NR}'  awk.test.txt

7
```

Vidimo da imamo ukupno 7 redova (linija)  u navedenoj datoteci.

**3.** Ispišimo broj stupaca za svaku liniju (red) unutar naše datoteke `awk.test.txt`

```
awk '{print NF}'  awk.test.txt

4
4
4
4
4
4
4
```

Vidljivo je kako u svakom redu imamo 4 stupca; ako je separator stupca razmak (engl. *Space*) ili tabularni razmak (engl. *Tab*).

**4.** Ispišimo samo zadnji stupac, bez obzira koliko stupaca ima, unutar naše datoteke `awk.test.txt`

```
awk '{print $NF}'  awk.test.txt

121.23.45.67
26.34.78.98
172.156.34.22
121.23.45.67
26.34.78.98
172.156.34.22
172.156.34.22
```

## *Awk* varijable koje možemo sami definirati

U programu odnosno programskom jeziku *awk*, moguće je definirati i svoje varijable.
Pogledajmo primjere osnovnog kreiranja i korištenja svojih varijabli.

**Primjer**i: Kreirajmo datoteku `varijable.awk` u kojoj ćemo pisati *awk* programski kôd odnosno *awk* program.
Za početak ćemo u navedenoj datoteci kreirati varijablu `BROJ` i dodijeliti joj vrijednost `10`

```
BEGIN {
       BROJ = 10
       print "vrijednost BROJ=" BROJ
}
```

*Awk* program (programski kôd) koji smo napravili, sada ćemo i pokrenuti: nakon prekidača `-f`, slijedi ime *awk* datoteke:

```
awk -f varijable.awk

vrijednost BROJ=10
```

Kao rezultat vidimo kako smo ispisali vrijednost varijable `BROJ`, uz tekst koji smo joj pridružili (`vrijednost BROJ=`)

---

### 5.7.3.3. *Awk* funkcije i for petlje

Programski jezik `awk` nam omogućava kreiranje i pozivanje funkcija. Kroz primjere ćemo vidjeti i kako.

**Primjeri**:
Kreirati ćemo funkciju imena `BROJAC`, koju ćemo poslije pozvati.
Stoga kreirajmo novu datoteku `funkcije.awk`, sa sljedećim sadržajem:

```
function BROJAC()
              {
                for (i = 0; i <10; i++)
                print "BROJAC=" i
              }
```

U ovoj funkciji imena `BROJAC` imamo sljedeću funkcionalnost, koju ćemo objasniti prema logičkim cjelinama:
`for (i = 0; i < 10; i++)` ⇒ kreiramo *FOR* petlju u kojoj varijabla `i` ima vrijednost `0` te se povećava (*increment*) za `1` (`i++`) u svakom prolazu, sve dok ne dosegne vrijednost `10`.
`print "BROJAC=" i` ⇒ ispisujemo tekst (`BROJAC=`) i vrijednost varijable `i`, koja se mijenja/povećava, svakim prolazom kroz petlju. Naime ovo će biti mali brojač od 0 do 9.

Sada nam je potrebno od negdje i pozvati našu funkciju `BROJAC`, a to ćemo postići tako da ćemo na kraj dodati sljedeći kod:

```
BEGIN {
             BROJAC()
}
```

Cijela datoteka: `funkcije.awk` sada izgleda ovako:

```
function BROJAC()
              {
                for (i = 0; i <10; i++)
                print "BROJAC=" i
              }

BEGIN {
             BROJAC()
}
```

Sada ju snimimo. I konačno ju i pozovimo odnosno pokrenimo:

```
awk -f funkcije.awk

BROJAC=0
BROJAC=1
BROJAC=2
BROJAC=3
BROJAC=4
BROJAC=5
BROJAC=6
BROJAC=7
BROJAC=8
BROJAC=9
```

Kao rezultat smo dobili ono što smo i očekivali. U svakom novom prolazu unutar petlje brojač (`i`) se povećava za jedan (1) te se i ispisuje, sve dok vrijednost brojača ne dođe do 9, jer smo rekli kako mora biti manji od 10. Tada petlja završava, kao i naš ***awk*** program.

### 5.7.3.4. *Awk* uvjeti

Kao i svaki drugi programski jezik i `awk` ima mogućnost korištenja uvjeta. Popis uvjeta koji su podržani je sljedeći:

- `if-else`
- `while`
- `do-while`
- `for`
- `switch`
- `break`
- `continue`
- `next`
- `nextfile`
- `exit`

Mi ćemo obratiti osnovi uvjet `if-else`, koji ima sljedeću logiku:

```
if (condition) then-body [else else-body]
```

**Primjeri**: Proširit ćemo prethodni primjer pozivanja funkcija, sa dva uvjeta, tijekom ispisivanja brojeva:

- Jesu li brojevi koji se kreiraju veći ili jednaki 5 ili manji ili jednaki 5.

Kopirajmo datoteku `funkcije.awk` u `if-else.awk`. Dodat ćemo i sljedeći kôd odnosno sljedeće retke:

```
function BROJAC()
             {
               for (i = 0; i <10; i++)
                    if (i <= 5)
                             print "i je manji ili jednak 5 : " i
                    else
                             print "i je veci od 5 : " i
}

BEGIN {
             BROJAC()
}
```

Objasnit ćemo samo `if else` dio:

Pošto se nalazimo u `for` petlji koja "vrti" brojeve od 0 do 9, unutar petlje smo dodali:

```
 if (i <= 5)
                             print "i je manji ili jednak 5 : " i
```

To znači da sve dok je `i` manje ili jednako `5` ispisujemo (`print`) `i je manji ili jednak 5` te vrijednost varijable `i`. I `else` uvjet koji kaže: <u>a inače</u> ispišimo (`print`) `i je veći ili jednak 5` te vrijednost varijable `i`.

```
else
                             print "i je veci od 5 : " i
```

Pokrenimo sada ovaj primjer, odnosno datoteku (`if-else.awk`) koju smo napravili:

```
awk -f if-else.awk

i je manji ili jednak 5 : 0
i je manji ili jednak 5 : 1
i je manji ili jednak 5 : 2
i je manji ili jednak 5 : 3
i je manji ili jednak 5 : 4
i je manji ili jednak 5 : 5
i je veci od 5 : 6
i je veci od 5 : 7
i je veci od 5 : 8
i je veci od 5 : 9
```

U ispisu vidimo kako smo u prvih pet iteracija (prolazaka petljom) postigli da je ispis, kada je vrijednost varijable `i` manja od pet (5), da se to i ispisuje: „`i je manji ili jednak 5 :`".

Dok je u slijedećim iteracijama, kada je vrijednost varijable `i` veća od pet (5), vidljivo kako se ispisuje druga poruka, koja upravo to i govori: „`i je veci od 5 :`".

### 5.7.3.5. *Awk* polja (*Array*)

Polje (engl. *Array*) predstavlja tablicu s vrijednostima koje zovemo elementima. Elemente polja razlikujemo po indeksima. Prema tome, polja možemo promatrati kao varijable koje sadrže više elemenata. Imena polja imaju istu sintaksu kao i imena varijabli, ali ne smijemo koristiti isto ime i za polje i za varijablu. U ***awk***-u se za polje, definicija broja elemenata ili komponenti ne mora definirati prije korištenja, za razliku od većine programskih jezika. Krenimo s primjerima:

**1.** Kreirajmo datoteku imena `array.awk` koja sadrži sljedeći ***awk*** programski kôd:

```
{
        IP[$4]++;
}

END {
           for (var in IP){
                 print var, "Pristupano", IP[var]," puta"
           }
}
```

Koristit ćemo već poznatu datoteku, kao ulaznu, a koja je pojednostavljena *log* datoteka izlaza na internet, koja u četvrtom stupcu ima popis IP adresa kojima se pristupalo na internetu. Dakle ta datoteka: `awk.test.txt` izgleda ovako:

```
1 PC-1 100 121.23.45.67
3 PC-3 150 26.34.78.98
2 PC-1 20 172.156.34.22
6 PC-2 45 121.23.45.67
5 PC-1 234 26.34.78.98
4 PC-2 124 172.156.34.22
7 PC-2 23 172.156.34.22
```

Mi s gore navedenim naredbama (***awk*** kodom) želimo kreirati polje imena `IP` koje će sadržavati sve IP adrese iz četvrtog stupca naše datoteke: `awk.test.txt`. To smo postigli sa slijedećim blokom odnosno logičkom cjelinom unutar našeg ***awk*** programskog kôda:

```
{
        IP[$4]++;
}
```

Ako pogledamo navedeni ***awk*** programski kôd: ovdje kreiramo polje imena `IP` u koje ubacujemo vrijednosti iz četvrtog stupca (to su IP adrese) te ih svaki puta uvećamo (*increment*: `++`), odnosno povećavamo polje, sa svakim novim elementom.

```
for (var in IP){
```

Ovdje za elemente (`var`) polja `IP` koje smo popunili u primjeru gore, sa:

```
print var, "Pristupano", IP[var]," puta"
```

Ispisujemo svaku pojedinu IP adresu iz polja (`print var`) uz pripadajući tekst te ispisujemo koliko je bilo ponavljanja iste IP adrese, sa: `IP[var]`

**2.** Sada pokrenimo naš novi program imena `array.awk`, koji će obraditi ulaznu log datoteku imena: `awk.test.txt`:

```
awk -f array.awk awk.test.txt

26.34.78.98 Pristupano 2  puta
121.23.45.67 Pristupano 2  puta
172.156.34.22 Pristupano 3  puta
```

Dobili smo uredno odrađeno upravo ono što smo i očekivali.

**3.** Istu stvar mogli smo pokrenuti i iz jedno linijskog `awk` kôda, s istim rezultatom. Probajmo to učiniti i na ovaj način:

```
awk '{IP[$4]++;} END{for (var in IP) print var, "Pristupano", IP[var]," puta"}'
awk.test.txt

26.34.78.98 Pristupano 2  puta
121.23.45.67 Pristupano 2  puta
172.156.34.22 Pristupano 3  puta
```

**4.** Napravit ćemo istu stvar i pomoću nîza dostupnih linux naredbi (izvršavamo ih u jednom redu, kako i piše):

```
cat awk.test.txt | awk  '{print $4}' | sort | uniq -c | sort -nr

      3 172.156.34.22
      2 26.34.78.98
      2 121.23.45.67
```

**1.** Ispisujemo datoteku `awk.test.txt` sa naredbom `cat` i preusmjeravamo ju ( `|` ) sljedećoj naredbi.
**2.** Potom sa naredbom `awk` ispisujemo samo četvrti (4) stupac i preusmjeravamo ga ( `|` ) sljedećoj naredbi.
**3.** Sortiramo IP adrese s naredbom `sort` i preusmjeravamo ( `|` ) ispis sljedećoj naredbi.
**4.** Prebrojavamo broj istih IP adresa naredbom `uniq -c` i preusmjeravamo ( `|` ) ispis sljedećoj naredbi.
**5.** Ponovno pozivamo naredbu `sort -nr` ali sada kako bismo poredali sve po prvom stupcu odnosno broju konekcija koje smo prebrojavali u koraku 4.

### 5.7.3.6. *Awk* pretraživanje

Pomoću jezika `awk`, između ostalog, moguće je i pretraživati tekst prema nekoj ključnoj riječi. U tom slučaju, sintaksa je:
`awk '/XY/ {print … … … }' FILE` → s time se traži pojam odnosno ključna riječ `XY` u datoteci imena: `FILE`

Krenimo s primjerima:
**1.** Iz datoteke ocjena studenata: `uvod-u-linux.csv` pronađimo sve studente koji se prezivaju `Anic`.
Pošto je ovo ***CSV*** datoteka u kojoj je delimiter `;` pa ga moramo definirati sa `-F`, ali uz korištenje kontrolne sekvence odnosno *escape* karaktera `\` što u konačnici znači kako ga moramo pozvati sa `-F\;` na sljedeći način:

```
awk -F\; '/Anic/' uvod-u-linux.csv

Ana;Anic;4
Maja;Anic;2
```

**2.** Isti primjer možemo proširiti i s drugim mogućnostima ***awk***a. Primjerice filtrirajmo samo treći (3) stupac iz primjera gore, a koji predstavlja brojčane ocjene studenata:

```
awk -F\; '/Anic/ {print $3}' uvod-u-linux.csv

4
2
```

---

### 5.7.3.7. *Awk* aritmetičke operacije i operatori

Programski jezik `awk` nam omogućava i upotrebu aritmetičkih operacija i operatora. Pogledajmo njihovu listu.

**Operatori :**

- `>` veće od te `>=` veće ili jednako od.
- `<` manje od te `<=` manje ili jednako od.
- `==` jednako sa.
- `!=` različito od.
- `&&` oba izraza moraju biti istinita.
- `||` bilo koji od izraza mora biti istinit.

**Aritmetičke operacije:**

- `+` za zbrajanje te `-` za oduzimanje.
- `/` za dijeljenje te `*` za množenje.
- `^` za potenciranje.
- `%` môdul broja (primjerice: 16%6=4, odnosno: 16 môdul 6 = 4).

**Popis unarnih operacija:**

- `+` pretvaranje brojeva u pozitivne.
- `-` pretvaranje brojeva u negativne.
- `++` automatsko povećavanje (engl. *Increment*).
- `--` automatsko smanjivanje (engl. *Decrement*).

Iz naše datoteke s ocjenama, želimo zbrojiti sve ocjene i podijeliti ih s brojem studenata kako bismo dobili prosjek ocjena. Pošto prvi redak naše ***CSV*** datoteke `uvod-u-linux.csv` sadrži opis svih stupaca, moramo ga maknuti. To ćemo napraviti pomoću naredbe `tail -n +2` koja će ispisati sve retke počevši od drugog retka, pa sve do kraja datoteke. Sada možemo zbrojiti sve zadnje stupce odnosno konkretno treći stupac (`x = x +$3`) te rezultat podijeliti s brojem svih redova (osim prvog kojeg smo maknuli), što postižemo sa: `print x/NR`. Pogledajmo što ćemo napraviti:

```
tail -n +2 uvod-u-linux.csv | awk -F\; '{ x = x + $3} END { print x/NR }'

3.66667
```

Iz naše datoteke sa logovima pristupanja internetu (`awk.test.txt`), pomnožimo u svakom redu prvi stupac sa 2:

```
awk '{print $1*2}' awk.test.txt

2
6
4
12
10
8
14
```

Pogledajmo i kako koristiti operator manje od (`<`). Dakle ispisat ćemo zauzeće RAM memorije te provjeriti imamo li na sustavu manje od 3GB RAM memorije slobodno (prikazano u MB). U ispisu naredbe `free`, nakon što filtriramo `Mem` statistiku, na sedmoj poziciji odnosno stupcu (`$7`) se nalazi informacija o količini dostupne RAM memorije koju provjeravamo.
Ako je uvjet **<3000** zadovoljen, dobit ćemo neki ispis; u suprotnom nećemo dobiti ništa ispisano.

Pogledajmo ovaj primjer:

```
free -m | grep Mem | awk '$7 < 3000'
```

Za više detalja o programskom jeziku `awk`, možete pogledati knjigu: https://www.gnu.org/software/gawk/manual/gawk.pdf

**Izvor informacija: (1060), `man awk`.**

## 5.7.4. Naredba *sed*

Naredba `sed` (engl. *Stream Editor*), koristi se za sintaksno provjeravanje (*parsiranje*) i transformiranje teksta odnosno podataka. Ova naredba podržava i regularne izraze. Najčešće korišteni regularni izrazi u ***sed-u*** su:

- `*` – označava niti jedan ili bilo koliko karaktera odnosno slova/brojeva/znakova.
- `\+` – slično kao `*` ali za jedan ili više karaktera odnosno slova/brojeva/znakova.
- `\?` – označava niti jedan ili jedan karakter odnosno slovo/broj/znak.
- `\{i\}` – slično kao `*` ali za točno `i` broj karaktera odnosno slova/brojeva/znakova.
- `\{i,j\}` – označava između `i` i `j` broja karaktera odnosno slova/brojeva/znakova, uključujući oba.
- `\{i,\}` - označava više ili jednako `i` karaktera odnosno slova/brojeva/znakova.
- `.` – označava bilo koji karakter odnosno slovo/broj/znak uključujući novu liniju.
- `^` – označava početak retka teksta.
- `$` – označava zadnji (ili prazni) redak (liniju).

**Slijede primjeri.**

Kreirat ćemo datoteku imena: `ulaz.txt` koja će sadržavati sljedeće retke:

```
POCETAK
STARO
NOVO
KRAJ
```

### 5.7.4.1. *Sed* zamjena stringova

U ovoj cjelini vidjeti ćemo kako se iz programa `sed` radi zamjena znakova/pojma (stringova). Iz datoteke `ulaz.txt`, svugdje gdje se pojavljuje pojam (*string/riječ*) `STARO`, zamijeniti ćemo sa riječi `NOVO` i to sve snimiti u novu datoteku imena: `izlaz.txt`:

```
sed 's/STARO/NOVO/g' ulaz.txt> izlaz.txt
```

Koristili smo slovo `s` koje označava zamjenu (engl. *Substitute*=***s***), te na kraju i slovo `g` koje označava kako želimo da se zamjena pojmova radi na cijelom dokumentu (engl. *Global*=***g***) odnosno datoteci koju čitamo (`ulaz.txt`).

Pogledajmo rezultat, koji se zapisao u datoteku: `izlaz.txt`, sa sljedećom naredbom:

```
cat izlaz.txt

POCETAK
NOVO
NOVO
KRAJ
```

Sada ćemo napraviti dvije zamjene odjednom (u datoteci: `ulaz.txt`): `STARO` u `NOVO` i `POCETAK` u `KRAJ` na sljedeći način:

```
sed 's/STARO/NOVO/g ; s/POCETAK/KRAJ/g' ulaz.txt> izlaz.txt
```

Pogledajmo što smo dobili:

```
cat izlaz.txt

KRAJ
NOVO
NOVO
KRAJ
```

Kreirajmo novu datoteku `ulaz.txt` koja će sadržavati sljedeće:

```
POCETAK
NOVO NOVO
NOVO NOVO
KRAJ
```

Zatim probajmo promijeniti pojam `NOVO` u `NOVIJE` počevši od početka datoteke, red po red ali samo jedan i to prvi (`1`) pronađeni pojam u svakom redu (ispis ćemo dobiti odmah jer nismo rezultat snimili u novu datoteku):

```
sed 's/NOVO/NOVIJE/1' ulaz.txt

POCETAK
NOVIJE NOVO
NOVIJE NOVO
KRAJ
```

Vidimo da smo dobili upravo ono što smo i tražili.

Da smo htjeli promijeniti samo drugi pronađeni pojam (`NOVO`) u (`NOVIJE`) u svakom redu, to bi mogli postići sa:

```
sed 's/NOVO/NOVIJE/2' ulaz.txt
```

Moguće je i samo i isključivo u određenom redu (liniji) napraviti željenu promjenu.
Mi ćemo napraviti promjenu u drugom redu (`2`), što ćemo napravili sa sljedećom naredbom:

```
sed '2 s/NOVO/NOVIJE/' ulaz.txt
```

Dodatno, moguće je navoditi i opseg redova/linija, odvajajući ih zarezom; primjerice napravi promjene od reda 1 do reda 3, pa bi to tada izgledalo ovako:

```
sed '1,3 s/NOVO/NOVIJE/' ulaz.txt
```

Ako želimo promjenu napraviti samo u zadnjem redu teksta (`$`) odnosno datoteke, to možemo na sljedeći način:

```
sed '$ s/KRAJ/NIJEKRAJ/' ulaz.txt
```

***Napomena: ako je zadnji red prazan odnosno u njemu ništa ne piše, ali on postoji, i on se računa kao red teksta.***

Kako pronaći i zamijeniti pojam koji sadrži neki posebni znak, poput znaka /. Naime ako želimo primjerice zamjeniti pojam `/bin/bash` on sadrži posebni znak /, za to moramo koristiti kontrolnu sekvencu odnosno koji moramo *escape*-ati odnosno naložiti da se on promatra kao obični znak (slovo ili broj).



Pogledjte poglavlje:
**5.12. Regularni izrazi**.

Ako bi htjeli primjerice pojam `/bin/bash` zamijeniti s pojmom `/usr/bin/bash` to bi onda napravili na sljedeći način:

```
sed 's/\/bin\/bash/\/usr\/bin\/bash/g' ulaz.txt
```

Kako zamijeniti cijeli red ako je određeni pojam pronađen. To se postiže korištenjem opcije `c`. Ovdje ćemo, ako u bilo kojem retku pronađemo pojam `NOVO` cijeli redak teksta zamijeniti s tekstom `PRAZNO`:

```
sed '/NOVO/ c PRAZNO' ulaz.txt
```

Sada ćemo vidjeti kako pronaći redak koji sadrži određeni pojam i onda unutar tog retka zamijeniti određeni (drugi pojam).
Prvo tražimo retke koji sadrže pojam `NOVO`, a onda ćemo u njima novi pojam `TEST` zamijeniti s pojmom `TIPKA` u novoj datoteci koja će se zvati: `ulaz2.txt`.

```
sed '/NOVO/ s/TEST/TIPKA/' ulaz2.txt
```

### 5.7.4.2. *Sed* brisanje stringova

U ovoj cjelini proći ćemo primjere u kojima ćemo s naredbom `sed` brisati određene riječi odnosno željene pojmove.
Obrišimo pojam `STARO` te rezultat upišimo u novu datoteku `izlaz.txt`

```
sed 's/STARO//' ulaz.txt> izlaz.txt
```

Datoteka `ulaz.txt` sadrži sljedeće retke teksta:

```
POCETAK
STARO
NOVO
KRAJ
```

Pogledajmo rezultat:

```
cat izlaz.txt

POCETAK

NOVO
KRAJ
```

**Napomena:** direktne promjene na datoteci mogu se raditi ako koristimo prekidač `-i` poput:

```
sed -i 's/STARO//' ulaz.txt
```

Unutar datoteke `ulaz.txt` obrišimo (`d`) prazne redove (`^$`)

```
sed -i '/^$/d' ulaz.txt
```

Na ispisu datoteke `ulaz.txt` obrišimo sve od 1 do 3 reda (uključujući 3. red)

```
cat ulaz.txt | sed '1,3d'
```

Ovo smo mogli napraviti i ovako (prekidač `-e` znači kako slijedi `sed` naredba):

```
sed -e '1,3d' ulaz.txt

KRAJ
```

Iz datoteke `ulaz.txt` obrišimo sve redove između određena dva retka koja sadrže riječi `POCETAK` i `NOVO`

```
sed -e '/POCETAK/,/NOVO/d' ulaz.txt

KRAJ
```

**Izvor informacija: (747), `man sed`, `info sed`.**

## 5.7.5. Naredba *tr*

Linux naredba `tr` (engl. *Translate characters*(*prevedi* znakove/karaktere)) koristi se za transformaciju ili brisanje (određenih) pojedinih karaktera odnosno slova ili znakova. Naredba `tr` radi razne operacije sa pojedinim karakterima odnosno znakovima, za razliku od naredbe `sed`, koja radi s cijelim riječima odnosno s cijelim *stringovima*.

**Sintaksa** naredbe je
`tr` *opcije PARAMETAR1 PARAMETAR2*
Pri tome je obično *PARAMETAR1* nîz koji zamjenjujemo, a *PARAMETAR2* nîz s kojim ga mijenjamo. Dodatno, kao parametar može biti neka od ugrađenih funkcionalnosti sâmog programa. Ovo pravilo vrijedi ukoliko ne koristimo prekidač `-d`.

Neke od ugrađenih funkcionalnosti naredbe `tr` su vidljive u tablici:

| Vrijednost | Opis |
|---|---|
| `[:alnum:]` | Označava sva slova i brojeve. |
| `[:alpha:]` | Označava sva slova. |
| `[:blank:]` | Označava sve razmake (*horizontal whitespace*). |
| `[:space:]` | Označava sve razmake (*horizontal or vertical whitespace*). |
| `[:cntrl:]` | Označava sve kontrolne karaktere (znakove). |
| `[:digit:]` | Označava sve brojeve. |
| `[:graph:]` | Označava sve ispisive (ASCII kodovi od 32 do 127) karaktere, ne uključujući razmak. |
| `[:lower:]` | Označava sva mala slova. |
| `[:print:]` | Označava sve ispisive (ASCII kodovi od 32 do 127) karaktere, uključujući razmak. |
| `[:upper:]` | Označava sva velika slova. |
| ... | … |

Osim toga, postoje i interpreterske sekvence, koje možemo koristiti:

| Vrijednost | Opis |
|---|---|
| `\\` | Obrnuta kosa crta *(engl. Backslash*). |
| `\b` | Pomak unatrag *(engl. Backspace).* |
| `\n` | Oznaka za novi redak *(engl. Newline).* |
| `\h` | Oznaka za horizontalni „*tab*“ razmaka (engl. *Horizontal tab*). |
| `\v` | Oznaka za vertikalnih „*tab*“ razmaka (engl. *Vertical tab*). |

**Primjeri**
Kreirat ćemo tekstualnu datoteku za primjer. Datoteka će se zvati: `ulaz.txt`. Sadržaj joj je slijedeći:

```
VELIKO
malo
12345
Ovo je test (x)
67890
```

**1.** Konverzija malih slova u velika. Ovo možemo izvesti na nekoliko načina. Vidjet ćemo kako su se sva mala slova konvertirala u velika.

**Slijedi prvi primjer**

```
tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ < ulaz.txt

VELIKO
MALO
12345
OVO JE TEST (X)
67890
```

Navedenom naredbom se napravilo sljedeće:

- Svako malo slovo "**a**" se mijenja s velikim slovom "**A**".
- Svako malo slovo "**b**" se mijenja s velikim slovom "**B**".
- …. i tako do kraja.

**Pogledajmo drugi primjer** iste funkcionalnosti:

```
tr [:lower:] [:upper:] < ulaz.txt

VELIKO
MALO
12345
OVO JE TEST (X)
67890
```

**Slijedi i treći primjer** iste funkcionalnosti:

```
tr a-z A-Z < ulaz.txt

VELIKO
MALO
12345
OVO JE TEST (X)
67890
```

Vidimo kako smo na tri načina postigli istu funkcionalnost. Vidljivo je i to kako se ulazna datoteka koristi pomoću preusmjeravanja (*redirekcije*) `<` . Ako imamo potrebu da se izlaz odnosno rezultat zapisuje u drugu datoteku, tada se može koristiti preusmjeravanje u drugom smjeru `>` i to prema novoj datoteci. Dakle u zadnjem primjeru bi to tada bilo:

```
tr a-z A-Z < ulaz.txt > izlaz.txt
```

**2.** Zamjena jednog (ili nîza) alfanumerika s drugim. U ulaznoj datoteci (`ulaz.txt`), zamijenimo nîz `67890` sa `12345`:

```
tr '67890' '12345' < ulaz.txt

VELIKO
malo
12345
Ovo je test (x)
12345
```

**3.** Zamijenimo obične zagrade `()` s uglatim zagradama `[]`

```
tr '()' '[]' < ulaz.txt

VELIKO
malo
12345
Ovo je test [x]
67890
```

**4.** Zamijenimo razmake s *tabovima*:

```
tr [:blank:] '\t' < ulaz.txt

VELIKO
malo
12345
Ovo	je	test	(x)
67890
```

**5.** Kreirajmo novu datoteku imena `ulaz2.txt`, sljedećeg sadržaja:

```
Ovo            je               test
```

Sada ćemo sažeti višak razmaka između riječi, pomoću prekidača `-s`.

```
tr -s [:space:] ' ' < ulaz2.txt

Ovo je test
```

Brisanje pojedinog karaktera (znaka/slova) ili niza karaktera: provjerava se svaki zasebno. Za brisanje se koristi prekidač `-d`.

Obrišimo karaktere (slova): veliko "`O`" i malo "`v`" iz datoteke `ulaz2.txt`:

```
tr -d 'Ov' < ulaz2.txt

o             je               test
```

**6.** Obrišimo sve brojeve (i samo brojeve) iz datoteke `ulaz.txt`

```
tr -d  [:digit:] < ulaz.txt

VELIKO
malo

Ovo je test (x)
```

**7.** Možemo raditi i invertne operacije, dodavanjem prekidača `-c`.
Sada napravimo invertnu operaciju od prijašnjeg primjera. Obrišimo sve što nisu brojevi:

```
tr -cd [:digit:] < ulaz.txt

1234567890
```

**8.** A sada obrišimo (uklonimo) sve karaktere koji nisu printabilni. Dakle i one koji označavaju novi redak ili kraj retka i slične:

```
tr -cd [:print:] < ulaz.txt

VELIKOmalo12345Ovo je test (x)67890
```

Vidimo kako su se sada svi redovi spojili u jedan jer više nema posebnih karaktera (znakova) na kraju svakog retka, koji upravo i označavaju kraj svakog retka teksta.



*Unix*/Linux format koristi samo ***EOL*** oznaku (engl. *End Of Line*) na kraju svakog retka, kao oznaku za kraj retka. **EOL** ima heksadecimalnu vrijednost **`0A`**.
Dakle mi smo praktično obrisali nevidljivi znak **`0A`** koji se nalazi na kraju svakog retka teksta.



Za više detalja o unutarnjem formatu datoteka, pogledajte poglavlja:
**4.7. Format datoteka (*MIME type*).**
**10.3.2 Kodiranje, dekodiranje i kodne stranice**.

**9.** Kreirajmo datoteku `test.txt` koja će sadržavati samo četiri prazna reda.
Koristit ćemo program `vi`, a vi možete koristiti bilo koji drugi uređivač teksta, poput programa `vim`, `nano` ili `ed`.

Pogledajmo sada ovu datoteku s heksadecimalnim preglednikom odnosno naredbom `hexdump`, na sljedeći način:

```
hexdump test.txt
```

Dobit ćemo sljedeći ispis:

```
0000000 0a0a 0a0a
0000004
```

Vidljivo je kako imamo četiri puta oznaku **0A** dakle četiri karaktera/znaka koji označavaju kraj retka.

Kontrolna sekvenca (tzv. *Escape sekvenca*) za novi redak je `\n` pa, ako recimo želimo maknuti samo oznaku za kraj retka, nakon koje slijedi novi redak teksta, možemo napraviti sljedeće (na datoteci `ulaz.txt`):

```
tr -s '\n' ' ' < ulaz.txt

VELIKO malo 12345 Ovo je test (x) 67890
```

**Ako želimo unositi ASCII kodove umjesto slova, brojeva ili znakova, moramo koristiti oktalni zapis.**
**Pogledajte kraj knjige s ASCII tablicom:** **poglavlje: 29.1 ASCII tablica.**

**Slijedi napredni primjer!**
Pogledajmo spregu naredbi `dd` (engl. *Disk dump*) i naredbe `tr`.
Kreirajmo datoteku koja se sastoji od binarnih nula, pomoću naredbe `dd` i generatora nula `/dev/zero` te potom konvertirajmo te nule u veliko slovo `A`. Sve snimimo u datoteku naziva: `slova.txt`. Veličina datoteke će biti 1024 *bajta*.

```
dd if=/dev/zero count=1 bs=1024 | tr '\000' '\101' > slova.txt
```

Pogledajmo sada našu datoteku `slova.txt`, s naredbom `cat`:

```
cat file.txt

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

**Izvor informacija:** **`man tr`**,**`info tr`**,**`man dd`**,**`man cat`**,**`man hexdump`**.

---

## 5.7.6. Naredbe *join* i *paste*

U određenim situacijama, imat ćemo potrebu kombinirati sadržaj datoteka, u kojemu će rezultat odnosno odredišna datoteka, kao rezultat kombiniranja, zadovoljiti naše potrebe. Za ovakve zadatke postoje dvije vrlo korisne naredbe:

- `join` - služi za povezivanje/spajanje linija (redova) iz više datoteka, na osnovu nekog (zajedničkog) polja ili riječi horizontalno odvojenih unutar datoteke (teksta) (engl. *Join*(spoji/udruži)).
- `paste` - služi za povezivanje/spajanje linija (redova) iz više datoteka, također obično horizontalno odvojenih (*tabom* ili nekim drugim delimiterom).

**Naredba `join`**

Naredba `join` standardno kao horizontalni delimiter prepoznaje razmak (engl. *Space*), a obično se koristi za povezivanje ili kombiniranje više ulaznih podataka, odnosno datoteka koje želimo kombinirati ili povezati na određeni način.

**Pogledajmo primjere**
Zamislimo da imamo sljedeće dvije datoteke: `kontinenti.txt` koja sadrži sljedeće retke teksta:

```
1. Azija:
2. Afrika:
3. Europa:
4. Sjeverna Amerika:
```

Te datoteku `drzave.txt` koja sadrži sljedeći tekst:

```
1. Indija
2. Egipat
3. Hrvatska
4. SAD
```

Dakle prva datoteka sadrži listu kontinenata, a druga listu država, logički povezanih s kontinentima iz druge datoteke.

Pozovimo našu naredbu `join` da poveže gornje dvije datoteke (`kontinenti.txt` i `drzave.txt`):

```
join kontinenti.txt drzave.txt
```

I dobivamo kao rezultat:

```
1. Azija: Indija
2. Afrika: Egipat
3. Europa: Hrvatska
4. Sjeverna Amerika: SAD
```

Dakle povezali smo kontinente s državama. Ako ipak prvi dokument proširimo dodavanjem nove države te dobijemo sljedeće:

```
1. Azija:
2. Afrika:
3. Europa:
4. Sjeverna Amerika:
5. Južna Amerika
```

I ponovno napravimo `join`, tada ćemo dobiti:

```
1. Azija: Indija
2. Afrika: Egipat
3. Europa: Hrvatska
4. Sjeverna Amerika: SAD
```

To je stoga jer se ne može pronaći par zadnjem kontinentu (dodanim pod rednim brojem `5.`), iz datoteke `drzave.txt`.

Ako ipak želimo prikazati i ne uparene podatke odnosno u ovom slučaju kontinente koji nisu upareni (povezani) s niti jednom državom, to možemo napraviti pomoću prekidača: `-a 1`

```
join kontinenti.txt drzave.txt -a1

1. Azija: Indija
2. Afrika: Egipat
3. Europa: Hrvatska
4. Sjeverna Amerika: SAD
5. Južna Amerika
```

Sada smo dobili i kontinent, iako u njemu nema definirane niti jedne države. Ako pak datoteku `drzave.txt` proširimo tako da dodajemo još jednu državu, ali ne u isti redak već u novi redak s identifikatorom, u našem slučaju prvim brojem, koji je zajednički za sve ulazne datoteke (u našem slučaju i za: `kontinenti.txt`), poput:

```
1. Azija: Indija
2. Afrika: Egipat
3. Europa: Hrvatska
3. Europa: Danska
4. Sjeverna Amerika: SAD
5. Južna Amerika
```

Tada ćemo s upotrebom naredbe `join` dobiti sljedeće:

```
join kontinenti.txt drzave.txt

1. Azija: Indija
2. Afrika: Egipat
3. Europa: Hrvatska
3. Europa: Danska
4. Sjeverna Amerika: SAD
```

U slučaju da smo samo u istom redu (`3.`) unutar datoteke `drzave.txt` dodali novu državu, pa bi taj red izgledao ovako:

```
3. Europa: Hrvatska Danska
```

Ispis `join` naredbe bi bio:

```
1. Azija: Indija
2. Afrika: Egipat
3. Europa: Hrvatska Danska
4. Sjeverna Amerika: SAD
```

Moguće je i mijenjati zajednički identifikator odnosno poveznicu. Promijenimo sada naše datoteke, da izgledaju tako, da više prvi znak u svakom redu neće biti poveznica, već će to postati drugi znak, odnosno znak na drugom mjestu.
Datoteka `drzave.txt` će sada izgledati ovako:

```
# 1. Indija
% 2. Egipat
* 3. Hrvatska
/ 3. Danska
```

Te datoteku `kontinenti.txt` promijenimo da izgleda ovako:

```
# 1. Azija:
/ 2. Afrika:
) 3. Europa:
& 4. Sjeverna Amerika:
# 5. Južna Amerika
```

Sada naredbi join moramo reći u kojem stupcu (koji znak) je identifikator, u našem slučaju je to i za prvu datoteku (-1) i za drugu datoteku (-2), pozicija dva odnosno drugi stupac (2), jer su tu brojevi, koji su stvarno i njihova poveznica:

```
join kontinenti.txt drzave.txt -1 2 -2 2

1. # Azija: # Indija
2. / Afrika: % Egipat
3. ) Europa: * Hrvatska
3. ) Europa: / Danska
4. & Sjeverna Amerika: " SAD
```

U slučajevima kada su nam poveznice slova, razlikuju se velika i mala, pa datoteka poput: `drzave.txt`, ako ovako izgleda:

```
A. Indija
B. Egipat
C. Hrvatska
D. SAD
```

I ako datoteka: `kontinenti.txt` izgleda ovako:

```
a. Azija:
b. Afrika:
c. Europa:
d. Sjeverna Amerika
```

Tada one neće biti povezane sve dok ne uključimo prekidač `-i` koji nalaže naredbi join da ignorira razliku između velikih i malih slova.

To ćemo napraviti na sljedeći način:

```
join -i kontinenti.txt drzave.txt

a. Azija: Indija
b. Afrika: Egipat
c. Europa: Hrvatska
d. Sjeverna Amerika: SAD
```

I na kraju, moguće je naložiti naredni join, da provjerava je li redoslijed po abecedi, sa prekidačem: `--check-order` ili da to ignorira, s prekidačem: `--nocheck-order`

**Pogledajmo i jedan praktičan primjer upotrebe ove naredbe.**

Kod NUMA arhitekture računala, koja statistike zapisuje u datoteke: `/sys/devices/system/node/node0/numastat` i `/sys/devices/system/node/node1/numastat` (za ***NUMA*** sustav s dva procesora), koje obično izgledaju ovako:

```
numa_hit 3655302
numa_miss 0
numa_foreign 0
interleave_hit 12602
local_node 3655302
other_node 0
```

S time da je u prvom stupcu opis koji je jednak za obje datoteke (pr. `numa_hit`), dok se mijenjaju samo vrijednosti odnosno brojevi unutar statistika.

Naredba join će nam ovdje pomoći jer će u obje datoteke pronaći zajednički naziv reda, i prikazati će samo vrijednosti (izmjerene statistike koje se mijenjaju) jedne i druge datoteke kao da se radi o novim stupcima.

Pokrenimo ju na sljedeći način:

```
join /sys/devices/system/node/node0/numastat /sys/devices/system/node/node1/numastat

numa_hit: 31413614, 16686285
numa_miss: 0, 0
numa_foreign: 0, 0
interleave_hit: 20108, 20261
local_node: 31413376, 16664958
other_node: 238, 21327
```

Opis retka — Vrijednost (**NUMA0**) — Vrijednost (**NUMA1**)



Za više detalja o ***NUMA*** arhitekturi pogledajte poglavlje:
**10.7.2.2 NUMA**

**Izvori informacija: (282),(283),(284), `man join`, `info join`.**

**Naredba paste**

Naredba `paste` je slična prethodnoj naredbi, pa ćemo uzeti slične datoteke uz malu razliku. Naime naredba `paste` ne treba zajednički identifikator koji se mora nalaziti u svakoj od datoteka koje povezujemo, već se povezuju red s redom.
Stoga datoteke mogu biti i sljedećeg sadržaja.
Datoteka `kontinenti.txt` može izgledati ovako:

```
Azija
Afrika
Europa
Sjeverna Amerika:
```

Te datoteka `drzave.txt` može izgledati ovako:

```
Indija
Egipat
Hrvatska
SAD
```

Ako pokrenemo naredbu `paste` na sljedeći način:

```
paste kontinenti.txt drzave.txt
```

Dobivamo:

```
Azija  Indija
Afrika Egipat
Europa Hrvatska
Sjeverna Amerika       SAD
```

Što odgovara onome što bi dobili i s naredbom `join`, ali ovdje bez potrebe za zajedničkim identifikatorom odnosno poveznicom.
Moguće je i promijeniti prikaz u vertikalni, pomoću prekidača `-s`, pa pogledajmo kako će to onda izgledati ovako:

```
paste  -s kontinenti.txt drzave.txt

Azija   Afrika  Europa  Sjeverna Amerika
Indija  Egipat  Hrvatska        SAD
```

Dodatno, moguće je mijenjati *delimiter*, koji je standardno razmak (engl. *Space*), pomoću prekidača `-d`.
Ako bi pak primjerice listu država proširili, ali ne standardno, dodavanjem nove države u isti red odvojenih s razmakom, već sa primjerice znakom `:` tada bi morali taj znak definirati kao *delimiter*. Pogledajmo sada novu datoteku: `drzave.txt`:

```
Indija:Nepal
Egipat:Maroko
Hrvatska:Danska
SAD
```

Pokrenimo naredbu `paste` s novim *delimiterom* koji će ovdje biti dvotočka (`:`):

```
paste -d: kontinenti.txt drzave.txt
```

I dobivamo sljedeće:

```
Azija:Indija:Nepal
Afrika:Egipat:Maroko
Europa:Hrvatska:Danska
Sjeverna Amerika:SAD
```

**Izvori informacija: (285),(286), `man paste`,`info paste`.**

---

## 5.7.7. Sortiranje sadržaja: naredba *sort*

Za sortiranje sadržaja datoteka, možemo koristiti naredbu `sort` koja dodatno prepoznaje i stupce teksta odnosno standardno za separator (*delimiter*) stupca za CSV format datoteke, koristi razmak (što možemo mijenjati). Važno je razumjeti kako ova naredba ne mijenja sadržaj izvorne tekstualne datoteke, već samo ispisuje rezultat sortiranja. Osim za sortiranje sadržaja datoteka, moguće je standardno i sortirati izlazni nîz podataka iz bilo koje prethodne naredbe pomoću *pipe* funkcionalnost.

Prvo pogledajmo neke od osnovnih prekidača ove naredbe:

- `-f` – ignoriraj veliko/malo sovo.
- `-g` – sortiraj prema općim brojčanim oznakama.
- `-i` – ignoriraj karaktere/znakove koji nisu ispisivi (*neprintabilni*).
- `-k` **`X`** – sortiraj prema stupcu **`X`** (za ***CSV*** format datoteke).
- `-M` – sortiraj po mjesecima (godine).
- `-h` – sortiraj prema ljudima razumljivim oznaka za veličinu datoteka (kB, MB, GB, TB, …).
- `-n` – sortiraj prema brojčanim oznakama (*string numerički*).
- `-r` – sortiraj obrnutim redoslijedom (reverzno).

Slijedi nastavak ispisa nekih od osnovnih prekidača ove naredbe:

- `-t'X'` – promijeni separator (*delimiter)* (za ***CSV*** format) u znak **`X`**.
- `-u` – sortiraj te prilikom sortiranja izbaci duplicirane retke (linije), ako ih ima.
- `-V` – sortiraj prema logici tzv. prirodnog sortiranja (brojeva unutar teksta).

Sada pogledajmo i nekoliko primjera; ali prvo pogledajmo sadržaj datoteke `/etc/passwd` (skraćen na prvih par redova):

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
```

U ovoj datoteci (ona je u [CSV](CSV) formatu), svaki stupac, u svakom redu odvojen je sa znakom `:`. U prvom stupcu su imena korisnika (`root`, `daemon`, `bin`, `sys`,…) u drugom je **`x`** (ne koristi se), a u trećem je ***UID*** broj korisnika (`0`,`1`,`2`,`3`…) i tako dalje. Mi ćemo sada pomoću naredbe `sort` sortirati riječi po abecedi, prvog stupca odnosno prve riječi (koja u ovoj datoteci konkretno označava ime korisnika).

Naime `sort` standardno sortira svaki redak teksta po abecedi, počevši od prve riječi u tekstu; pa ćemo dobiti sljedeće (skratili smo ispis):

```
sort /etc/passwd

bin:x:2:2:bin:/bin:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
games:x:5:60:games:/usr/games:/bin/sh
root:x:0:0:root:/root:/bin/bash
. . .
```

Sada probajmo definirati da je *delimiter* znak `:` te ćemo sve sortirati prema trećem stupcu odnosno prema ***UID*** broju korisnika:

```
sort -t ':' -k 3 /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
. . .
```

Vidimo kako se sve uredno sortiralo prema trećem stupcu koji predstavlja ***UID*** broj korisnika.

Probajmo ispisati sadržaj direktorija s naredbom `ls` te ispis sortirajmo numerički prema drugom stupcu koji označava broj simboličkih poveznica na datoteke (*simbolički linkovi*):

```
ls -al | sort -nk2
```

U slučaju kada trebamo ispisati veličinu svih datoteka, primjerice unutar vršnog direktorija (mape) `/home` unutar kojeg se nalaze svi korisnički direktoriji i njihove datoteke, to možemo učiniti s naredbom `du`. Ali ako ih još sve želimo sortirati prema veličini; prema oznakama **k**B, MB i GB; tada to sve možemo napraviti proširenjem s naredbom `sort`, na sljedeći način:

```
du -hl /home/ | sort -h
```

Također je moguće i sortiranje više datoteka istovremeno, u jednom prolazu. Primjerice ako želimo sortirati datoteke: `datoteka-1.txt` i `datoteka-2.txt` istovremeno; to bi mogli napraviti ovako:

```
sort datoteka-1.txt datoteka-2.txt
```

**Izvori informacija: (914),(915), `man sort`.**

## 5.8. Usporedba sadržaja datoteka

**Slijedi napredna cjelina!.** U ovoj cjelini upoznat ćemo se s naredbama koje možemo koristiti za uspoređivanje sadržaja dvije (ili više) tekstualnih datoteka. Neki od tih programa su:

- `comm` – program za jednostavnu usporedbu sadržaja dvije sortirane datoteke, redak po redak.
- `diff` – program za usporedbu sadržaja dvije datoteke, redak po redak.
- `vimdiff` – program za usporedbu sadržaja dvije datoteke, redak po redak (on dolazi u paketu s programom `vim`).

Sa sva tri program uspoređivati ćemo dvije datoteke (`file1` i `file2`) koje imaju sljedeći sadržaj:

| Prva datoteka: **file1** | Druga datoteka: **file2** |
|---|---|
| aa | cc |
| bb | zz |
| cc | dd |
| dd | xx |
| ee | yy |

### Program `comm`

Program `comm` je prvi i možda najjednostavniji od navedenih jer standardno vrlo brzo uspoređuje datoteke koje su po mogućnosti već sortirane, ali ne nužno. Usporedimo navedene dvije datoteke i pogledajmo što ćemo dobiti:

```
comm file1 file2

aa
bb
                cc
dd
ee
        zz
comm: file 2 is not in sorted order
        dd
        xx
        yy
```

U ispisu imamo tri stupca (ispis smo zatamnili):

- Prvi stupac pokazuje sadržaj prve datoteke (`file1`) pa u njoj imamo: `aa` i `bb` te prazno mjesto pa: `dd` i `ee` pa slijedi poruka kako dvije datoteke nisu sortirane (`not in sorted order`).
- Drugi stupac prikazuje samo ono što postoji u drugoj datoteci (`file1`), ali ne i u prvoj.
- U trećem stupcu se navode oni redci koji su zajednički za obje datoteke, pa tu imamo redak: `cc`

Dakle vidljivo je kako bi bilo dobro da datoteke sadrže sortirane redke za bolju usporedbu. Ova naredba ima i nekoliko prekidača odnosno opcija poput: `--nocheck-order` koja nalaže naredbi da ne ispisuje poruku o nepravilno sortiranom sadržaju. Korisna je i opcija: `--output-delimiter=`**`STR`** kojom se definira *delimiter* (**`STR`**) stupaca.

### Program `diff`

Program `diff` je malo kompleksniji te vrlo dobro uspoređuje datoteke koje nisu nužno sortirane.
Usporedimo navedene dvije datoteke i pogledajmo što ćemo dobiti s ovim programom:

```
diff file1 file2

1,2d0
< aa
< bb
3a2
> zz
5c4,5
< ee
---
> xx
> yy
```

Ovdje je ispis malo kompleksniji pa ćemo objasniti redak po redak:

- `1,2d0` - označava da linije `1` do `2` u prvoj datoteci trebaju biti obrisane (*Delete*=`d`) kako bi se podudarale s početkom (redak 0) druge datoteke. Potom se navode te linije odnosno redci, a to su: `aa` i `bb`.
- `3a2` - označava da treba dodati (*Append*=`a`) u redak `3`, prve datoteke, redak `2` druge datoteke (`zz`) kako bi bile dobro sortirane.
- i tako dalje.

Naredba `diff` ima cijeli nîz opcija, od kojih je korisna recimo `-y` koja nam daje ispis sličan kao prethodna naredba.
Dakle daje nam ispis u dva stupca od kojih lijevi predstavlja prvu datoteku, a desni dio drugu.

Pogledajmo i ovakav ispis koji je ipak malo logičniji:

```
diff -y file1 file2

aa                                                          <
bb                                                          <
cc                                                          cc
                                                          > zz
dd                                                          dd
ee                                                          | xx
                                                          > yy
```

Ovdje vidimo prvu datoteku s lijeve strane te kako su njena prva dva retka: `aa` i `bb` dobro sortirana. Pošto u drugoj datoteci nema takvih redaka pa s desne strane imamo znakove (`<`) za ta prva dva reda.
Potom za treći redak prve datoteke (`cc`) i druge datoteke imamo podudaranje pa se `cc` vidi i s desne strane koja označava drugu datoteku (`file2`).
Zatim u drugoj datoteci s desne strane imamo pojavljivanje retka (`zz`) kojeg nema u prvoj datoteci s lijeve strane.
Nadalje vidimo usporedbu prve datoteke (lijeva strana) i druge datoteke (desna strana).



Pogledajte i animaciju rada programa `diff`.

## Program `vimdiff`

Program `vimdiff` je još malo kompleksniji te također vrlo dobro uspoređuje datoteke koje nisu nužno i sortirane.
Program `vimdiff` dolazi u paketu s uređivačem teksta `vim` koji je zapravo novija varijanta programa `vi`, a koji je obično u današnje vrijeme već instaliran na sustavu. U većini slučajeva naredba/program `vi` je poveznica (simbolički link) na uređivač teksta `vim`. Konkretno `vimdiff` pokreće dva uređivača teksta `vim` u dva *„prozora"* u kojima uspoređuje označene datoteke.
Međutim, ako on nije instaliran možete ga instalirati na sljedeći način:

```
yum -y install vim
```

Usporedimo navedene dvije datoteke i pogledajmo što ćemo dobiti s ovim programom:

```
vimdiff file1 file2

  aa                                  |----------------------------------
  bb                                  |----------------------------------
  cc                                  |  cc
  ------------------------------------|  zz
  dd                                  |  dd
  ee                                  |  xx
  ------------------------------------|  yy
  ~                                   |  ~
```

Ovdje imamo *„grafičku"* reprezentaciju sadržava lijeve i desne datoteke koje se uspoređuju. Dakle prva dva retka prve datoteke: `aa` i `bb` ne postoje u drugoj datoteci s desne strane (`|-------`). Potom imamo redak druge datoteke koji sadrži (`zz`), ali tog sadržaja nemamo u lijevoj datoteci, stoga s lijeve strane imamo (`-------|`). Zatim slijedi redak koji imamo u obje datoteke (`dd`). Redak s lijeve strane (`ee`) i onaj s desne strane (`xx`) nemaju podudaranja pa je ovaj redak označen u drugoj boji. I na kraju imamo samo u drugoj datoteci (desna strana), samo redak koji sadrži (`yy`), a kojeg nemamo u prvoj datoteci odnosno u lijevom stupcu.



Pogledajte i animaciju rada programa `vimdiff`.



Program `vimdiff` zapravo pokreće program `vi` u pozadini i to u dva prozora, tako da sve naredbe iz programa `vi` rade i primjenjive su i ovdje.



Za detalje o radu programa `vi` odnosno `vim` pogledajte poglavlje:
**5.6. Uređivač teksta *vi*.**

**Izvori informacija:** **`man comm`**,**`man diff`**,**`man vimdiff`**.

## 5.9. Traženje datoteka

S vremena na vrijeme potrebno nam je pronaći neku datoteku (skriptu ili naredbu), prema određenom kriteriju.

U tu svrhu se koriste sljedeće naredbe za pretraživanje:

- `whereis` – naredba je koja pronalazi lokaciju naredbi (binarne datoteke), *source* datoteke ili *man* datoteke.
- `which` – naredba je koja nam daje lokaciju izvršnih datoteka (programa/naredbi).
- `locate` – za pronalazak datoteke po imenu i `updatedb` - za nadopunu baze s imenima datoteka.
- `find` – naredba je koja se koristi za razne vrste pretraživanja datoteka ili direktorija, prema mnogim kriterijima.
- `xargs` – naredba je koja proširuje način rada naredbe `find`, ali i drugih naredbi.

U sljedećim cjelinama, upoznat ćemo se sa svakom naredbom pojedinačno, kroz primjere koji slijede.

---

### 5.9.1. Naredba *whereis*

Naredba `whereis` koristi se kada želimo pronaći gdje se u strukturi direktorija koji su nam u putanji preko sistemske `PATH` varijable, te u putanji za `man` stranice, nalazi tražena naredba ili datoteka.

Prvo pogledajmo koji direktoriji su nam u putanji (`PATH`), za trenutno prijavljenog (*logiranog*) korisnika odnosno iz kojih direktorija sustav može pokretati programe i naredbe:

```
echo $PATH

/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/ucenik/bin
```

Dakle u svim i iz svih navedenih direktorija (mapa) odvojenih dvotočkom `:` će se moći pokretati naredbe bez da ulazimo u njih, a i unutar njih će naredba `whereis` raditi pretraživanje.

**Pogledajmo jedan primjer**
Pronađimo gdje se nalazi naredba `ls` i pripadajuće upute (*manual*) za nju:

```
whereis ls

ls: /bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
```

Vidimo kako se naredba `ls` nalazi u direktoriju: `/bin/` te kako se upute (*manual*) za nju (jer isto u imenu datoteke imaju ***ls***), nalaze u datotekama: `/usr/share/man/man1/ls.1.gz` i `/usr/share/man/man1p/ls.1p.gz`

Pogledajmo i osnovne prekidače naredbe `whereis`:

- `-b` – pretražuj samo binarne datoteke.
- `-m` – pretražuj samo datoteke s uputama (*man stranice*).
- `-s` – pretražuj samo datoteke s izvornim (programskim kôdom).

Sumirajmo osnovnu funkcionalnost naredbe `whereis`:

- Naredba `whereis` pretražuje unaprijed definirane direktorije za binarnim datotekuma, izvornim kodom i stranicama s uputama (tz. man stranicama).
- Ispisuje putanju do pronađenih datoteka.
- Ima opcije za filtriranje rezultata i prikazivanje dodatnih informacija.

**Izvor informacija: `man whereis`.**

---

## 5.9.2. Naredba *which*

Naredba `which` koristi se kada želimo pronaći gdje se točno u putanji direktorija nalazi određena izvršna naredba i na što pokazuje njen `alias` ako on postoji.

**Slijede primjeri**
**1.** Pronađimo gdje se nalazi naredba `more` odnosno odakle ju zapravo pozivamo tj. pokrećemo:

```
which more

/bin/more
```

Vidimo da se naredba `more` nalazi u direktoriju (mapi): `/bin/more` .

**2.** Pronađimo gdje se nalazi naredba `less` odnosno odakle ju zapravo pozivamo tj. pokrećemo:

```
which less

/usr/bin/less
```

Ovdje vidimo da se naredba `less` nalazi u direktoriju (mapi): `/usr/bin/less`

**3.** Pronađimo gdje se nalazi naredba `ls` odnosno odakle ju zapravo pozivamo:

```
which ls

alias ls='ls --color=auto'
        /bin/ls
```

Što znači ispis koji smo dobili?

- Znači kako se naredba `ls` nalazi u `/bin/` direktoriju, pa je ona vidljiva kao `/bin/ls`
- Međutim to znači i da naredba `ls` ima `alias` koji presreće izvornu naredbu `ls` odnosno poziva ju s određenim prekidačem, u našem konkretnom slučaju ju pokreće ovako:
  `ls --color=auto`

**Izvor informacija:** **`man which`**.

---

## 5.9.3. Naredbe *locate* i *updatedb*

Naredba `locate` koristi se za brzo traženje datoteka. Njena brzina je u tome što se pomoću naredbe `updatedb` skeniraju datoteke na cijelom tvrdom disku (hard disk) te se njihov naziv i putanja odnosno lokacija unutar direktorija, spremaju u posebnu bazu podataka: takozvanu "***mlocate***" bazu, koja se obično nalazi u datoteci: `/var/lib/mlocate/mlocate.db`.
Definicija koje vrste datoteka (uz ostale parametre) će se pretražiti pokretanjem nadogradnje ove baze tj. pokretanjem naredbe `updatedb` je definirana u datoteci: `/etc/updatedb.conf`. Naime nakon pokretanja naredbe `updatedb` skenira se cijeli disk odnosno sve datoteke na njemu te se obnavlja **`mlocate`** baza podataka koju zapravo naredba `locate` i pretražuje tijekom potrage za datotekama. Često se na raznim distribucijama Linuxa naredba `updatedb` pokreće automatski svaki dan (noć) ili svakih nekoliko dana, kako bi **`mlocate`** baza bila što točnija.
Ova baza i navedene naredbe dolaze u softverskom paketu `mlocate`. Stoga, ako ih nemate, možete ih instalirati sa:

```
yum -y install mlocate
```

**Slijede primjeri**

**1.** Kreirajmo novu **`mlocate`** bazu podataka skenirajući cijeli tvrdi disk za novim datotekama.
Ova operacija može potrajati prilično dugo. Dakle pokrenimo naredbu `updatedb`:

```
updatedb
```

**2.** Pronađimo datoteku imena koje počinje sa `sysctl.conf`

```
locate sysctl.conf

/etc/sysctl.conf
/usr/share/man/man5/sysctl.conf.5.gz
```

Vidimo da nam je sustav pronašao dva kandidata datoteke: `/etc/sysctl.conf` i `/usr/share/man/man5/sysctl.conf.5.gz`.

**3.** Pronađimo gdje se sve nalazi datoteka imena `rc.local`

```
locate rc.local

/etc/rc.local
/etc/rc.d/rc.local
```

I ovdje vidimo rezultate pretrage u dvije pronađene datoteke: `/etc/rc.local` i `/etc/rc.d/rc.local`.

**4.** Pronađimo koje sve datoteke imaju ime koje počinje s riječi `apache`

```
locate apache

/etc/selinux/targeted/modules/active/modules/apache.pp
/usr/lib/python2.6/site-packages/sos/plugins/apache.py
/usr/lib/python2.6/site-packages/sos/plugins/apache.pyc
/usr/lib/python2.6/site-packages/sos/plugins/apache.pyo
/usr/share/selinux/devel/include/services/apache.if
/usr/share/selinux/targeted/apache.pp.bz2
/usr/share/vim/vim72/syntax/apache.vim
/usr/share/vim/vim72/syntax/apachestyle.vim
```

Osim navedenih primjera postoje i neki prekidači naredbe `locate`, za dodatne osnovne opcije pretraživanja. Međutim za sva složenija pretraživanja, obično se koristi naredba `find` koju ćemo upoznati u slijedećoj cjelini.

**Izvori informacija:** **`man updatedb`**, **`man locate`**.

---

## 5.9.4. Naredba *find*

Naredba `find` koristi se za pronalaženje datoteka prema raznovrsnim parametrima pretraživanja, kojih naredba `find` ima na stotine. Njena osnovna sintaksa je:

`find /početni-direktorij-za-pretraživanje` **`-opcije`** `ono-što-tražimo`

Osnovne opcije naredbe `find` za pretraživanje su neke od navedenih (navodimo ih samo nekoliko):

- `-name` – za ime datoteke, dok `-iname` koristimo za ime datoteke kada se ignoriraju velika i mala slova.
- `-type f` – označava datoteku, dok `-type d` označava direktorij, a `-type l` označava simbolički link.
- `-inum` **`XY`** – pronalazi datoteke s određenim *INODE* brojem (isti *inode* broj ukazuje na *hard link* datoteke).
- `-perm` – označava da slijede ovlasti (*permisions*) (pr. pronađi sve s ovlastima 777: `-perm 777`).
- `-exec` – označava da nakon ove opcije slijede naredbe iz operativnog sustava.
- `-mtime` – slijedi oznaka vremena kada je (datoteka ili direktorij) modificiran (pr. zadnjih 10 dana: `-mtime 10`).
- `-atime` – slijedi oznaka vremena kada je (datoteci ili direktoriju) pristupano (*read* ili *copy*).
- `-ctime` – slijedi oznaka vremena kada je datotekama mijenjan status.
- `-size` – slijedi veličina datoteke (pr. veće od 1.GB: `-size +1G` ili manje od 1GB: `-size -1G`).
- `-user` – slijedi ime korisnika za kojeg se traži (što već tražimo), `-group` - slijedi ime grupe za koju se traži…
- `-delete` – briši sve pronađeno (datoteke ili direktorije).
- `-xdev` – u potrazi **ne** prelazi na drugi datotečni sustav, ako je montiran na postojeći direktorij (unutar onog koji gledamo ).

Moguće je koristiti i negaciju odnosno znak `!`, kao i kombinacije navedenih (i drugih) opcija i prekidača.

Primjerice pronađimo sve datoteke koje nemaju ovlasti **`777`**, unutar direktorija `/home`, a veće su od 1GB:

```
find /home -type f ! -perm 777 -size +1G
```

**Slijede dodatni primjeri**

**1.** Pomoću naredbe `find` pronađimo datoteku imena `rc.local` za koju znamo da se nalazi negdje u stablu `/etc/` direktorija, pa nećemo pretraživati cijeli tvrdi disk počevši od vršnog `/` direktorija već počevši od direktorija (mape): `/etc/`

```
find /etc/ -name rc.local

/etc/rc.d/rc.local
/etc/rc.local
```

Vidimo da je tražena datoteka `rc.local` pronađena u dva direktorija: `/etc/rc.d/rc.local` i `/etc/rc.local`.

**2.** Već se nalazimo u `/etc/` direktoriju, te krenimo s istom pretragom iz prethodnog primjera, ali označivši da krećemo iz trenutnog direktorija, sa znakom `.` koji u UNIX/Linux svijetu označava trenutni direktorij:

```
find . -name rc.local

./rc.d/rc.local
./rc.local
```

**3.** Pronađimo istu datoteku iz prethodnog primjera, ali s time da se ignoriraju velika i mala slova, jer nismo sigurni koje je veliko, a koje malo slovo u nazivu datoteke:

```
find . -iname rc.local

./rc.d/rc.local
./rc.local
```

**4.** Pronađimo direktorij imena `ssh`, ali sada želimo pretražiti cijeli disk počevši od vršnog direktorija (`/`) jer ne znamo gdje se točno traženi direktorij nalazi:

```
find / -type d -name ssh

/etc/ssh
```

**5.** Pronađimo datoteku unutar `/etc/` direktorija (i svih njenih poddirektorija), a koja ima ekstenziju `.sample`

```
find /etc/ -type f -name "*.sample"

/etc/gdm/PostLogin/Default.sample
```

**6.** Pretražimo korisničke `/home` (pod)direktorije u potrazi za datotekama koje imaju ovlasti `777`. Dakle tražimo datoteke koje imaju ovlasti: *`Read+Write+eXecute`* (**`rwx`**), postavljene za sve korisnike (vlasnik, grupa i ostali):

```
find /home/ -type f -perm 0777 -print
```

**7.** Pronađimo sve *MP3* datoteke u korisničkim *home* direktorijima i obrišimo ih:

```
find /home/ -type f -name "*.mp3" -exec rm -f {} \;
```

Opis: ovdje pozivamo prekidač `-exec` s kojim pozivamo vanjsku naredbu od operativnog sustava: `rm -f` s dodatnim parametrima.

U ovom primjeru, ako je lista pronađenih datoteka prevelika, dobiti ćemo grešku:

```
/usr/bin/find: Argument list too long
```

Tada pogledajte sljedeće poglavlje ili primjer (**7.1**). Naime u ovom primjeru, lista svih pronađenih datoteka; primjerice: `pjesma1.mp3`, `pjesma2.mp3` i druge, će se proslijediti vanjskoj naredbi, unutar vitičastih zagrada: `rm -f {}`.

Tada će naredba `rm -f` biti pozvana s cijelom listom pronađenih datoteka, pa će to izgledati ovako: `rm -f pjesma1.pm3 pjesma2.mp3`. U slučaju kada imamo tisuće ili stotine tisuća pronađenih datoteka, njihova lista će biti isto tako proslijeđena naredbi `rm` i to sve u jednom retku (liniji), što će uzrokovati problem, jer dužina svakog naredbenog retka (linije) nije beskonačna. Stoga ovo nije najbolja metoda u slučajevima kada očekujemo veliki broj pronađenih datoteka.

**7.1** Isto smo mogli napraviti i s prekidačem `-delete`

```
find /home/ -type f -name "*.mp3" -delete
```

Ovdje ne bi smjelo biti problema vezanih za dužinu liste pronađenih datoteka, a dodatni primjer za istu stvar pogledajte u sljedećem poglavlju.

**8.** Pronađimo sve prazne datoteke u *home* direktoriju svih korisnika:

```
find /home/ -type f -empty
```

**8.1.** Pronađimo sve prazne direktorije, također u *home* direktoriju svih korisnika.

```
find /home/ -type d -empty
```

**9.** Pronađimo sve datoteke koje su promijenjene (*modify*) u zadnjih 10 dana, u *home* direktoriju svih korisnika.

```
find /home/ -mtime 10
```

**10.** Pronađimo sve datoteke kojima je pristupano (*access*) u zadnjih 10 dana, u *home* direktoriju svih korisnika. Pristupano znači da su te datoteke pročitane (*read*) ili kopirane.

```
find /home/ -atime 10
```

**11.** Pronađimo sve datoteke u *home* direktoriju (**`/home/`**) svih korisnika, koje su veće od 100 MB.

```
find /home/ -size +100M
```

**12.** Pronađimo sve datoteke u *home* direktoriju (**`/home/`**) svih korisnika, koje su veće od 50MB, ali manje od 100MB.

```
find /home/ -size +50M -size -100M
```

13. Potraga za *Inode* brojevima datoteka (što u slučaju kada dvije datoteke imaju isti *Inode* broj ukazuje da su *hard link* povezane datoteke). Ovdje potragu radimo unutar direktorija `/root/`:

```
find /root/ -inum 4215621
/root/test/test.txt
/root/test/test-hard-link.txt
```

14. Potraga za datotekama koje imaju postavljen ***SUID*** bit:

```
find / -perm /u=s
```

15. Potraga za datotekama koje imaju postavljen ***SGID*** bit:

```
find / -perm /g=s
```

**Izvori informacija:** **`man find`**,**`info find`**.

### 5.9.5. Naredba *xargs* i ograničenje duljine argumenata

***Slijedi napredna cjelina!***
Naredba `xargs` nije vezana za pretraživanje, ali se često koristi u kombinaciji sa naredbom `find` ili drugim naredbama.

**Zbog čega i kada koristiti `xargs`?**
Linux kao i svaki drugi operativni sustav ima određena ograničenja prema broju *file deskriptora,* ali i na veličinu memorije koja je u slučaju Linuxa rezervirana za broj odnosno količinu argumenata koje svaka naredba može pohraniti u memoriji. Standardno Linux na razini cijelog sustava definira maksimalnu veličinu (duljinu) memorije koja se može koristiti za nîzanje naredbi, kao i za argumente koje možemo poslati nekoj od naredbi.

Maksimalno ograničenje na duljinu reda (ili argumenata) možemo dobiti s naredbom `getconf` na sljedeći način:

```
getconf ARG_MAX
```

Vrijednost varijable `ARG_MAX` je u ovisnosti o veličini maksimalnog memorijskog stôga, a koju možemo vidjeti s naredbom `ulimit` na sljedeći način:

```
ulimit -s
```

Točan izračun koji sustav koristi za postavljanje ***POSIX*** varijable: `ARG_MAX`=$\frac{vrijednost\ od\ \textbf{ulimit-s}\ \times 1024}{4}$.



Vezano za sistemske varijable, pogledajte poglavlje:
**6.2.2.1. Posebne sistemske varijable.**



Vezano za **POSIX**, pogledajte i poglavlje:
**10.4.4.1. POSIX standard.**

Pri tome nam je dostupno nešto manje od ovdje vidljive vrijednosti, zbog toga što je potrebno oduzeti i određenu vrijednost koja se uzima za postavljanje i čitanje varijabli okruženja (*environment*) sustava.
Dostupnu dužinu argumenata možemo vidjeti i s naredbom: `xargs --show-limits`



*Važno je razumjeti kako je ograničenje postavljeno ovdje, ograničenje na ukupnu dužinu, a ne na broj argumenata ili nîz naredbi.*

Vratimo se na naredbu `xargs` koja radi tako da uzima sve sa standardnog ulaza (*standard input* [ `stdin` ]) te lomi dugačke ulazne podatke na više manjih odnosno kraćih, a koje potom prosljeđujemo nekoj drugoj naredbi. Na taj način se rješava problem ograničenja navedenih u tekstu gore, bez potrebe za promjenama na razini Linux *file descriptora (opisnika datoteka).*

Dakle ovo se događa samo u slučajevima kada nam na ulaz (ili izlaz naredbe prije) dolazi velika količina podataka ***u nîzu***, te kada obično dobijemo sistemsku pogrešku poput: `Argument list too long`



Ovaj problem u praksi se događa samo kod slučajeva kada se barata s vrlo velikom količinom argumenata koje želimo proslijediti nekoj naredbi.

Na primjeru **7.** iz naredbe `find`, gdje sve pronađene `.mp3` datoteke iz `/home/` direktorija šaljemo u listu `{}` te pozivamo vanjsku naredbu u koju se puni ta lista, možemo doći do tog problema. Naime u slučaju kada je pronađeni broj takvih datoteka izuzetno ili ekstremno velik, te ne stane u listu koju prosljeđujemo naredbi `rm -f`, mogli bi dobiti već spomenutu grešku:

```
Argument list too long
```

Ponovno pronađimo sve *MP3* datoteke u korisničkim *home* direktorijima i obrišimo ih, na stari (potencijalno problematičan) način:

```
find /home/ -type f -name "*.mp3" -exec rm -f {} \;
```

**1.** U našem novom primjeru prilagodit ćemo se upotrebom naredbe `xargs` s kojom ćemo pokrenuti naredbu `rm -f` s listom `{}` imena `.mp3` datoteka koju nam je pronašao i popunio `find`, pa nećemo moći doći do navedenog ograničenja:

```
find /home/ -type f -name "*.mp3" -print | xargs rm -f {} \;
```

Istu stvar možemo napraviti i bez liste, već slanjem imena datoteka od `find`-a kroz `xargs` prema naredbi `rm -f` i to ovako:

```
find /home/ -type f -name "*.mp3" -print | xargs rm -f
```

Pogledajmo i što nam sve `find` pronalazi te što `xargs` prosljeđuje: šaljemo sve pronađeno na ispis na ekran sa `echo`:

```
find /home/ -type f -name "*.mp3" -print | xargs echo

/home/ucenik/1.mp3 /home/ucenik/3.mp3 /home/ucenik/2.mp3
```

**1.1** U slučaju kada bismo imali neku datoteku koja ima razmak (*space*) u imenu, `find` ju ne bi pronašao niti bi ju `xargs` proslijedio na brisanje. Riješimo ovaj problem u `find`-u, dodavanjem `-print0` što znači kako se ispisuju i one datoteke koje imaju razmak u imenu te isto recimo i `xargs`-u (`-0`), kako bi i on to prihvatio.

Pogledajmo prvo listu `.mp3` datoteka:

```
ls -al *.mp3

-rw-r--r-- 1 root root 0 Jul 24 18:42 0.mp3
-rw-r--r-- 1 root root 0 Jul 24 18:42 1.mp3
-rw-r--r-- 1 root root 0 Jul 24 18:42 2.mp3
-rw-r--r-- 1 root root 0 Jul 24 18:42 3 0.mp3
-rw-r--r-- 1 root root 0 Jul 24 18:29 3.mp3
```

Vidimo kako postoji jedna datoteka koja se zove `3 0.mp3` dakle ima razmak u imenu, pa ćemo koristiti metodu naučenu gore:

```
find /home/ -type f -name "*.mp3" -print0 | xargs -0 rm -f
```



Vezano za naredbu `find`, pogledajte poglavlje:
**5.9.4. Naredba *find*.**

**2.** Ispišimo niz brojeva u jednom retku, na sljedeći način:

```
echo 1 2 3 4 5 6 7 8


1 2 3 4 5 6 7 8
```

A sada iskoristimo mogućnost ograničavanja maksimalnog broja elemenata u svakom redu, naredbe `xargs` (prekidač `-n`), te ograničimo ispis na dva elementa po redu:

```
echo 1 2 3 4 5 6 7 8 | xargs -n2

1 2
3 4
5 6
7 8
```

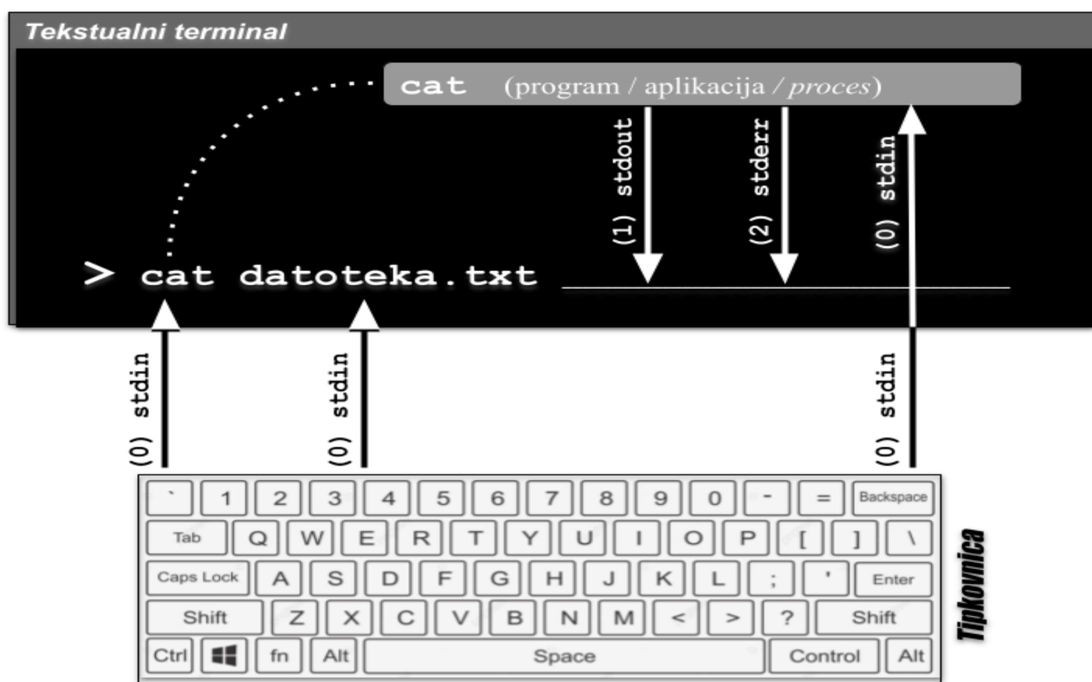
**Izvori informacija: (590),(591),(592),(593), `man xargs`, `man find`, `getconf --help`.**

# 5.10. Preusmjeravanje (*redirekcija*) i Pipe funkcionalnost

Tijekom standardnog izvršavanja nekog programa (pr. `cat`), ulazni podaci obično se unose s tipkovnice, a izlazni rezultat njegove obrade se ispisuju na naš radni tekstualni *terminal* odnosno na monitor računala (naredbeni redak). Poruke o greškama se standardno ispisuju također na *terminal*, kao na slici 15. Klasičan primjer bi bio slučaj u kojem naš program treba učitati neke podatke iz datoteke, obraditi ih i ispisati rezultat na naš *terminal* odnosno monitor (*ekran*). U Linuxu nakon instalacije operativnog sustava dobivamo na tisuće korisnih programa, poput cijelog nîza programa za filtriranje teksta: prema ključnoj riječi, retku ili stupcu i drugim kriterijima, kao i zamjenu dijelova teksta s drugim tekstom i slično, te raznih specijaliziranih programa za druge namjene. Stoga se pojavljuje nekoliko potreba: jedna od njih je preusmjeravanje odnosno *redirekcija*, kod koje ispis (izlaz) odnosno rezultat obrade jedne naredbe (programa) preusmjeravamo obično u neku datoteku, koju potom možemo obraditi s drugim programom ili s njom raditi što već želimo. Preusmjeravanje (*redirekcija*), se radi pomoću meta znakova `<<` `<` `>` i `>>`
Po pitanju preusmjeravanja je važno razlikovati nekoliko pojmova:

- ***Standardni ulaz*** odnosno `stdin` (`0`) – predstavlja ulaz podataka koji naš proces ili naredba (program) čitaju kao ulazne podatke. Ulazni podaci mogu biti primjerice s tipkovnice ili mogu dolaziti iz drugog programa odnosno procesa.
- ***Standardni izlaz*** odnosno `stdout` (`1`) – predstavlja izlaz podataka iz procesa odnosno programa. Ovi podaci se obično šalju na naš izlazni uređaj odnosno terminal (*Monitor/Ekran [naredbeni redak]*) ili prema nekom drugom programu.
- ***Standardna greška*** odnosno `stderr` (`2`) – na nju se šalju sve poruke o greškama svakog pojedinog procesa (programa).

**Slika 15. Standardni ulaz i izlaz te standardna greška.**




Standardno ponašanje programa prilikom kojega program rezultat svoje obrade šalje na terminal (monitor/ekran) može se promijeniti upotrebom preusmjeravanja odnosno **redirekcije**, o kojem ćemo pričati u ovom poglavlju.
Na najnižoj razini, za tu namjenu postoji nekoliko *file deskriptora* za svaki proces, i to po jedan za standardni ulaz (`stdin`), jedan za standardni izlaz (`stdout`) i jedan za standardne greške (`stderr`).



Za više detalja o file deskriptorima pogledajte poglavlje:
**4.5.4. Opisnici *datoteke* (*File deskriptori* ).**

Ako stvari promatramo s razine programa (procesa), tada možemo reći sljedeće:

- Kada je potrebno *učitati* podatke u program, koristi se `stdin` odnosno standardni ulaz prema programu.
- Kada program treba ispisati izlazne podatke "van" (na terminal), oni se šalju na `stdout` odnosno na standardni izlaz.
- Kada program mora ispisati neke poruke o greškama u radu ili obradi podataka, ove poruke se šalju na `stderr` odnosno ispisuju se kao poruke za standardne greške.

Ako želimo koristiti preusmjeravanje, ono se uobičajeno radi pomoću meta znakova `>` i `>>`. Pomoću ovih meta znakova zapravo prosljeđujemo podatke koji bi inače završili na standardnom izlazu, a to je ono što završava ispisano na teminalu (ekranu), negdje drugdje; obično u neku datoteku.
Drugi način preusmjeravanja jest preusmjeravanje pomoću meta znakova `<` i `<<`. S ovim meta znakovima radimo preusmjeravanje sa standardnog ulaza. Standardni ulaz može biti i datoteka, pa u tom slučaju možemo raditi preusmjeravanje iz datoteke prema nekoj naredbi i slično. Mada se češće za te potrebe koristi "*Pipe*" funkcionalnost (o njoj malo kasnije).

**Izvor informacija: `man bash`.**

## 5.10.1. Preusmjeravanje (*redirekcija*)

Preusmjeravanje je metoda s kojom određujemo kamo se šalju ulazi ili izlazi vaših naredbi. Prema zadanim postavkama, naredbe primaju podatke sa standardnog ulaza (pr. tipkovnice) i zatim izlazne rezultate šalju na standardni izlaz koji je uobičajeno terminal odnosno monitor/ekran računala.
Preusmjeravanje je najkorisnije pri radu s naredbama i datotekama. Moguće je, na primjer, preusmjeriti izlaz naredbe u datoteku umjesto ispisa izlaza na terminal. Alternativno, moguće je definirati i željenu datoteku kao ulaz u naredbu.

**Krenimo s primjerima**
Preusmjerimo izvršenu naredbu odnosno njen ispis na standardni izlaz u neku datoteku, konkretno: preusmjerimo izlistanje direktorija pomoću naredbe: `ls -al` u datoteku, prema principu: **naredba → datoteka.txt**

```
ls -al > datoteka.txt
```

Kada koristimo standardno preusmjeravanje (`>`) sadržaj datoteke u koju preusmjeravamo ispis neke naredbe, bit će svaki puta prebrisan, a potom popunjen novim sadržajem. Odnosno ovo je popunjavanje datoteke nekim sadržajem počevši od početka datoteke. Pogledajmo i preusmjeravanje u drugom smjeru: iz datoteke u naredbu. Dakle sada želimo ispisati sadržaj datoteke koju smo kreirali u koraku prije, ali pomoću preusmjeravanja sa standardnog ulaza. Za ispisivanje sadržaja datoteke ćemo koristiti naredbu `cat`.
Stoga napravimo ovo preusmjeravanje u drugom smjeru, što nije baš uobičajeno, ali je moguće:

```
cat  < datoteka
```

Ako ipak želimo koristiti preusmjeravanje na standardni izlaz, ali želimo nadodati svoj (novi) sadržaj na kraj datoteke, tada moramo koristiti dvostruko preusmjeravanje odnosno znak `>>` koji radi upravo to; **nadodaje** novi sadržaj na stari sadržaj:

```
ls -al >> datoteka.txt
```

Sada preusmjerimo datoteku `file1` u naredbu `cat` koja će ispisati tu datoteku na standardni izlaz (terminal/*ekran*), te ćemo to preusmjeriti u datoteku `file2`. I ovakva metoda rada nije baš uobičajena, ali je moguća:

```
cat < file1 > file2
```

Dakle ulaz naredbe `cat` je datoteka imena `file1`, a izlaz se preusmjerava u datoteku imena: `file2`.



***Osim preusmjeravanja ispisa željene naredbe ili skripte u datoteku, moguće je i preusmjeravati ispis u ili iz file deskriptora.***

Sjetimo se da *file deskriptor* broj `2` označava standardnu grešku, a *file deskriptor* broj `1` standardnu izlaznu poruku programa.
Stoga je moguće primjerice naložiti trenutnom programu da sâmo sve standardne greške (*file deskriptor* `2`) zapisuje u datoteku.

```
ls -al 2> greške.pri.ispisu.txt
```

Možemo imati i potrebu standardne greške (`2`) preusmjeriti u standardne poruke (`1`) kako bi nam se sve ispisalo na jednom mjestu. Međutim preusmjeravanje u *file deskriptor* `1` (ili bilo koji drugi) moramo napraviti sa znakom `&`.

```
ls -al 2>&1
```

Naime preusmjeravanjem u sâmo broj jedan (`1`) bi se kreirala datoteka imena `1`. Stoga, pošto se radi o preusmjeravanju u *file deskriptor*, moramo naznačiti sustavu da to radimo prema *file deskriptorima*, koji se označavaju upravo posebnim znakom `&`.

Moguće je raditi preusmjeravanje i u posebne datoteke u `/dev/` direktoriju.

Primjerice preusmjeravanje svih grešaka (`2`) programa preusmjerimo u posebni uređaj (`/dev/null`) koji poput crne rupe guta sve ulazne podatke i ne prikazuje ih nigdje.

```
ls -al 2> /dev/null
```

Pogledajmo sve dostupne operatore za usmjeravanje i primjere preusmjeravanja:

- **`>`** - preusmjeravanje u željenu datoteku (s brisanjem postojećeg sadržaja).
- **`2>`** - preusmjeravanje poruka o greškama (`stderr`) u željenu datoteku (s brisanjem postojećeg sadržaja).
- **`>>`** - preusmjeravanje u željenu datoteku s time da će se novi sadržaj nadodati u datoteku.
- **`2>>`** - preusmjeravanje poruka o greškama (`stderr`) u željenu datoteku (s dodavanjem na postojeći sadržaj).
- **`<`** - preusmjeravanje sadržaja datoteke u naredbu (za izvršavanje).
- **`&>`** - preusmjeravanje standardnog izlaza (`stout`) i standardnih grešaka (`stderr`) u željenu datoteku (s brisanjem postojećeg sadržaja).
- **`&>>`** - preusmjeravanje standardnog izlaza (`stout`) i standardnih grešaka (`stderr`) u željenu datoteku (s dodavanjem na postojeći sadržaj).

**Izvori informacija: (675),(676),(817),(K-1),(K-12), `info pipe`,`man 7 pipe`.**

## 5.10.2. Pipe funkcionalnost za povezivanje naredbi

Engleski izraz *pipe* (cijev) izveden je iz riječi *pipeline* (cjevovod) koji se odnosi na svojevrsnu cijev koja predstavlja tok podataka koji teče između dva programa (*procesa*) koji su ili blisko povezani ili nemaju zajedničko podrijetlo. To znači da izlazni rezultat jednog programa služi kao ulaz drugom programu. Između ostalog, to nam omogućuje da veće zadatke podijelimo na manje podzadatke radi boljeg pregleda i kasnijeg razumijevanja. S obzirom da na Unix ili Linux operativnim sustavima uvijek imamo predinstalirano na tisuće malih, ali vrlo brzih i nevjerojatno optimiziranih programa, goto svaki zadatak možemo razložiti na manje segmente koje može odraditi po jedan od tih programa. Jedino što zatim trebamo napraviti jest nekako ih povezati.

Vidjeli smo kako se izlaz obrade naredbe/programa može preusmjeriti u neku datoteku, zbog prikaza rezultata ili zbog dalje obrade u nekom drugom programu. Međutim Unix/Linux nam nudi i takozvanu *Pipe* funkcionalnost, koju predstavlja znak **|** koja preusmjerava standardni izlaz jedne naredbe u standardni ulaz druge, bez potrebe za korištenjem privremenih datoteka.

To znači da praktično prosljeđujemo podatke koje je obradila jedna naredba ili program, na obradu drugoj naredbi koja ih, nakon što obradi podatke, šalje na obradu sljedećoj naredbi i tako dalje. S ovom funkcionalnosti povezujemo naredbe pomoću znaka **|** odnosno *pipe-a* to jest "*cijevi*" poveznice te tako stvaramo potpuno nove mogućnosti, prema našim potrebama.

U primjeru ćemo u prvom koraku napraviti preusmjeravanje ispisa direktorija u datoteku, a u drugom koraku ćemo ispisati sadržaj te datoteke i prebrojati koliko riječi sadrži ta datoteka.
Za prebrojavanje riječi ćemo koristiti naredbu `wc` (engl. *Word count*) te ćemo ju pozvati s prekidačem `-l`, kako bi nam prikazala broj riječi u ispisu (što ovdje zapravo znači broj datoteka).

Prvo preusmjerimo ispis direktorija u datoteku imena `datoteka.txt`:

```
ls > datoteka.txt
```

A sada korištenjem *pipe* (`|`) funkcionalnosti ispišimo tu datoteku i prebrojimo riječi u njoj:

```
cat datoteka.txt | wc -l

17
```

Dakle broj riječi je `17` što znači kako ukupno imamo `17` datoteka u direktoriju (mapi) u kojoj se nalazimo.

Ovdje smo s naredbom `cat` ispisali sadržaj datoteke `datoteka.txt` odnosno poslali ga na *standardni izlaz (*`stdout`*)*, koji je znak pipe **|** proslijedio sljedećoj naredbi kao njen ulaz odnosno "*standard input*" (`stdin`).
Potom ga je naredba `wc` uzela i obradila, te nam na kraju dala rezultat svoje obrade.
Sada ćemo istu stvar iz prethodnog primjera još više pojednostaviti:

```
ls | wc -l

17
```

Ovdje je još jasnije vidljiva *pipe* funkcionalnost:

- Naredba `ls` ispisuje sadržaj trenutnog direktorija i šalje ga na standardni izlaz, ali taj standardni izlaz nikada neće završiti na terminalu (monitoru/ekranu) jer ga dohvaća pipe funkcionalnost (`|`) i uvlači ga u sebe, kako bi ga proslijedila sljedećem programu u nîzu, na njegov standardni ulaz (`stdin`).
- Sada se iz pipe funkcionalnosti (`|`) preuzeti nîz podataka šalje na naredbu `wc`, na njen standardni ulaz (`stdin`) te ga ona prihvaća, uzima te podatke, obrađuje ih i tek tada rezultat svoje obrade šalje na standardni izlaz *(*`stdout`*)* odnosno na naš terminal odnosno ekran/monitor (*naredbeni redak*).



Za više detalja o *redirekciji* i *pipe* funkcionalnosti, pogledajte poglavlje:
**9.4 Komunikacija između procesa**, odnosno konkretno cjelinu:
**9.4.1 Unix Pipes**.

Pogledajte i cjelinu:
**6.7. Nizanje i ulančavanje naredbi i statusni kodovi.**

**Izvori informacija: (675),(676),(817),(K-1),(K-12),** **`info pipe`**, **`man 7 pipe`**.

# 5.11. Meta znakovi

Meta znakovi su nealfanumerički znakovi ili karakteri koji imaju posebno značenje u naredbenoj ljusci (*shellu*) ili drugim programima. Meta znakove često koristimo za regularne izraze. Neki od meta znakova su:

**`^ < > * ? [ ] $ & - ! ; | " ' ``**

*Wildcards* znakovi su podgrupa meta znakova i zamjenjuju alfanumeričke znakove, a to su sljedeći znakovi:

**`* ? ! [ ] -`**

## 5.11.1. Značenje nekih meta znakova

***Opis posebnih (wildcards)* znakova, koji su podskup skupa meta znakova, je sljedeći**:

- `*` označava bilo koji i bilo koliko znakova (uključujući i niti jedan od njih).
- `?` označava točno jedan, bilo koji znak.
- `[ ]` označava listu znakova, primjerice *[a-zA-Z]* (što označava sva mala slova od ***a*** do ***z***, ali i sva velika slova).

**Opis nekih meta znakova**:

- `^` označava početak reda (linije).
- `;` koristi se za pokretanje više naredbi (programa) u nîzu; koje će se pokretati jedna za drugom, ulančano.
- `&` koristi se uglavnom za pokretanje programa u pozadini (engl. *background*).



Vezano za meta znak **&** pogledajte poglavlje:
**9.5 Linux poslovi (*jobs*).**

---

# 5.12. Regularni izrazi

Regularni izrazi predstavljaju zapise koji se odnose na određeni uzorak.
Osnovni regularni izrazi su (kasnije ćemo upoznati i druge):

- `^` označava početak retka (linije teksta).
- `$` označava kraj retka, a kod varijabli, ako se nalazi ispred varijable, daje nam njenu vrijednost.
- `*` označava nula ili više ponavljanja.
- `.` označava pojedini znak (karakter).
- `\` označava kako se slijedeći znak (karakter) koji slijedi nakon ovoga, interpretira na poseban način odnosno da se prikazuje doslovno ono što slijedi, čak iako slijedi neki poseban meta znak. Sa znakom `\` se označavaju takozvane kontrolne sekvence odnosno *Escape* sekvence. Meta znak `\` se koristi za kontrolne sekvence, koje imaju posebno značenje. Pogledajmo i neke od *kontrolnih* sekvenci koje se često koriste:
  - **`\t`** označava "*Tab*" odnosno nîz od obično četiri (4) razmaka (a ponekada i više) ovisno kako je definirano.
  - **`\n`** označava kraj retka (engl. *New Line*), prema UNIX/Linux načinu zapisivanja (***EOL*** konverziji), nakon ovoga slijedi novi redak teksta.

Možemo reći kako se regularni izrazi koriste za pronalaženje i baratanje s određenim uzorcima, a osim upotrebe u ljusci (*Shellu*), koriste ih i razni programi poput: `vi`, `more`, `grep`, `sed`, `awk` ili ***notepad++***.

### Primjeri upotreba meta znaka *

Izlistajmo datoteke (naredbom `ls`) čije ime počinje sa slovom `d` nakon čega u nazivu imena datoteke može slijediti bilo što:

```
ls -al d*

-rw-r--r-- 1 root root 320 Nov 15 11:28 datoteka.txt
```

U ovom primjeru ispisali smo sadržaj trenutnog direktorija, s time da smo tražili samo datoteke čije ime počinje s malim slovom `d` nakon kojega može slijediti bilo koliko znakova (slova ili brojeva) ili niti jedan znak. To smo postigli s meta znakom **`*`** koji znači: *bilo koji i bilo koliko njih ili čak niti jedan znak*.
U našem slučaju, u ovoj potrazi je pronađena samo datoteka imena `datoteka.txt` jer počinje s malim slovom `d` i ima još neka druga slova u imenu koja slijede slovo `d`.

## Upotreba meta znaka ?

Izlistajmo datoteke čije ime počinje s `datoteka.tx` i ima još jedan bilo koji karakter (slovo ili broj):

```
ls -al datoteka.tx?

-rw-r--r-- 1 root root 320 Nov 15 11:28 datoteka.txt
```

Ovdje smo koristili meta znak **?** koji zamjenjuje samo i isključivo jedan karakter (slovo ili broj) ili niti jedan od njih.
Pošto smo željeli ispisati sve datoteke koje se zovu `datoteka.tx`, ali imaju još jedan (bilo koji) karakter nakon toga, dobili smo samo datoteku imena `datoteka.txt`.

Naš meta znak se može naravno koristiti na bilo kojem mjestu, kao na sličnom primjeru, dolje:

```
ls -al datotek?.?x?

-rw-r--r-- 1 root root 320 Nov 15 11:28 datoteka.txt
```

U ovom primjeru je vidljivo kako meta znak `?` mijenja točno jedan karakter odnosno slovo ili broj.

---

## Upotreba meta znaka .

Meta znak **.** zamjenjuje točno i isključivo samo jedan karakter (slovo ili broj) za razliku od **?** koji može služiti, za provjeru jednog karaktera ili ako ga uopće nema (prema logici: ili jedan ili niti jedan).
Primjena ovog meta znaka nije direktna u ljusci (*shellu/naredbenom retku)*, već u programima poput programa `grep`.
Pogledajmo primjer, kada tražimo bilo koji, ali točno samo jedan karakter (znak) koji nedostaje:

```
ls -al | grep dat.teka.txt

-rw-r--r-- 1 root root 320 Nov 15 11:28 datoteka.txt
```

U našem slučaju meta znak `.` u pola imena datoteke koji je pronađen je slovo "o" pa je vidljiva pronađena datoteka imena: `datoteka.txt`.

---

## Upotreba meta znaka ^

U ovom primjeru ćemo koristiti ***pipe*** **(**znak **| )** i naredbu `grep`. Osim toga potrebno je i poznavanje Linux ovlasti. Ukratko:

- Ako je prvi znak ovlasti **–** (prva oznaka s lijeve strane) tada se radi o datoteci.
- Ako je prvi znak ovlasti **d** (prva oznaka s lijeve strane) tada se radi o direktoriju (mapi).

Izlistajmo sadržaj trenutnog direktorija u kojem se nalazimo, koji izgleda ovako:

```
ls -al

total 20
drwxr-xr-x 5 root root 4096 Apr 25 15:47 .
drwxr-xr-x 3 root root 4096 Apr 25 15:39 ..
-rw-r--r-- 1 root root   51 Apr 25 15:39 datoteka.txt
drwxr-xr-x 2 root root 4096 Apr 25 15:47 primjer1
drwxr-xr-x 2 root root 4096 Apr 25 15:47 primjer2
drwxr-xr-x 2 root root 4096 Apr 25 15:47 primjer3
```

Vidljivo je da imamo tri poddirektorija te jednu datoteku. S obzirom da smo upoznati s ovlastima (engl. *Permissions*) jasno nam je da, ako je prvi znak kod ovlasti znak **–** to znači da se radi o datoteci poput naše datoteke `datoteka.txt` koja ima ovlasti:
*-rw-r-r-*
Sada ćemo pretražiti izlistani sadržaj direktorija od gore, te tražiti sve ovlasti koje počinju sa znakom **–**. Odnosno u ovom slučaju to znači sve redove, jedan po jedan, kako ih filtrira naredba `grep`, a koji počinju (oznaka početka je ^) sa znakom **–**.

```
ls -al |grep ^-

-rw-r--r-- 1 root root    0 Apr 25 15:39 datoteka.txt
```

Dobili smo očekivani rezultat; jedini red ispisa sadržaja direktorija (`ls -al`) koji počinje (ima na početku) sa – znakom, je naša datoteka (`datoteka.txt`).

U slučaju kada bi tražili koji su sve poddirektoriji unutar našeg trenutnog direktorija, tražili bi čije ovlasti počinju sa slovom `d`.

```
ls -al |grep ^d
...
drwxr-xr-x 2 root root 4096 Apr 25 15:47 primjer1
drwxr-xr-x 2 root root 4096 Apr 25 15:47 primjer2
drwxr-xr-x 2 root root 4096 Apr 25 15:47 primjer3
```

Pogledajmo sadržaj naše datoteke imena: `datoteka.txt`, koja sadrži sljedeće redove:

```
# Ovo je komentar
Ovo nije komentar

Tekst 1
Tekst 2
```

## Upotreba meta znaka #

Meta znak **#** se često koristi za pisanje komentara. Poslije njega se može pisati neki komentar ili tekst koji Linux ljuska (*Shell*) promatra kao tekst komentara, te ga ne procesira na bilo koji drugi način odnosno ne pokušava ga izvršiti i slično.
U slučaju da smo željeli pronaći sve redove unutar naše tekstualne datoteke (`datoteka.txt`), koji počinju sa znakom `#`, to bi mogli napraviti na sljedeći način:

```
grep ^# datoteka.txt

# Ovo je komentar
```

ili smo isto mogli dobiti i na duži način:

```
cat datoteka.txt | grep ^#

# Ovo je komentar
```

U oba slučaja smo meta znak `^` i njemu pripadajući pojam koji tražimo, mogli staviti unutar dvostrukih ili jednostrukih navodnika:

```
grep "^#" datoteka.txt
```

Isto možemo postići i ovako:

```
grep '^#' datoteka.txt
```

Sada postaje jasnija upotreba meta znaka `^` za početak reda (engl. *Caret* ili *Power*).

## Upotreba meta znaka $

Sada ćemo vidjeti upotrebu meta znaka `$` u slučajevima kada se on ne koristi za ispisivanje vrijednosti varijable.
Naime meta znak **$** se u ovoj primjeni koristi za pronalaženje kraja reda teksta, koji sadrži željena pojedina slova/brojeve ili riječ.
U našoj tekstualnoj datoteci ćemo stoga probati pronaći sve redove teksta koji završavaju s riječi `komentar`:

```
grep komentar$ datoteka.txt

# Ovo je komentar
Ovo nije komentar
```

Ili isto možemo postići na duži način:

```
cat datoteka.txt | grep komentar$

# Ovo je komentar
Ovo nije komentar
```

U oba slučaja dobili smo redove teksta koji završavaju s riječi `komentar`. I naravno u oba slučaja smo meta znak i pripadajuću riječ mogli staviti unutar jednostrukih ili dvostrukih navodnika. Povežimo sada prethodna dva meta znaka: početak i kraj, odnosno: **^** i **$**.

S ovom novom pretragom ćemo zapravo dobiti potragu za praznim linijama (redovima teksta):

```
grep ^$ datoteka.txt
```

Odnosno isto možemo postići s kombinacijom naredbe `cat` za ispis pa naredbe `grep` za filtriranje:

```
cat datoteka.txt | grep ^$
```

I ovdje je moguće koristiti jednostruke ili dvostruke navodnike za navedene meta znakove **^$**.



Vezano za uporabu meta znakova, pogledajte sljedeća poglavlja:
**6.7. Nizanje i ulančavanje naredbi i statusni kodovi.**
**9.2. Procesi i signali koje im možemo poslati.**
**9.4. Komunikacija između procesa.**
**9.4.1. Unix Pipes.**
**9.4.2. Named pipe.**

**Izvori informacija: (817),(K-1),(K-12), `man bash`.**

## 5.12.1. Regularni izrazi i meta znakovi

***Slijedi napredno poglavlje (5.12.1.x)!***
Ponekad želimo isključiti djelovanje meta znakova odnosno prikazati ih upravo onako kako ih vidimo, za to služi meta znak `\`. Meta znak **** zovemo i "*Escape*" karakter odnosno kontrolna sekvenca i iza njega obavezno slijedi neki meta znak, bez razmaka, a čiju funkcionalnost želimo isključiti odnosno *Escape-ati*.

**Primjer**: Želimo li ispisati riječ `test*`, ali pošto je ***** meta znak, sustavu nekako moramo naznačiti da ga ipak samo ispiše.

```
echo test\*

test*
```

Vidljivo je da smo koristili poseban meta znak, za *escape-anje* **** iza kojeg je slijedio meta znak koji smo sâmo željeli ispisati (*****). Ispis smo napravili upotrebom naredbe `echo`.

**Izvori informacija za ovo poglavlje: (817), `man bash`.**

### 5.12.1.1. Upotreba meta znaka za komentar #

Meta znak # se koristi za označavanje komentara koji slijedi nakon ovog znaka. Meta znak za komentar se može koristiti direktno u ljusci (*shellu)* ili u *shell* skriptama. Komentari se najčešće koriste za opis (komentar) uz pripadajući programski kôd, neku funkciju ili nîz naredbi, opisujući drugima, za što one služe. Takvi komentari se ignoriraju od strane ljuske (ili programskog jezika) te nam služe isključivo za pojašnjavanje onoga što želimo komentirati drugima. Pogledajmo primjere

```
echo Upišite neki broj: # Ovdje se traži upis broja
Upišite neki broj:
```

Vidimo kako se tekst koji se nalazi iza meta znaka # ne ispisuje.

```
echo Upišite neki broj: '#' Ovdje se traži upis broja
Upišite neki broj: # Ovdje se traži upis broja
```

i

```
echo Upišite neki broj: "#" Ovdje se traži upis broja
Upišite neki broj: # Ovdje se traži upis broja
```

Zadnja dva primjera u kojima naš meta znak # koristimo ili unutar dvostrukih ili jednostrukih navodnika prikazuje meta znak # u oba slučaja, jednostavno jer se nalazi unutar navodnika koji ga uredno prikazuju kao da se radi o običnom tekstu (slovima ili brojevima). Isto će se dogoditi, ako između zadnjeg slova ili broja nema razmaka, a slijedi naš meta znak #.

```
echo Upišite neki broj# Ovdje se traži upis broja
Upišite neki broj# Ovdje se traži upis broja
```

Drugi primjeri upotrebe su unutar *shell* skripti, koje mogu izgledati ovako.

```
#!/bin/bash

#####################################################
# Sada slijedi dio u kojemu se traži upis teksta
#####################################################

echo "Upišite tekst"
```



Za više detalja o *shell* skriptama pogledajte poglavlje: **6. Shell skripte.** Osim standardne upotrebe meta znaka **#** za komentar, pogledajte i upotrebu na vrijednosti varijabli, u poglavlju: **6.2.4. Operacije nad varijablama**.

### 5.12.1.2. Upotreba navodnika ' " ` (jednostruki, dvostruki i jednostruki kosi)

Pod posebne meta znakove spadaju i navodnici. Navodnika postoje tri vrste:

- Jednostruki navodnici `' '` isključuju interpretaciju svih meta znakova osim **!** i ****
- Dvostruki navodnici `" "` isključuju interpretaciju svih meta znakova osim **$** (koji služi za supstituciju varijabli) te **!** kao i ` te ****
- Jednostruki lijevi kosi navodnici `` ` ` `` - sve što se nalazi unutar njih će biti izvršeno kao naredba ili nîz naredbi.

**Slijede primjeri**: Postavit ćemo varijablu `TEST` na vrijednost jedan (`1`)

```
TEST=1
```

Potom ćemo ispisati sâmu varijablu preko dvostrukih navodnika `" "` i to upotrebom naredbe `echo`.

```
echo "$TEST"
1
```

Zatim ćemo ju ispisati upotrebom jednostrukih navodnika `' '` također s upotrebom naredbe `echo`.

```
echo '$TEST'
$TEST
```

Ovdje je vidljivo kako smo s upotrebom jednostrukih navodnika `' '` ispisali meta znak **$** i pripadajuću varijablu `TEST`, ali ne i njenu vrijednost. Sada ćemo ispisati tekst komentara (`komentar`) pomoću dvostrukih navodnika `" "` te pozvati drugu naredbu, pomoću jednostrukih kosih navodnika `` ` ` `` na sljedeći način:

```
echo "komentar `ls -al datoteka.txt`"

komentar -rw-r--r-- 1 root root 0 Nov 21 09:49 datoteka.txt
```

Obratite pažnju na to kako će sve što je upisano između jednostrukih kosih navodnika `` ` ` `` biti pokrenuto kao naredba.
U ovom slučaju, nakon što je ispisan tekst `komentar` pokrenuta je naredba za ispis sadržaja direktorija za navedenu datoteku odnosno naredba: `ls -al datoteka.txt`

A sada ćemo isti primjer pokrenuti pomoću jednostrukih navodnika `' '`

```
echo 'komentar `ls -al datoteka.txt`'

komentar `ls -al datoteka.txt`
```

U ovom slučaju dobili smo sve ispisano kao običan tekst, jer smo koristili jednostruke navodnike tako da je sve što se našlo unutar njih, sâmo ispisano.

I sada ponavljamo sve isto, ali pomoću jednostrukih navodnika `' '`, a naredbu unutar njih, pišemo pomoću dvostrukih navodnika `" "` i to ovako:

```
echo 'komentar "ls -al datoteka.txt"'

komentar "ls -al datoteka.txt"
```

Vidimo kako se ovdje također nije izvršila naredba unutar dvostrukih navodnika: `ls -al datoteka.txt` jer oni ne služe tome, niti bi išta bilo pokrenuto da smo koristili jednostruke *kose* navodnike jer se sve nalazi unutar jednostrukih navodnika.



*Upotreba jednostrukih kosih navodnika je korisna ako želimo vrijednosti neke varijable dodijeliti rezultat pokretanja neke naredbe ili nîza naredbi.*

Postavimo našu varijablu `integritet` na sljedeći način:

```
integritet=`md5sum vaznadatoteka.pdf`
```

Čitanjem vrijednosti ove varijable zapravo pozivamo naredbu `md5sum vaznadatoteka.pdf` koja će nam napraviti *MD5* provjeru integriteta datoteke imena: `vaznadatoteka.pdf`. Sada provjerimo vrijednost naše varijable:

```
echo $integritet
```

Dobivamo:

```
fa8e75a54297fb9a107c263cc3cfbe68 vaznadatoteka.pdf
```

Dobili smo ***MD5*** provjerni zbroj (engl. *checksum*) naše datoteke, koji je: `fa8e75a54297fb9a107c263cc3cfbe68`.
Ovakva upotreba jednostrukih kosih navodnika je vrlo česta u *shell* skriptama.

**Izvori informacija: (817), `man bash`.**

---

### 5.12.1.3. Upotreba uglatih [ ] zagrada

Uglate zagrade **[ ]** kod regularnih izraza koriste se za označavanje niza numeričkih, alfabetskih ili posebnih znakova odnosno karaktera. Dakle govorimo o pronalasku ili upotrebi svakog (ili bilo kojeg) pojedinačnog, željenog znaka.
Najčešća upotreba uglatih zagrada je kod sljedeće primjene:

- `[ABC]` – označava sâmo pojedina velika slova: `A`, `B` ili `C`
- `[A-Z]` – označava nîz pojedinačnih velika slova od `A` do `Z` (A B C D E F G H … Z)
- `[[:`**`kategorija`**`:]]` – označava neku od **kategorija**:
    - `alnum` – označava sve alfanumeričke znakove (slova i brojeve) - ovo je ekvivalent `[0-9A-Za-z]`
    - `alpha` – označava sva velika i mala slova (alfabet) - ovo je ekvivalent `[A-Za-z]`
    - `ascii` – označava sve znakove iz ASCII tablice (slova, brojevi, posebni znakovi, …).
    - `cntrl` – označava sve kontrolne znakove iz ASCII tablice (oktalno 000 - 037 i 177).
    - `digit` – označava sve brojeve (0,1,2,3,4,5,6,7,8 i 9).
    - `graph` – označava sve grafičke znakove: `alnum` i `punct`, a to su znakovi:
      **! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~**
    - `lower` – označava sva mala slova - ovo je ekvivalent `[a-z]`
    - `print` – označava sve "*printabilne*" odnosno ispisive znakove: `alnum`, `punct` i razmak (engl. *Space*)
    - `punct` – označava sve posebne znakove: ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~
    - `space` – označava razmak (engl. *Space*).
    - `upper` – označava sva velika slova - ovo je ekvivalent `[A-Z]`
    - `xdigit` – označava sve heksadecimalne znakove: 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
- `[^…]` – označava negaciju onoga što slijedi (…)



Ovisno o lokalnim odnosno regionalnim postavkama jezika i postavkama za baratanje s karakterima odnosno alfabetom i (en)kodiranjem, mogući su problemi kod pretvorbe i rada s velikim i malim slovima.



Za više detalja o (en)kodiranju pogledajte poglavlje:
**10.3.2 Kodiranje, dekodiranje i kodne stranice.**

Provjerena kombinacija regionalnih postavki je upotreba ***POSIX*** ili ***C*** kao standarda kao definicije za nacionalne (lokalne) postavke te odabir "***Locale***" kategorije kod baratanja, poglavito s velikim i malim slovima i zapravo većinom tipova navedenih gore, pod [[**:kategorija:**]].

Samo u slučaju kada imate problem kod izvršavanja primjera dolje, stavite slijedeća dva reda u datoteku `/etc/profile`, snimite ju te se odlogirajte i ponovno logirajte na sustav. Ovdje smo koristili naredbu `export` za eksportiranje varijabli:

```
export LANG=POSIX
export LC_CTYPE=en_US.UTF-8
```

Zamislimo sljedeći primjer: u trenutnom direktoriju nam se nalazi sljedeća struktura datoteka (vidljiva dolje u ispisu).
To su radne datoteke, prema kategorijama:

- sve koje u nazivu imaju "`.1.mj-`" su za radne grupe u prvom mjesecu
- sve koje u nazivu imaju "`.2.mj-`" su za radne grupe u drugom mjesecu itd.

Istovremeno imamo i potkategorije nakon oznake mjeseca, a to su velika slova: `A`, `B` i `C` za završene projekte te mala slova: `a`,`b` i `c` za nedovršene ili projekte u izradi, koji kada budu dovršeni, mijenjaju se u velika slova.

Pogledajmo te datoteke (skraćeni ispis):

```
ls -al
-rw-r--r-- 1 root root   53 Apr 26 12:28 datoteka.1.mj-A.txt
-rw-r--r-- 1 root root   53 Apr 26 12:28 datoteka.1.mj-B.txt
-rw-r--r-- 1 root root   53 Apr 26 12:28 datoteka.1.mj-C.txt
-rw-r--r-- 1 root root   53 Apr 26 12:28 datoteka.2.mj-A.txt
-rw-r--r-- 1 root root   53 Apr 26 12:28 datoteka.2.mj-B.txt
-rw-r--r-- 1 root root   53 Apr 26 12:28 datoteka.2.mj-C.txt
-rw-r--r-- 1 root root   53 Apr 26 12:29 datoteka.3.mj-a.txt
-rw-r--r-- 1 root root   53 Apr 26 12:29 datoteka.3.mj-b.txt
-rw-r--r-- 1 root root   53 Apr 26 12:29 datoteka.3.mj-c.txt
```

**Pogledajmo i nekoliko primjera upotrebe uglatih zagrada [ ]**
Pronađimo sve datoteke iz svih mjeseci koji imaju velika slova (od `A` do `Z`) jer ne znamo koliko projekata imamo.
Dakle tražimo sve dovršene projekte:

```
ls -al datoteka.*.mj-[A-Z].txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-A.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-B.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-C.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-A.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-B.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-C.txt
```

Dobili smo listu svih datoteka koje se zovu `datoteka.` `*` što znači bilo koji broj (ili slovo) `.mj-` i potom bilo koje slovo od **A** do **Z** "[A-Z]" te završetak naziva imena sa `.txt`. Ovdje smo imali upotrebu uglatih zagrada koje mijenjaju bilo koje slovo u nîzu koji smo naveli (sva velika slova od **A** do **Z**). Pošto smo filtrirali sve kako treba, dobili smo sve mjesece (`.1.mj-`) i sve velika slova (**A** do **Z**, a pošto ih je bilo od **A** do **C**, oni su i prikazani).

U drugom primjeru ćemo pokušati napraviti suprotno. Sada nas zanimaju projekti koji su u izradi ili nedovršeni, prema tome sada na poziciji gdje su se tražila sva velika slova od **A** do **Z**, tražimo sva mala slova od **a** do **z**.

```
ls -al datoteka.*.mj-[a-z].txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-a.txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-b.txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-c.txt
```

Ovdje bi sve trebalo biti jasno jer smo umjesto velikih slova na istoj poziciji sada tražili mala slova `[a-z]`.

Isto smo mogli postići i s kategorijom `[[:lower:]]` koja također označava sva mala slova:

```
ls -al datoteka.*.mj-[[:lower:]].txt

-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-a.txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-b.txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-c.txt
```

U slijedećem primjeru nas zanima koliko projekata uopće imamo od prvog (`.1.mj`) do trećeg mjeseca (`.3.mj`), bez obzira u kojoj su fazi (završeni ili u izradi):

```
ls -al datoteka.[1-3].mj-*.txt

-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-A.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-B.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-C.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-A.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-B.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-C.txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-a.txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-b.txt
-rw-r--r-- 1 root root 53 Apr 26 12:29 datoteka.3.mj-c.txt
```

Sada smo na poziciji oznake broja mjeseca koristili opseg brojeva od 1 do 3: `[1-3]` dakle odgovaraju svi koji imaju na toj poziciji brojeve 1,2 i 3.

U gornjim primjerima smo spominjali nizove slova `[A-Z]` (A,B,C,D, … Z), `[a-z]` (a, b, c, d, … z) ili brojeva `[0-9]` (0,1,2,3,4, … 9). Na sličan način mogu se koristiti i točno specificirana slova ili brojevi koje želimo, u slijedu.
Njih je potrebno definirati u slijedu unutar uglatih zagrada, primjerice : `[ABKL]` ili `[1246]`.

U ovom primjeru tražimo sâmo sve datoteke za projekte isključivo iz prvog (1) i drugog mjeseca (2):

```
ls -al datoteka.[12].mj-*.txt

-rw-r--r-- 1 root root 79 Apr 26 13:47 datoteka.1.mj-A.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-B.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.1.mj-C.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-A.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-B.txt
-rw-r--r-- 1 root root 53 Apr 26 12:28 datoteka.2.mj-C.txt
```

**Negacija**
Iz predzadnjeg primjera, možemo isključiti sve od prvog (1) i drugog mjeseca (2) je nas zanimaju svi ostali mjeseci (`1-9`), za sada. Za ovu potrebu se koristi znak negacije `^` koji inače može imati i drugačiju namjenu.

```
ls -al datoteka.[^1-2].mj-*.txt

-rw-r--r-- 1 root root 78 Apr 26 13:50 datoteka.3.mj-a.txt
-rw-r--r-- 1 root root 75 Apr 26 13:53 datoteka.3.mj-b.txt
-rw-r--r-- 1 root root 78 Apr 26 13:53 datoteka.3.mj-c.txt
```

Sada ćemo pogledati sadržaj jedne datoteke s projektima, koji su podijeljeni u četiri (4) faze.
Ovako izgleda projekt, odnosno sadržaj datoteke u kojoj su popisani projekti koji su dovršeni:

```
# Projekt 1 - A   - ovo je komentar
Faza1-OK
Faza2-OK
Faza3-OK
Faza4-OK
```

Faze koje nisu dovršene imaju umjesto `OK` oznake, oznaku `NotOK`. Nas zanima projekt iz trećeg mjeseca; datoteka koja ima u imenu: `.3.mj` i to projekt `a`. Konkretno želimo iz ove datoteke izvući podatak, koliko "*faza*" projekta je dovršeno:

```
cat datoteka.3.mj-a.txt  | grep "Faza[1-4]-OK"

Faza1-OK
Faza2-OK
```

Ovdje smo za pronalaženje riječi/pojma unutar teksta koristili naredbu `grep`. Njoj smo na poziciji nakon riječi odnosno pojma koji tražimo: "`Faza`" koristili naš regularni izraz za pretraživanje brojeva od jedan (1) do četiri (4). S time smo dobili pretraživanje svih riječi: `Faza1-OK`, `Faza2-OK`, `Faza3-OK` i `Faza4-OK`.

Pošto su samo `Faza1` i `Faza2` izgledale točno kako smo i tražili (`Faza1-OK` i `Faza2-OK`), one su i prikazane.
S time smo dobili traženi rezultat, da su samo `Faza1` i `Faza2` dovršene.

**Izvori informacija: (817), `man bash`.**

### 5.12.1.4. Upotreba vitičastih { } zagrada

Vitičaste `{ }` zagrade se često koriste:

- Za kreiranje nîza znakova.
- Kod rada s varijablama polja (pogledajte poglavlje: **6.2.3 Varijable polja (*Array variable*)**).
- Za proširenje vrijednosti varijabli (pogledajte poglavlje: **6.2.1 Standardne odnosno obične varijable**).
- Za proširenje parametara pozicije većih od devet (pogledajte poglavlje: **6.3 Parametri pozicije**).
- Za razne operacije nad vrijednostima varijabli (pogledajte poglavlje: **6.2.4 Operacije nad varijablama**).

Pogledajmo nekoliko primjera upotrebe vitičastih `{ }` zagrada, direktno u ljusci (*shellu).*
Kreirat ćemo listu (nîz) riječi, u ovom slučaju niz riječi: `prvo` `drugo` `treće` `četvrto` upotrebom naredbe `echo`.

```
echo {prvo,drugo,treće,četvrto}

prvo drugo treće četvrto
```

Sada smo dobili samo ispis svih elemenata, odvojenih zarezom, koji se nalazi unutar vitičastih zagrada, unutar kojih se nalaze element polja (riječi). Zatim ćemo kreirati novu listu (nîz) znakova, u ovom slučaju slova od **A** do **E**

```
echo {A..E}

A B C D E
```

Isto možemo napraviti i s brojevima. Kreirati ćemo nîz brojeva od 10 do 15:

```
echo {10..15}

10 11 12 13 14 15
```

Potom kreirajmo decimalne brojeve od 1.1 do 1.5 na sljedeći način:

```
echo 1.{1..5}

1.1 1.2 1.3 1.4 1.5
```

Naravno, moguće su i kombinacije više nîzova. Sada ćemo kreirati nîz slova i brojeva (A do E) i (1 do 5) i to kombinirano:

```
echo {A..E}{1..5}

A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5 D1 D2 D3 D4 D5 E1 E2 E3 E4 E5
```

Također je moguće i kombinirati nîzove i raditi ugnježđivanje, poput ovoga u primjeru:

```
echo {a,b{1..4},c{1..3},d}

a b1 b2 b3 b4 c1 c2 c3 d
```

A sada kreirajmo alfabet, prvo sva velika slova u nîzu, a potom i mala slova:

```
echo {{A..Z},{a..z}}

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q
r s t u v w x y z
```

U ovom primjeru probajmo kreirati više poddirektorija unutar osnovnog direktorija, odjednom (`-p`):

```
mkdir -p /home/korisnik/test/{prvi,drugi,treci,cetvrti}
```

Dakle unutar direktorija `/home/korisnik/test/` smo kreirali poddirektorije: `prvi`, `drugi`, `treci` i `cetvrti`.
Osim gore navedenih primjera upotrebe vitičastih zagrada one se koriste i u ljusci (*shellu*) i u radu s običnim varijablama.



Pogledajte i poglavlje:
**6.2.1 Standardne odnosno obične varijable**.

**Izvori informacija:** `man bash`

#### 5.12.1.4.1. Posebne varijable *bash* ljuske

***Slijedi napredna cjelina!***
Osim navedenih primjera, postoji i nekoliko posebnih sistemski definiranih varijabli, koje možemo koristiti unutar vitičastih zagrada, kako bismo dobili njihovu vrijednost.

Ovdje ćemo spomenuti nekoliko sistemskih varijabli. Jedna od njih je varijabla čije pozivanje generira slučajne brojeve, ne previše slučajne, ali zadovoljavajuće za osnovnu upotrebu.

To je varijabla `RANDOM`. Ispišemo li njenu vrijednost upotrebom naredbe `echo` dobit ćemo neki slučajni broj:

```
echo ${RANDOM}

17891
```

Sljedeća varijabla `SECONDS` broji vrijeme u sekundama od kada ste se logirali, pa pogledajmo i nju:

```
echo ${SECONDS}

745
```

Još je zanimljiva i varijabla `LINENO` koja prati i može ispisati u kojoj liniji (redu) se koja naredba u skripti izvršila.

Pogledajmo primjer *shell* skripte, koju smo nazvali `lineno.sh`:

```
#!/bin/bash
echo DRUGI RED
echo ${LINENO}
echo TRECI RED
```

Ako pokrenemo našu skriptu `lineno.sh`, dobit ćemo:

```
DRUGI RED
3
TRECI RED
```

Dakle dobili smo ispis teksta `DRUGI RED` pa broj `3` u drugom redu ispisa koji označava ispis varijable `LINENO` koja se u skripti nalazi u trećem redu, a zatim slijedi ispis teksta `TRECI RED`.

**Izvori informacija: (817), `man bash`.**

---

## 5.12.2. Drugi napredni primjeri upotrebe *bash* ljuske

***Slijedi napredna cjelina!***
Uđimo u novi direktorij i pogledajmo koje sve datoteke imamo u njemu, upotrebom naredbe `ls`:

```
ls
a.jpg  a.pdf  b.jpg  b.pdf  c.jpg  c.pdf
```

Ispišimo sve datoteke s ekstenzijom `.jpg` korištenjem posebnog regularnog izraza odnosno zamjenskog znaka `*`

```
ls *.jpg
a.jpg  b.jpg  c.jpg
```

Sada ispišimo sve datoteke s ekstenzijom `.jpg` i s ekstenzijom `.pdf`

```
ls *.jpg *.pdf
a.jpg  a.pdf  b.jpg  b.pdf  c.jpg  c.pdf
```

Međutim ***Linux bash*** ljuska omogućava nam i korištenje proširenih regularnih izraza odnosno zamjenskih znakova s kojim možemo pojednostaviti određene zadatke. Zamjenski znakovi se često nazivaju ***glob*** uzorcima (engl. *Glob*= komad, blok). ***Bash*** ljuska nam nudi i upotrebu proširenih *glob* uzoraka, koje moramo prvo aktivirati.

Naime, aktivaciju upotrebe proširenih regularnih izraza odnosno ***glob*** uzoraka, možemo napraviti sa naredbom ugrađenom u sâmu ***bash*** ljusku. Radi se o naredbi `shopt`.

Prvo možemo vidjeti koje su sve opcije i parametri uključeni za našu trenutnu *bash* ljusku; (ispis ove naredbe nećemo prikazati u cijelosti, već samo njen vrlo mali dio). Ove opcije ćemo dobiti upotrebom naredbe `shopt` na sljedeći način:

```
shopt -p

shopt -u checkhash
shopt -s cmdhist
shopt -u direxpand
shopt -u dotglob
shopt -u execfail
shopt -u extdebug
shopt -u extglob
shopt -s extquote
shopt -s login_shell
shopt -u failglob
shopt -u globstar
```

Dakle vidimo kako je standardno uključeno dosta naprednih opcija.

Jedna od opcija koja primjerice utječe na način rada ljuske ***(remote/interactive/non-interactive/non-login***) je `login_shell`. Naime moguće je da netko u radu promjeni način rada ***bash*** ljuske. Primjerice iz normalnog (***interactive login***) u **non-login**, što osim što utječe na način rada ljuske, utječe i na to koje se inicijalizacijske datoteke s postavkama sustava učitavaju, primjerice:

- Za ***interactive login*** način rada je redoslijed učitavanja inicijalizacijskih datoteka: `/etc/profile`, `~/.bash_profile`, `~/.bash_login` i na kraju se učitava `~/.profile` datoteka.
- Za ***non-login*** način rada je redoslijed učitavanja drugačiji jer učitava samo `~/.bashrc` datoteka.

Provjerimo imamo li normalni (***interactive***) ili primjerice ***non-login*** način rada ljuske, što možemo postići sa:

```
shopt login_shell
login_shell     on
```

U ovom slučaju vidimo da ljuska radi u normalnom (***interactive login***) načinu rada jer imamo (`login_shell     on`).



Vezano za načine rada **bash** ljuske (***remote/interactive/non-interactive/non-login***) pogledajte poglavlje:
**3.1.5. Bash ljuska detaljnije.**

Sada ćemo uključiti i našu opciju koja se naziva `extglob`, sa sljedećom naredbom:

```
shopt -s extglob
```

Upoznajmo se s nekima od proširenih regularnih izraza:

- `?(uzorci)` – označava niti jedno ili samo jedno pojavljivanje uzoraka.
- `*(uzorci)` – označava niti jedno ili više pojavljivanja uzoraka.
- `+(uzorci)` – označava jedno ili više pojavljivanja uzorka.
- `@(uzorci)` – označava jedno pojavljivanje uzoraka.
- `!(uzorci)` – označava sve osim navedenih uzoraka.

Sada možemo nastaviti s naprednim primjerima; ponovimo zadnji primjer potrage za datotekama s ekstenzijama `.jpg` i `.pdf`

```
ls *(*.jpg|*.pdf)
a.jpg  a.pdf  b.jpg  b.pdf  c.jpg  c.pdf
```

U ovom primjeru, probajmo pronaći sve datoteke čija ekstenzija **nije** `.jpg`

```
ls !(*.jpg)
a.pdf  b.pdf  c.pdf
```

Moguće su i kombinacije poput ove u kojoj želimo ispisati sve datoteke koje u nazivu imaju slovo `a` ili `b` (ili više ponavljanja istih slova) te su ekstenzije: `.jpg` i `.pdf`

```
ls +(a|b)+(.jpg|.pdf)
a.jpg  a.pdf  b.jpg  b.pdf
```



**Navedene mogućnosti ovise o sâmoj ljusci odnosno njenoj inačici!**

**Izvori informacija: (626),(627),(817),(1201),(1202),(1569),** `man shopt`,`man bash`.

# 6. Shell skripte

***Slijedi napredno poglavlje (6.x)!***
Podsjetimo se: naredbena ljuska ili *Shell* je sučelje između korisnika i operativnog sustava, iz nje pokrećemo razne programe i upravljamo cijelim sustavom.
Ovakvih ljuski postoji veći broj, od kojih svaka ima neke specifičnosti, a mi ćemo se bazirati na `bash` (*Bourne-Again SHell*) ljusci. Najveća snaga svake ljuske je mogućnost izrade *shell* skripti. ***Bash*** nam pri tome daje prilično široke mogućnosti.

Važno je znati, kako svaka *shell* skripta mora zadovoljiti dva uvjeta:

1. Prvi redak svake skripte mora sadržavati tzv. *Sha-bang* ili *shebang*, koji daje instrukcije operativnom sustavu, o kakvoj se skripti radi, to jest što odnosno koji program sustav prvo treba pokrenuti kada pokreće skriptu.

U našem slučaju to je *bash* shell skripta, dakle za njeno pokretanje sustav će za nas u pozadini prvo pokrenuti još jednu *bash* ljusku (*shell)* te potom našu skriptu u njemu. To izgleda tako da prvi red skripte (datoteke) mora počinjati sa znakovima: **`#!`** nakon kojih slijedi točna putanja do izvršne datoteke odnosno programa *bash ljuske*, poput:

```
#!/bin/bash
```

2. Skripta potom mora biti izvršna odnosno imati postavljene ***eXecute*** ovlasti (`x`). Dakle potrebno je promijeniti joj ovlasti kako bi skripta (skriptna datoteka) postala izvršna.
   To možemo napraviti s naredbom `chmod` na sljedeći način:

```
chmod +x ime.skripte
```

**Izvori informacija: (95),(96),(133),(817),(K-1),(K-12), `man bash`,`man chmod`.**

---

# 6.1. Kako rade Shell skripte

Svaka *Shell* skripta nakon prvog retka teksta koji sadrži `#!/bin/bash`, u kojem se definira ljuska koja se poziva, ima nîz naredbi koje skripta mora izvršiti. Naredbe se obično nîžu; svaka u novom redu, prema redoslijedu kojim se moraju izvršavati.
Ovo navođenje naredbi, za koje želimo da se izvršavaju, izvršava se vrlo slično kao kada bi iste te naredbe pokrenuli i sâmi iz svoje ljuske (*shell-a)* i to jednu nakon druge.

Zamislimo *shell* skriptu, koja sadrži sljedeće retke:

```
#!/bin/bash

cd /root/
ls -al
```

Pokretanjem ove *shell* skripte pokreće se nova (pod) ljuska `/bin/bash` u kojoj se pokreće prva naredba koju smo pozvali u *shell* skripti. U konkretnom slučaju je to naredba: `cd /root/` što znači kako ulazimo u direktorij: `/root/`.
Potom se pokreće sljedeća naredba u novom retku. U našem slučaju je to naredba: `ls -al` koja nam ispisuje sadržaj našeg trenutnog direktorija u koji smo upravo ušli.

Osim čistog navođenja ili nîzanja naredbi, u svakom novom retku, *Bash ljuska* nam omogućava i korištenje varijabli te kreiranje željenih uvjeta i petlji.
Dodatno možemo praviti i svoje funkcije koje možemo kasnije pozivati. Sve u svemu ***BASH*** *ljuska* nam omogućava pisanje *shell* skripti koje podsjećaju na neki jednostavniji programski jezik.

## Što se inicijalno događa tijekom pokretanja svake *shell* skripte?

Sada kada smo u grubo shvatili kako rade *shell* skripte, upoznat ćemo se sa procesima koji se događaju kod inicijalizacije svake *shell* skripte koju pokrećemo; u trenutku svakog njenog pokretanja. U inicijalnom trenutku nova ljuska (***BASH*** u našem slučaju) pretražuje sistemske (*environment*) varijable i njihove vrijednosti, koje zatim ljuska (*shell)* postavlja kao svoje.

Sve dodatne varijable koje nisu eksportirane s metodom: `export IME_VARIJABLE=VRIJEDNOST`, neće doći do pôd ljuske odnosno u konačnici do *ljuske* u kojoj se izvršava naša skripta.

Naime svaka skripta u pozadini pokreće svoju pôd ljusku, koja je prema hijerarhiji ljuska ispod one iz koje se pokreće sâma skripta, pa tako imamo po jednu pôd *ljusku* za svaku pokrenutu skriptu. Kada je pôd ljuska odnosno druga ljuska koju pokreće naša skripta u procesu inicijalizacije dohvatila i postavila sve sistemske (*Environment*) varijable, kao i varijable koje su "eksportirane" odnosno izvezene, ona kreće s postavljanjem posebnih varijabli kao što je varijabla `$#` koja pokazuje s koliko argumenata je sama skripta pokrenuta.

Nadalje postavljaju se i varijable pozicije i to: `$0`, `$1`, `$2`, ...  te druge posebne varijable.
Još jedna od posebnih varijabli je varijabla `SHLVL` koja numerira razinu vršne ili pôd *ljuske*.
Tako je uvijek prva (login) ljuska, razine **1**, a prva njena pôd ljuska razine **2**, a njena pôd ljuska razine **3** i tako dalje.



Vezano za varijable pozicij, poglddajte poglavlje:
**6.3. Parametri pozicije.**

Pogledajmo primjer kada iz prve, odnosno login ljuske (*shella*) pokrećemo drugu pôd ljusku (`bash`), a iz te pôd ljuske pokrećemo pôd, pôd ljusku odnosno novi `bash`. Dakle u svakoj novoj pôd *ljusci* se ova varijabla povećava:

```
[root@server: ~]# echo $SHLVL

1
[root@server: ~]# bash
[root@server: ~]# echo $SHLVL

2
[root@server: ~]# bash
[root@server: ~]# echo $SHLVL

3
```

Dalje u radu naša skripta koja se sada izvršava u novoj ljusci, u trenutku kada se pokreće neka naredba, prvo provjerava nalazi li se ta naredba u standardnoj sistemskoj varijabli `PATH`, koju je ova ljuska pokupila od vršne ljuske (*shella*).
Varijabla `PATH` sadrži putanje do izvršnih datoteka odnosno naredbi. Ako je naša naredba u nekom drugom direktoriju osim onoga definiranog u ovoj varijabli, tada ju *ljuska* neće moći pronaći niti pokrenuti te će za nju izbaciti grešku i nastaviti dalje.

Stoga je važno provjeriti vrijednost ove varijable ili unutar skripte definirati novu `PATH` varijablu koja će sadržavati nama potrebne direktorije (mape) unutar kojih su naredbe koje ćemo u skripti pozivati.
Naša *shell* skripta bi sada izgledala ovako:

```
#!/bin/bash

PATH=$PATH:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin

cd /root/
ls -al
```

Ovdje novoj varijabli `PATH` dodajemo postojeće vrijednosti stare varijable `PATH` (sa ispisom njene vrijednosti pomoću `$PATH`) te dodajemo i nove direktorije, koji se odvajaju znakom `:`. Sada sve naredbe koje ćemo koristiti dalje u skripti, a nalaze se u nekom od definiranih direktorija unutar ove varijable, naša pôd ljuska može uredno i pokrenuti jer zna kako doći do njih. Točnije sada zna na kojoj putanji bi se mogle nalaziti naredbe koje ćemo pokretati/pozivati.

Dodatna važna funkcionalnost pôd *shella* odnosno pôd ljuske je prosljeđivanje Tzv. “***Exit statusa***” znanog i kao “***Return code***” ili “***Exit code***” prema vršnoj ljusci. Ovaj statusni kôd radi na principu da, ako je sve u redu, šalje se vrijednost `0`, a ako nešto nije u redu s izvršavanjem, šalje se neka druga vrijednost odnosno statusni kôd.

Ova vrijednost se šalje vršnoj ljusci, kao parametar `$?`.

Osim slanja statusnih kôdova vršnoj ljusci (engl. *Parent Shell*), ove statusne kôdove možemo i čitati unutar naše ljuske ili skripte.



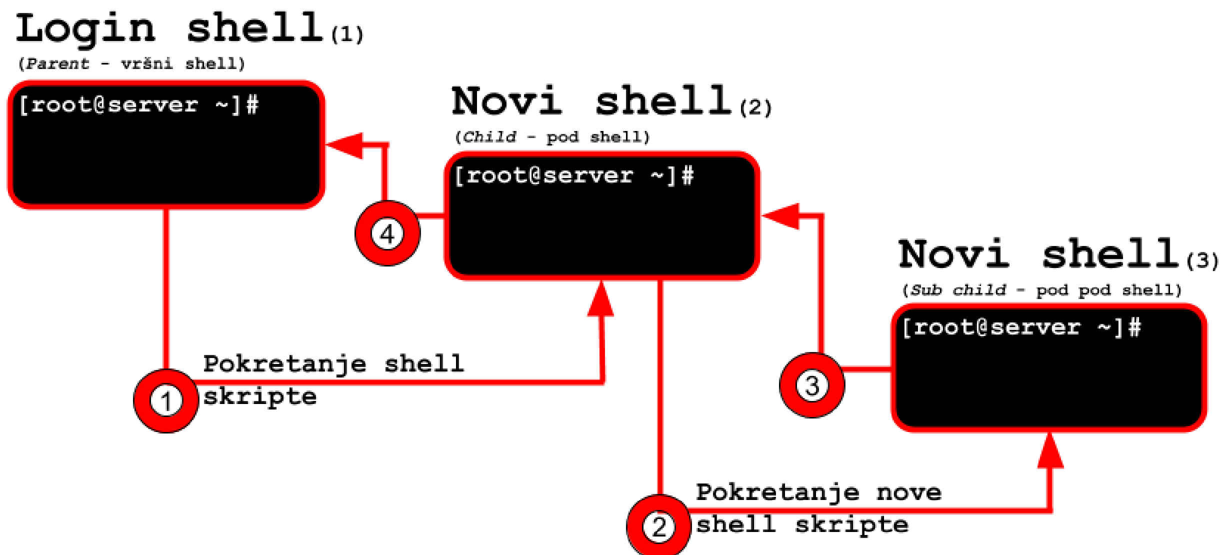
Pogledajte primjere i opis, te statusne kodove u poglavlju:
**6.7 Nizanje i ulančavanje naredbi.**

Nakon što je naša pôd ljuska locirala prvu naredbu pretraživanjem svih putanja direktorija koji su definirani u `PATH` varijabli ona provjerava je li naredba izvršna odnosno ima li postavljena “*execute*” (`x`) prava. U ovom koraku se provjerava koja je vrsta datoteke; i dalje govorimo o naredbi koju treba izvršiti unutar skripte (jednu po jednu). Ako se radi o binarnoj datoteci ona se izvršava.
Ako se ipak radi o drugoj skriptnoj datoteci ili nekoj drugoj tekstualnoj datoteci koja je izvršna, ona se također pokreće na način na koji je i naša vršna roditeljska (engl. *Parent*) ljuska pokrenula našu skriptu. Dakle pokreće se pôd ljuska u kojoj se dalje izvršava ta nova skripta.

Na slici 16. je vidljiv primjer kada iz svoje radne ljuske (*login shella*) pokrećemo skriptu: (korak **1**), koja otvara svoju ljusku i pokreće se. Naša skripta potom pokreće drugu skriptu koja isto mora pokrenuti svoju ljusku (korak **2**) i pokrenuti se u njemu.

Kada je zadnja skripta završila, vraća rezultat prijašnjoj ljusci (korak **3**) i zatvara svoju ljusku. Na kraju kada naša skripta završi s radom, također šalje rezultat vršnoj ljusci i zatvara svoju ljusku (korak **4**):

**Slika 16. Ljuska (*shell*) i njene pod ljuske.**




**Sumirajmo** :

- Svaka *shell* skripta može pokrenuti jednu ili više pôd *ljuski.*
  - Svaka pôd ljuska je jedan novi proces sa svojim identifikacijskim ***PID*** brojem.
- Pôd *ljuske* ne nasljeđuju standardne varijable; osim ako nisu `export`-irane (izvezene).
  - Sâmim tim je i djelovanje standardnih varijabli ograničeno na konkretnu ljusku, odnosno unutar nje.
- Promjena radnog direktorija kao i korisnika (*User ID*) pod ljuske ne utječe na radni direktorij (mapu) i korisnika vršne (*engl. Parent*) ljuske.
- Svaka ljuska, i vršna i svaka njena pôd ljuska, mogu pokretati više programa (procesa) u paraleli.



Pogledajte poglavlje:
**9.5 Linux poslovi (*jobs*)** te pokretanje procesa u pozadini, pomoću meta znaka **&** na kraju reda.



Upotrebom okruglih **(** **)** zagrada se također mogu pokretati posebne vrste pôd ljuski (pod *shell*ova).

**Izvori informacija: (95),(96),(133),(817),(1569),(K-1),(K-12), `man bash`.**

## 6.2. Varijable

***Slijedi napredno poglavlje (6.2.x)!***
Naredbeni redak (naredbeno linijska ljuska ili sâmo ljuska) odnosno *shell*, omogućava nam upotrebu varijabli poput varijabli u nekom programskom jeziku. Pri tome razlikujemo nekoliko vrsta varijabli:

- Environment varijable odnosno varijable za okruženje sustava znane i kao varijable (radne) okoline.
- Obične (*shell*) odnosno standardne varijable ljuske.

### 6.2.1. Standardne odnosno obične varijable

Standardne ili obične varijable vrijede sâmo unutar ljuske u kojoj su pokrenute, Ako unutar postojeće ljuske pokrenemo novu ljusku (tzv. *subshell*), varijable iz vršne ljuske (roditeljske) se neće naslijediti. U tu svrhu možemo koristiti varijable okoline odnosno takozvane "*Environment* varijable" koje se prenose u sve pôd *ljuske*, na način da se globalno izvoze (*exportiraju*).

Pogledajmo primjere: prvo kreirajmo sistemsku varijablu `TEST` koja će imati vrijednost `korisnik1`

```
TEST=korisnik1
```

Ispišimo vrijednost varijable `TEST` u trenutnoj ljusci (vrijednost varijable dobivamo sa znakom **$**), upotrebom naredbe `echo`:

```
echo $TEST

korisnik1
```

Vidimo kako smo u trenutnoj ljusci uredno dobili njenu vrijednost koja je postavljena na: `korisnik1`.



Pogledajte i poglavlje:
**5.12.1.2 Upotreba navodnika ' " ` (*jednostruki, dvostruki i jednostruki kosi*)** te se prisjetite upotrebe navedenih navodnika u kombinaciji s varijablama.

**Delimitiranje vrijednosti varijable**
U slučaju kada imamo potrebu nakon ispisivanje vrijednosti varijable s njom spojiti i neku riječ, ovdje ćemo imati problem. Ako primjerice želimo riječ `DOMENA` prilijepiti (nadodati) vrijednosti varijable `TEST`, nećemo ništa dobiti. Pokušajmo:

```
echo $TESTDOMENA
```

Stoga je potreban delimiter varijabli odnosno njenoj vrijednosti. Za ovu potrebu se koriste vitičaste zagrade **{ }**.
Sada pogledajmo novi primjer ovakve upotrebe. Dakle vrijednosti prve varijable dodajemo novu riječ `DOMENA`.

```
echo ${TEST}DOMENA

korisnik1DOMENA
```

Ovo izgleda pomalo nezgrapno, ali pisanjem *shell* skripti, često se može pojaviti potreba kada nakon potrebe za ispisivanjem vrijednosti varijable imamo i potrebu nadodati joj neku ključnu riječ ili slično, a što je moguće, isključivo na ovaj način.
Ako želimo ispisati vrijednost varijable koju koristimo u nekoj *shell* skripti tada možemo koristiti i dvostruke navodnike `" "`.

U ovom primjeru ćemo dobit isti ispis ali u skripti ćemo biti sigurni da smo proslijedili upravo ono što smo i željeli:

```
echo "$TEST"

korisnik1
```

**Primjer *shell* skripte s varijablama**
Kreirajmo osnovnu *shell* skriptu koja će postaviti vrijednost varijable te ju ispisati. Slijedeći kôd (naredbe) zapišimo u datoteku: `varijable.sh` na sljedeći način:

```
#!/bin/bash

TEST=korisnik1
echo "$TEST"
```

Sada moramo našoj *shell* skripti dodijeliti izvršna prava (engl. *execute*). To ćemo napraviti na sljedeći način:

```
chmod a+x varijable.sh
```

Sada možemo i pokrenuti našu *shell* skriptu iz trenutnog direktorija:

```
./varijable.sh

korisnik1
```

Zbog sigurnosnih razloga *shell* nikada ne pretražuje trenutni direktorij za izvršnim datotekama (pa tako niti izvršnim *shell* skriptama). Zbog toga je u slučaju kada želimo pokrenuti izvršnu datoteku (*shell* skriptu) u trenutnom direktoriju, potrebno naglasiti *ljusci* kako se shell *skripta* nalazi u trenutnom direktoriju `.` odnosno da ju pokrećemo iz trenutnog direktorija `./`
Za sve ostale izvršne datoteke *shell* gleda sistemsku varijablu `PATH` te ih može pokretati iz svih direktorija navedenih u toj varijabli. Vidjeli smo kako nam je naša *shell* skripta ispisala vrijednost varijable `TEST`.
Sada smo sve što smo napravili i ručno u *ljusci*, pokretanjem naredbi jedne za drugom, postigli i u jednoj *shell* skripti.

U drugom slučaju ako želimo ispisati varijablu sa znakom **$** koji bi nam inače dao njenu vrijednost, a u ovom slučaju to u nekoj *shell* skripti želimo proslijediti dalje, tada moramo koristiti jednostruke navodnike `' '`

```
echo '$TEST'

$TEST
```

Postavimo ovu varijablu kao varijablu okruženja (sustava) odnosno *environment* varijablu, na način da ju izvezemo.

Za to se koristi naredba `export`, kako bismo varijablu kasnije mogli koristiti u bilo kojoj novoj pod ljusci (*sub shellu)*:

```
export TEST
```

Sada i pokrenimo pôd ljusku (*subshell*):

```
bash
```

Potom ispišimo vrijednost ove varijable čiju vrijednost smo postavili u roditeljskoj ljusci:

```
echo $TEST

korisnik1
```

Vidimo kako se varijabla (i njena vrijednost) prenijela u pôd ljusku, kako smo i očekivali.
Sada obrišimo postavljenu varijablu iz početne ljuske. Za to se koristi naredba `unset` na sljedeći način:

```
unset TEST
```

Osim standardnih (običnih) i *environment* varijabli koje smo već spomenuli, moguće je definirati i konstante odnosno varijable koje se neće mijenjati. Za to se koristi naredba `readonly` iza koje slijede varijabla i njena vrijednost.

**Primjer upotrebe, odnosno definiranja konstanti**
Definirajmo konstantu (nepromjenjivu varijablu) naziva `PI` koja će imati vrijednost `3.14`. To postižemo na sljedeći način:

```
readonly PI=3.14
```

**Izvori informacija: (1569), `man bash`,`man echo`.**

---

## 6.2.2. Sistemske (*Environment*) varijable i postavke terminala

Sistemske (engl. *Environment*) varijable odnosno varijable okruženja su varijable koje koristi cijeli operativni sustav kao i svi programi u radnom terminalu. To su varijable koje su vidljive i u svim ljuskama (*shell*-ovima). Ovo su praktično eksportirane standardne varijable te su sâmim time vidljive svugdje, te ih mogu koristiti sve komponente sustava, kao i svi programi.
Ovim varijablama se postavljaju određene opcije i parametri važni za funkcioniranje cijelog sustava i sistemskih programa odnosno naredbi.
Ove varijable su definirane u nekim od sistemskih konfiguracijskih datoteka (pr. `/etc/profile`) ili konfiguracijskih datoteka ljuske (*shella*). Osnovne sistemske varijable možemo vidjeti (ispisati) s naredbom `printenv`.

Pogledajmo i neke od njih (ispisali smo samo one koje su nam trenutno zanimljive):

```
printenv

HOSTNAME=server1.domena.hr
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=192.168.10.50 38501 22
SSH_TTY=/dev/pts/2
USER=root
MAIL=/var/spool/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
PWD=/root
LANG=en_US.UTF-8
HOME=/root
LOGNAME=root
SSH_CONNECTION=192.168.10.50 38501 192.168.1.101 22
VISUAL=vi
```

Objasnit ćemo samo neke od ovih varijabli:

- HOSTNAME je ime računala odnosno poslužitelja (engl. *hostname*).
- SHELL nam govori koja ljuska se koristi (u ovom slučaju je to *bash ljuska*).
- HISTSIZE je veličina ***history*** memorije; u ovom slučaju se pamti 1.000 zadnje upisanih naredbi.
- SSH_CLIENT nam govori s koje IP adrese smo se spojili na SSH poslužitelj (192.168.10.50), koji je lokalni TCP port (38501) na klijentu te TCP port SSH servisa (22).
- SSH_TTY označava na koji smo virtualni *terminal* spojeni preko SSH protokola (konkretno /dev/pts/2).
- USER označava trenutnog korisnika (dakle mi smo logirani kao korisnik *root*).
- MAIL označava datoteku u koju nam se pohranjuju *e-mailovi* (za našeg korisnika).
- PATH daje nam listu putanja (Engl. *Path*) do izvršnih programa. Direktoriji u putanji su odvojeni sa znakom :
- PWD pokazuje nam u kojem se trenutnom direktoriju nalazimo (engl. *Print Working Directory*).
- LANG definira jezik i kôdnu stranicu (u našem slučaju je to engleski te **UTF-8** kôdna stranica).
- HOME definira direktorij korisnika (engl. *home*) koji je logiram (u našem slučaju smo prijavljeni kao korisnik *root* pa je to direktorij: /root/).
- LOGNAME je ime korisnika koji je prijavljen (*logiran*).
- SSH_CONNECTION nam govori o *SSH* konekciji : klijentska IP (192.168.10.50), klijentski TCP lokalni port (38501), *SSH* poslužitelj i njegova IP (192.168.1.101) i njegov TCP port (22).
- VISUAL definira koji je uređivač teksta postavljen kao standardni (engl. *Default*): u našem slučaju je to ***VI***.

Sistemske kao i druge varijable mogu i ne moraju, biti eksportirane odnosno javno vidljive svima. Da bismo provjerili koje su sve varijable trenutno eksportirane, možemo pokrenuti sljedeću naredbu (vidjet ćemo oznaku declare ispred svake od njih):

```
export -p
```

Ako želimo vidjeti sve sistemske, ali i sve varijable ljuske koje su trenutno u upotrebi, za to možemo pokrenuti naredbu set. Upotrebom naredbe set ćemo dobiti znatno veću listu varijabli koje sadrže i one gore navedene, pa ćemo stoga filtrirati njih samo nekoliko:

```
set

HOSTTYPE=x86_64
OSTYPE=linux-gnu
COLUMNS=151
LINES=45
UID=0
EUID=0
PPID=18749
PS1='[\u@\h \W]\$ '
SHLVL=1
TERM=xterm
DISPLAY=localhost:10.0
```

Sada ćemo vidjeti i njihovo značenje (objasnit ćemo ih ponovno samo nekoliko):

- HOSTTYPE prikazuje arhitekturu procesora (**x86_64** = 64. bitni x86 kompatibilni procesor).
- OSTYPE pokazuje koji operativni sustav je u pitanju (***Linux***).
- COLUMNS pokazuje koliko slova stane u jedan red (trenutno). Mijenja se ovisno kako proširujemo ili skupljamo prozor terminala.
- LINES slično kao prethodan varijabla samo što nam govori koliko ukupno redova ima naš trenutni terminal.
- UID nam daje ***UID*** (User ID) trenutnog korisnika (0=root).
- EUID nam daje efektivni ***UID*** - tj. ***UID*** od trenutnog korisnika, ako smo mijenjali korisnika tj. sa jednog korisnika prešli u drugog. Za ovakav rad, pogledajte naredbu su u poglavlju: **7.1.5.**
- PPID (*Parent Proces ID*) ***PID*** broj od roditeljskog procesa s kojim smo spojeni na sustav. U našem slučaju 18749 je ***PID*** (*Process* ID) procesa koji je pokrenuo SSH servis koji je pokrenuo našu ljusku (*shell*):

Pogledajmo procese s naredbom ps, pa će nam sve biti jasnije (dodali smo *opise* dolje); pogledajte odnos donja dva procesa: ***PID*** i ***PPID*** brojeve. ***PID*** je identifikator našeg procesa, a ***PPID*** identifikator roditeljskog procesa koji je pokrenuo naš proces.

```
ps -ef |grep 18749 | grep -v grep

ID :      PID:  PPID:
root     18749  1332   0 11:01 ?        00:00:00 sshd: root@pts/3
root     11097 18749   0 11:01 pts/3    00:00:00 -bash
```

- PS1 ovdje su definirane postavke sučelja (engl. *Prompt settings*) odnosno svega onoga što želimo vidjeti u radu u ljusci (*shellu*), prije pozivanja svake naredbe. Dakle sve do znaka #, nakon kojeg radimo u ljusci (*shellu)* tj. pozivamo naredbe ili skripte.
  U našem slučaju to izgleda ovako:

```
[root@SERVER1 ~]#
```

Naša varijabla PS1 ima vrijednost: [\u@\h \W]\$.

Što ona znači, uz još nekoliko drugih objašnjenih vrijednosti:

    - `\u` - “*Username*” - prikazuje ime trenutnog korisnika (u primjeru gore je to: `root`).
    - `\h` - “*Hostname*” - prikazuje ime računala (u primjeru gore je to: `SERVER1`).
    - `\W` - prikazuje samo trenutni vršni direktorij u kojem se nalazimo.
    - `\t` - prikazuje trenutno vrijeme (sat).
- `SHLVL` varijabla nam govori koja je razina *shella* u kojem se nalazimo. Dakle ako je vrijednost **1** tada je ovo prvi *shell*. Ako pokrenemo novi *shell* (pôd/*subshell*) vrijednost će biti **2**, ako iz drugog *shella* pokrenemo još jedan pôd *shell*, vrijednost će biti **3** itd.
- `TERM` varijabla označava vrstu terminala koja se emulira. Terminala postoji cijeli nîz. Najčešće se koriste `xterm`, `vt100` ili `linux`, `putty` i slično. **O njihovom odabiru ovisi ponašanje emuliranog terminala (prikaza na ekranu)!.**

Listu podržanih terminala možemo vidjeti upotrebom naredbe `toe`. Pogledajmo skraćenu listu podržanih terminala na sustavu:

```
toe /usr/share/terminfo/

konsole         KDE console window
gnome           GNOME Terminal
putty           PuTTY terminal emulator
putty-vt100     VT100+ keyboard layout
vt102           dec vt102
vt100           dec vt100 (w/advanced video)
vt220           dec vt220
vt52            dec vt52
ansi            ansi/pc-term compatible with color
xfce            Xfce Terminal
xterm           xterm terminal emulator (X Window System)
```

Terminali su zaduženi za baratanje i prikaz teksta, posebnih znakova i signala te takozvane “*Escape*” sekvence unutar našeg radnog terminala odnosno ljuske. Standardni terminal je onaj na koji se spajamo direktno dok se nalazimo ispred računala.

S druge strane terminal s kojim i na koji se spajamo može biti ostvaren i udaljenom vezom preko *SSH* ili *Telnet* protokola odnosno pomoću *SSH* ili telnet klijenata, a koji su zapravo aplikacije za spajanje na sâm terminal. Za posebne kontrolne znakove pogledajte ***ASCII*** tablicu, na kraju knjige, i to prvi dio tablice: 0 - 31. *Escape* sekvencu može činiti i kombinacija **`ESC`** kôda i nekog od *ASCII* znakova. Svaka od *Escape* sekvenci naravno ima posebno značenje odnosno funkciju.

U konačnici, vrsta terminala odnosno vrsta emulirane vrste terminala definira način na koji obje strane komuniciraju. Bit će vam jasnije kada pogledate primjere koji slijede.

Svaki od terminala ima neke specifičnosti. Loše usklađena vrsta terminala između poslužitelja i klijenta može uzrokovati probleme u radu, a najčešće se manifestira krivim prikazom nekih znakova (alfabeta/posebnih znakova i slično) na ekranu, ili nemogućnošću upotrebe određenih poziva, upotrebe funkcijskih tipki (pr. **F1**, **F2**,…**F12** ili sl.) i slično.

Ovi problemi mogu biti usko povezani s krivim odabirom kôdne stranice odnosno translacije karaktera (pr. *UTF-8*, *ISO-8859-1*, *ISO8859-2* i drugih), kao i iscrtavanja posebnih znakova (box-drawing*****) na ekranu.



Pogledajte i sljedeća poglavlja:
**11.1.2 Uređaji (devices) ukratko** – (***Terminali***).
**3.1. Unix/Linux ljuska *(Shell)* i terminali.**

**Slika 17. Primjer konfiguracije: TERM=*vt100*, uz** *UTF-8*, za program `mc` (*Midnight Commander*)

```
  Left      File      Command     Options     Right
┌<- ~ ─────────────────────────────.[^]>┐┌<- ~ ─────────────────────────────.[^]>┐
|'n        Name           | Size  |Modify time ||'n        Name           | Size  |Modify time |
|/..                      |UP--DIR|Mar 16 21:51||/..                      |UP--DIR|Mar 16 21:51|
|/.cache                  |   4096|Jul  2  2014||/.cache                  |   4096|Jul  2  2014|
|/.config                 |   4096|May 20  2015||/.config                 |   4096|May 20  2015|
|/.gem                    |   4096|Sep 24  2013||/.gem                    |   4096|Sep 24  2013|
|/.gerritcodereview       |   4096|Sep 30  2013||/.gerritcodereview       |   4096|Sep 30  2013|
|/.gnome2                 |   4096|Sep 29  2013||/.gnome2                 |   4096|Sep 29  2013|
|/.ivy2                   |   4096|Sep 14  2014||/.ivy2                   |   4096|Sep 14  2014|
|/.local                  |   4096|Jun 30  2015||/.local                  |   4096|Jun 30  2015|
|/.mc                     |   4096|Mar 18 16:07||/.mc                     |   4096|Mar 18 16:07|
|/.mozilla                |   4096|Sep 28  2013||/.mozilla                |   4096|Sep 28  2013|
|/.npm                    |  20480|Jan 16 23:38||/.npm                    |  20480|Jan 16 23:38|
|/.pki                    |   4096|Aug 13  2013||/.pki                    |   4096|Aug 13  2013|
|/.sbt                    |   4096|Sep 14  2014||/.sbt                    |   4096|Sep 14  2014|
```

Osim što smo izgubili boju, ne rade niti funkcijske tipke: **F1**, **F2**, …, **F12**.

**Slika 18. Primjer konfiguracije: TERM=***linux* **ili TERM=***putty***, uz** *UTF***-8,** za program `mc`

```
  Left       File       Command      Options       Right
┌<─ ~ ─────────────────────────────────.[^]>┐┌<─ ~ ─────────────────────────────────.[^]>┐
│.n      Name          │  Size  │Modify time ││.n      Name          │  Size  │Modify time │
│/..                   │UP--DIR │Oct  8 13:21││/..                   │UP--DIR │Oct  8 13:21│
│/.cache               │     27 │Oct 22 13:29││/.cache               │     27 │Oct 22 13:29│
│/.config              │     42 │Apr  6  2020││/.config              │     42 │Apr  6  2020│
│/.local               │     19 │Jan 24  2020││/.local               │     19 │Jan 24  2020│
│/.ssh                 │     25 │Apr 21  2020││/.ssh                 │     25 │Apr 21  2020│
│/bkp                  │     27 │Jun  3 09:04││/bkp                  │     27 │Jun  3 09:04│
│/test                 │     86 │Oct 26 12:48││/test                 │     86 │Oct 26 12:48│
│ .bash_history        │  19960 │Nov  2 14:44││ .bash_history        │  19960 │Nov  2 14:44│
│ .bash_logout         │     18 │May 11  2019││ .bash_logout         │     18 │May 11  2019│
│ .bash_profile        │    176 │May 11  2019││ .bash_profile        │    176 │May 11  2019│
│ .bashrc              │    176 │May 11  2019││ .bashrc              │    176 │May 11  2019│
│ .cshrc               │    100 │May 11  2019││ .cshrc               │    100 │May 11  2019│
│ .lesshst             │    426 │Oct 23 11:58││ .lesshst             │    426 │Oct 23 11:58│
│ .nmcli-history       │    167 │Jun  2 14:44││ .nmcli-history       │    167 │Jun  2 14:44│
│ .tcshrc              │    129 │May 11  2019││ .tcshrc              │    129 │May 11  2019│
├────────────────────────────────────────────┤├────────────────────────────────────────────┤
│UP--DIR                                     ││UP--DIR                                     │
└───────────────────────── 11G/15G (73%) ────┘└───────────────────────── 11G/15G (73%) ────┘
```

U ovoj kombinaciji sve radi i lijepo je iscrtano jer smo uključili iscrtavanje posebnih znakova* sa: „***Use Unicode Line drawing***".

**Slika 19. Primjer konfiguracije: TERM=***linux* **ili TERM=***putty***, uz** *ISO-8859-1:1998* **(Latin-1, West Europe)**, program `mc`

```
  Left      File      Command     Options     Right
â<â ~ ââââââââââââââââââââââââââââââââââ.[^]>ââ<â ~ ââââââââââââââââââââââââââââââââââ.[^]>â
â'n        Name           â Size  âModify time ââ'n        Name           â Size  âModify time â
â/..                      âUP--DIRâMar 16 21:51ââ/..                      âUP--DIRâMar 16 21:51â
â/.cache                  â   4096âJul  2  2014ââ/.cache                  â   4096âJul  2  2014â
â/.config                 â   4096âMay 20  2015ââ/.config                 â   4096âMay 20  2015â
â/.gem                    â   4096âSep 24  2013ââ/.gem                    â   4096âSep 24  2013â
â/.gerritcodereview       â   4096âSep 30  2013ââ/.gerritcodereview       â   4096âSep 30  2013â
â/.gnome2                 â   4096âSep 29  2013ââ/.gnome2                 â   4096âSep 29  2013â
â/.ivy2                   â   4096âSep 14  2014ââ/.ivy2                   â   4096âSep 14  2014â
â/.local                  â   4096âJun 30  2015ââ/.local                  â   4096âJun 30  2015â
â/.mc                     â   4096âMar 18 16:14ââ/.mc                     â   4096âMar 18 16:14â
â/.mozilla                â   4096âSep 28  2013ââ/.mozilla                â   4096âSep 28  2013â
â/.npm                    â  20480âJan 16 23:38ââ/.npm                    â  20480âJan 16 23:38â
â/.pki                    â   4096âAug 13  2013ââ/.pki                    â   4096âAug 13  2013â
â/.sbt                    â   4096âSep 14  2014ââ/.sbt                    â   4096âSep 14  2014â
```

U ovoj kombinaciji (slika 19) rade sve tipke (pr.: **F1**, **F2**, …, **F12**), ali imamo problema sa prikazom. Pogledajte rubove i slova `â` umjesto posebnih znakova* korištenih za iscrtavanje rubova. Ako vas zanima koji terminal uređaji se koriste na sustavu.

U slučaju kada vam se nešto poremeti unutar terminala, terminal možete resetirati upotrebom naredbe `reset`, ovako:

```
reset
```

*Za detalje oko konfiguracije terminala, proučite i naredbe:* `stty`, `tty` *i* `tput`.

I konačno zadnja varijabla (s prethodne stranice) koju smo spomenuli jest:

- `DISPLAY` - za konfiguracija "*ekrana/monitora*" za ***X Window*** sustav.

Pogledajte poglavlje:
**18. X Window sustav.**

**Izvori informacija: (817),(K-1),(K-12),** `man bash`, `man toe`, `man reset`, `man printenv`.

#### 6.2.2.1. Posebne sistemske varijable

Prema ***POSIX*** (engl. *Portable Operating System Interface*) standardima, kojih se osim UNIXa drži i Linux, definirano je aplikacijsko programsko sučelje odnosno ***API*** (engl. *Application programming interface*), zajedno s naredbenim retkom (ljuskom) i sučeljima za razne programe, kao i cijelo okruženje koje uključuje i posebne sistemske varijable.
Trenutno aktivni ***POSIX*** standard je ***IEEE Std 1003.1-2017***. Kako bismo uopće mogli vidjeti posebne **POSIX** varijable, moramo koristiti naredbu: `getconf`.



Pogledajte i poglavlja:
**10.4.4.1 POSIX.**
**4.5.6. Ograničenja sustava i naredba *ulimit*.**
**5.9.5 Naredba *xargs* i ograničenje duljine argumenata.**

Konkretno nam naredba `getconf -a` daje ispis svih ***POSIX*** sistemskih varijabli i njihovih vrijednosti.
Pogledajmo samo neke od njih, iako ih je standardno definirano nekoliko stotina:

```
getconf -a

ARG_MAX                            2097152
LINK_MAX                           2147483647
NAME_MAX                           255
PATH_MAX                           4096
OPEN_MAX                           65535
CHILD_MAX                          65535
LINE_MAX                           2048
LOGNAME_MAX                        256
```

Opisat ćemo samo neke od njih uz napomenu kako na višoj razini mogu postojati i postoje ograničenja koja su manja ili veća od onih postavljenih ovdje. Dakle u ovim varijablama su postavljene vrijednosti koje bi se trebale garantirati, ali one efektivno u upotrebi se mogu smanjiti ili povećati prema potrebi (pr. s naredbom `ulimit`).



Neke od postavljenih vrijednosti definiranih ovdje se ne mogu povećavati jer su zapisane kao nepromjenjive odnosno ugrađene („*hard kodirane*“) vrijednosti.

Stoga pogledajmo opise nekih od njih:

- `ARG_MAX` – varijabla označava maksimalan broj argumenata ili nîza naredbi koje možemo pokrenuti u nizu.
  Postoji i ograničenje na maksimalnu veličinu jednog argumenta, a ono je nepromjenjivo (*hard kodirano)*:
  **MAX_ARG_STRLEN = PAGESIZE x 32 ~ 131072**
- `LINK_MAX` – varijabla označava maksimalan broj linkova na pojedinu datoteku.
- `NAME_MAX` – varijabla označava maksimalnu dužinu imena datoteke.
- `PATH_MAX` – varijabla označava maksimalnu veličinu `PATH` varijable u bajtima.
- `OPEN_MAX` – varijabla označava maksimalni broj otvorenih datoteka za svaki pojedini proces.
- `CHILD_MAX` – varijabla označava maksimalan broj svih pokrenutih procesa za pojedinog korisnika.
- `LINE_MAX` – varijabla označava maksimalan broj karaktera (znakova) koje program definiran za prihvat ili obradu teksta može prihvatiti kao jedan ulazni nîz podataka (na standardni ulaz).
- `_NPROCESSORS_CONF` – varijabla označava maksimalan broj CPU jezgri koji je uopće dostupan na računalu.
- `_NPROCESSORS_ONLN` – varijabla označava maksimalan broj CPU jezgri koji može biti dostupan Linux *task/process. scheduleru*, a koje su samim time dostupne svim programima/aplikacijama o čijem radu se i brine *task/process scheduler*.

Ako imamo potrebu, možemo zatražiti vrijednost sâmo pojedine **POSIX** varijable; primjerice: `_NPROCESSORS_ONLN` sa:

```
getconf _NPROCESSORS_ONLN
```

I dobit ćemo njenu vrijednost.

**Izvori informacija: (594),(595), `man getconf`.**

#### 6.2.2.2. Ugrađene, posebne varijable ljuske

Osim navedenih standardnih (običnih) varijabli koje možemo sami definirati i koristiti, kao i varijabli sustava te posebnih sistemskih varijabli *bash* ljuska ima i nekoliko specijalnih rezerviranih varijabli za posebne namjene, od kojih ćemo neke i navesti. Njihovu vrijednost možemo izvlačiti na više načina (primjerice: `echo XY`):

- `$0`,`$1`,`$2`,..,`$9` su (**6.3. Parametri pozicije**) varijable koje označavaju parametre definirane prema poziciji
  - `$*` nam daje sve vrijednosti opcija (iz svih parametara pozicija:`$0-$9`): u nîzu.
  - `$@` nam daje sve vrijednosti opcija (iz svih parametara pozicija:`$0-$9`): svaki zasebno.
- `$#` nam daje vrijednost koliko se argumenata koristilo u nekoj naredbi (**6.1. Kako rade Shell skripte**).
- `$?` nam daje statusni kod prethodno izvršene naredbe (**6.7. Nizanje i ulančavanje naredbi i statusni kodovi**).
- `$-` nam daje sve zastavice (*flags*) koje su proslijeđene nekoj skripti.

**Izvor informacija: (751),(K-1),(K-12).**

---

### 6.2.3. Varijable polja (*Array variable*)

Osim gore navedenih tipova varijabli, možemo koristiti i posebne varijable polja (engl. *Array*).
Dakle ako imamo potrebu unutar jedne varijable upisati više vrijednosti, koristit ćemo varijablu polja, prema principu:

```
POLJE=(VRIJEDNOST1 VRIJEDNOST2 VRIJEDNOST3 ..)
```

**Primjeri**
**1.** Kreirajmo jedno polje (varijablu polja) koje će sadržavati tri vrijednosti: `jedan`, `dva`, `tri`.

```
POLJE=(jedan dva tri)
```

Naše polje sada ima tri elementa, kako doći do pojedinog elementa u polju?.
Jedan od načina je korištenje vitičaste zagrade za polje unutar koje možemo definirati elemente unutar uglatih zagrada.

Kao na primjeru, gdje ćemo ispisati prvi element polja `[0]` (elementi polja se broje od 0).

```
echo ${POLJE[0]}

jedan
```

Ako želimo ispisati sve elemente polja, moramo koristiti `[*]`. Odnosno to bi izgledalo ovako:

```
echo ${POLJE[*]}

jedan dva tri
```

**2.** Napravimo `for` petlju s kojom ćemo ispisati elemente polja, jedan po jedan.
Stoga kreirajmo datoteku imena: `for.polje.sh`, koja će sadržavati sljedeće retke:

```
#!/bin/bash
POLJE=(jedan dva tri)

for PETLJA in ${POLJE[@]}; do
 echo $PETLJA

done
```

Opis:
`POLJE=(jedan dva tri)` kreirali smo polje s tri elementa
`for PETLJA in ${POLJE[@]}; do` kreiramo `for` petlju, koja internu varijablu `PETLJA` popunjava s elementima polja i to u svakom prolazu sa jednim elementom polja.
Sa `echo $PETLJA` ispisujemo varijablu `PETLJA` (u svakom prolazu `for` petlje) .

Promijenimo ovlasti naše skripte kako bismo ju mogli pokrenuti. To ćemo napraviti pomoću naredbe `chmod`:

```
chmod +x for.polje.sh
```

Zatim pokrenimo našu skriptu na sljedeći način:

```
./for.polje.sh

jedan
dva
tri
```

Vidimo da smo kao ispis naše skripte dobili ispis svih elemenata polja.

#### 6.2.3.1. Brisanje polja ili elemenata polja

Moguće je obrisati cijelo polje, primjerice naše polje imena `POLJE`:
```
unset POLJE
```
ili možemo brisati samo pojedine elemente unutar polja. Primjerice obrišimo samo drugi element `[1]` polja:
```
unset POLJE[1]
```
Pogledajmo što smo dobili izbacivši drugi element polja:
```
echo ${POLJE[*]}

jedan tri
```
Vidimo da je drugi element uredno izbačen.

**Izvori informacija: (817),(K-12), `man set`,`man unset`.**

---

### 6.2.4. Operacije nad varijablama

Nad varijablama je moguće raditi razne operacije, pogledajmo i koje. Za brojanje duljine varijable ili prebrojavanje broja elemenata polja je sintaksa slijedeća: `${#VAR}`, pri čemu je `VAR` ime varijable.

Prebrojimo broj elemenata polja koje smo kreirali; varijabla polja se zove: `POLJE`. Probajmo to sada učiniti:
```
echo ${#POLJE}

3
```
U slučaju kada bi to bila obična varijabla, dobili bi rezultat prebrojavanja alfabeta unutar vrijednosti varijable.
Sada kreirajmo varijablu `INVENTAR` sa vrijednosti `monitor`
```
INVENTAR=monitor
```

Prebrojimo vrijednost varijable (konkretno se radi o riječi: `monitor`).
```
echo ${#INVENTAR}

7
```
Dobili smo broj sedam (`7`), jer ova riječ (`monitor`) koja je postavljena u varijabli `INVENTAR` ima sedam slova.

---

#### 6.2.4.1. Rad s uzorcima od vrijednosti varijable

U narednom tekstu, proći ćemo mogućnosti rada s uzorcima koje možemo primjenjivati na vrijednosti varijabli novijih inačica *bash* ljuske. **Neke starije inačice *bash* ljuske (3.x ili neke 4.x) nemaju sve od dolje navedenih mogućnosti.**

**Rad s prednjim djelom uzorka vrijednosti varijable**
Postoji i mogućnost izuzimanja određenih uzoraka od vrijednosti varijable, upotrebom znakova **`#`** i **`##`**.
Pogledajmo kako:

- Znak `#` predstavlja prvi uzorak na koji će se naići od početka.
- Znak `##` predstavlja zadnji uzorak na koji će se naići od početka (ovisi o *ljusci*).

Sintaksa je sljedeća:
```
${VARIJABLA#UZORAK}
```

Zamislimo da imamo varijablu imena `var` čija je vrijednost: `1234abcd1234abcd`. Stoga ju i kreirajmo na sljedeći način:
```
var=1234abcd1234abcd
```
Sada, ako želimo izuzeti prvi dio (`1234`) od vrijednosti varijable, krenuti ćemo s izuzimanjem navedenog uzorka od početka:
```
echo ${var#1234}

abcd1234abcd
```
Dakle izbačen je početni uzorak `1234` kako smo i htjeli.

Ako s druge strane želimo krenuti od najdaljeg uzorka koji zadovoljava, krenuvši od početka, tada ćemo koristiti **`##`**:
```
echo ${var##1234}

abcd
```
I ovdje vidimo da je uredno izbačeno sve do uzorka s kraja (`1234`).

**Rad sa stražnjim dijelom uzorka vrijednosti varijable**
Sada ćemo raditi sa stražnjim djelom uzorka varijable, pomoću dva operatora:

- Znak `%` predstavlja prvi uzorak na koji će se naići od kraja prema početku.
- Znak `%%` predstavlja zadnji uzorak na koji će se naići od kraja prema početku (ovisi o *ljusci*).

Izostavimo zadnji uzorak od kraja tj. prvi od kraja prema početku, koji zadovoljava. Uzorak je `abcd`

```
echo ${var%abcd}

1234abcd1234
```

Zatim probajmo izostaviti zadnji uzorak koji zadovoljava, od kraja prema početku:

```
echo ${var%%abcd}

1234
```

**Izvori informacija: (817),(K-12), `man echo`,`man bash`.**

---

### 6.2.4.2. Izvlačenja dijela vrijednosti varijable

Moguće je izvući i dio vrijednosti varijable, prema sljedećem principu:

```
${VARIJABLA:OFFSET:LENGTH}
```

Pri tome je:

- `OFFSET` - odmak s lijeve strane od početka vrijednosti varijable.
- `LENGTH` - je broj karaktera (znakova) koliko želimo da se uzima u obzir.

**Primjer**i
Postavimo novoj varijabli `INVENTAR` vrijednost `monitor` na sljedeći način:

```
INVENTAR=monitor
```

Iz vrijednosti postojeće varijable `INVENTAR` izvucimo slova, s odmakom od četiri pozicije (slova) u nîzu i to samo dva slova:

```
echo ${INVENTAR:4:2}

to
```

---

## 6.3. Parametri pozicije

***Slijedi napredno poglavlje!***
U radu sa skriptama, dosta često možemo imati potrebu prosljeđivanja više parametara našoj skripti. Primjerice, želimo pri pokretanju naše skripte imati mogućnost definiranja nekoliko parametra. Za tu namjenu se koriste parametri pozicije.
Parametri pozicije imaju sljedeće oznake na sustavu:

- `$0` - označava sâm program koji pokrećemo (tj. našu skriptu).
- `$1` - označava prvi parametar (prva pozicija) koji prosljeđujemo.
- `$2` - označava drugi parametar (druga pozicija) koji prosljeđujemo.
- `$3` - označava treći parametar (treću poziciju) koji prosljeđujemo.

**Primjeri**
1. Napravimo skriptu koja zbraja dva broja, na način da pozivamo našu skriptu s dva parametra: prvi parametar je prvi broj, a drugi parametar je drugi broj. Stoga kreirajmo sljedeću skriptu: `pozicija-zbroj.sh` koja će sadržavati:

```
#!/bin/bash

echo "Zbroj je : "
echo $(( $1 + $2 ))
```

Promijenimo joj ovlasti i pokrenimo našu skriptu, s time da ćemo joj dati brojeve `5` i `4` kao parametre (kako bi ih skripta zbrojila) :

```
chmod +x pozicija-zbroj.sh
```

Pokrenimo ju iz trenutnog direktorija, s navedena dva parametra: `5` i `4`

```
./pozicija-zbroj.sh 5 4
Zbroj je :
9
```

Da bi bilo malo jasnije: pokretanjem naše skripte sa `./pozicija-zbroj.sh 5 4` prosljeđujemo joj dva parametra sa vrijednostima: `5` (ovo je prva pozicija odnosno `$1`) te `4` (ovo je druga pozicija odnosno `$2`).
Kasnije u našoj skripti, čitamo vrijednost (`5`) sa prve pozicije tj. `$1` odnosno vrijednost (`4`) sa druge pozicije tj. `$2`.

Standardni parametri pozicije su: `$1`, `$2`, `$3`, `$4`, `$5`, `$6`, `$7`, `$8` i `$9`. U slučajevima kada nam je potrebno više od devet (9) parametara pozicije, moramo ih pozivati unutar vitičastih zagrada.
Primjerice 10-ti parametar pozicije je `${10}` ili 11-ti: `${11}` i tako dalje.

**Izvori informacija: (K-12), `man bash`.**

---

# 6.4. Upotreba okruglih zagrada ( )

***Slijedi napredno poglavlje (6.4.x)!.***
U ***bash*** ljusci je moguće koristiti i okrugle zagrade. U ljusci zapravo razlikujemo dvije upotrebe okruglih zagrada: jednostruke okrugle zagrade **`( )`** i dvostruke okrugle zagrade **`(( ))`**.

## 6.4.1. Jednostruke okrugle zagrade

Jednostruke okrugle zagrade imaju najčešću primjenu:
- Tijekom pozivanja odnosno pokretanja naredbi u posebnoj pôd ljusci (*shellu*).
- Kod varijabli polja.

Proći ćemo neke od primjena upotrebe jednostrukih zagrada kroz primjere.

**Primjeri**
**1.** Primjena kod kreiranja varijabli polja. Kreirajmo varijablu polja s dva elementa: `prvi` i `drugi`

```
varijabla=(prvi drugi)
```

**2.** Slijedeća primjena bi bila upotreba u slučajevima kada za vrijednost varijable želimo postaviti rezultat pozivanja neke naredbe ili nîza naredbi. Pogledajmo sljedeću skriptu:

```
#!/bin/bash

integritet=$(md5sum datoteka.txt)
echo $integritet
```

U prethodnom primjeru, varijabli `integritet` dodjeljujemo vrijednost koja će se dobiti pozivanjem naredbe: `md5sum datoteka.txt`. Navedena naredba će napraviti provjeru integriteta datoteke `datoteka.txt`, korištenjem ***MD5*** algoritma. Nakon toga u novom redu očitavamo njenu vrijednost sa: `echo $integritet`. Pokrenimo gore navedenu skriptu:

```
./test.sh

ab8e75a54297fb9a107c263cc3cfbe68 datoteka.txt
```

U prethodnom ispisu, dobili smo ***MD5*** provjerni zbroj kao rezultat ispisa (`ab8e75a54297fb9a107c263cc3cfbe68`).



Pogledajte i poglavlja:
**7.1.3 Datoteka /etc/shadow** - i to dio: **7.1.3.1 Provjerni zbroj (hash/checksum)**
**5.12.1.2 Upotreba navodnika ' " ` (jednostruki, dvostruki i jednostruki kosi)** i to upotrebu jednostrukih kosih navodnika s kojima možemo dobiti istu funkcionalnost kao kod ovakve upotrebe okruglih zagrada.

Pripazite:
- Sve naredbe koje se izvršavaju u nîzu (unutar okruglih zagrada), bit će izvršene u posebnoj pôd ljusci.
- Sve varijable koje ćete koristiti u toj pôd ljusci, ostaju lokalne za tu ljusku te se ne nasljeđuju u vršnu *ljusku*. Dakle kada se ta pôd *ljuska* zatvori, gube se i njene varijable.

**3.** Druga slična primjena je direktno kreiranje posebne pôd ljuske i izvršavanja naredbi unutar nje. Za opis rada pôd ljuske, prisjetite se te poglavlja: **6.1. Kako rade Shell skripte**. U ovom slučaju se radi o pôd ljusci, koja se ne inicijalizira kao standardna pôd ljuska sa svim *environment* varijablama. Sjetimo se da su varijable koje se nalaze u pôd ljusci lokalne za tu ljusku te se ne prenose na vršnu ljusku odnosno nisu vidljive u vršnoj ljusci.
Provjerimo trenutni radni direktorij u kojem se nalazimo, s naredbom `pwd`:

```
pwd

/home/korisnik1
```

Dakle to je `/home/korisnik1`
Sada ćemo pozvati posebnu pôd ljusku i u njoj izvršiti određene naredbe.

Konkretno ćemo ući u direktorij `/tmp` i ispisati u kojem se direktoriju nalazimo.

```
(cd /tmp; pwd)

/tmp
```

Dobili smo podatak da se nalazimo u direktoriju `/tmp` što je u redu.
Sada kada se posebna pôd ljuska s naredbama koje smo pokretali u nîzu, završila, pogledajmo u kojem direktoriju se nalazimo:

```
pwd

/home/korisnik1
```

Vidimo kako se nalazimo u direktoriju `/home/korisnik1` u kojem smo i bili iz glavne ljuske. Ovo je dokaz da su sve naredbe koje smo pokretali u posebnoj pôd ljuski, bile izvršene sâmo unutar nje, te da nisu imale utjecaj na našu vršnu ljusku.

**4.** Moguće je koristiti i "***Pipe***" mogućnost ( | ) unutar okruglih zagrada. Pogledajmo primjer.
U primjeru želimo pronaći sve datoteke unutar direktorija `/usr/bin` koje u imenu imaju riječ `zip` te s naredbom `file` provjeriti koje su vrste te datoteke (skripta/binarne/…). Pri tome ćemo malo skratiti ispis naredbe `file`:

```
file $(ls /usr/bin/* | grep zip)

... ... ...
/usr/bin/bunzip2:       symbolic link to `bzip2'
/usr/bin/bzip2:         ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
/usr/bin/gunzip:        symbolic link to `../../bin/gunzip'
/usr/bin/gzip:          symbolic link to `../../bin/gzip'
... ... ...
```

Dakle ovdje se prvo izvršava: `ls /usr/bin/* | grep zip`, a potom se na svaki red tog ispisa pokreće naredba `file`.

**Izvori informacija: (817),(K-12), `man bash`,`man file`.**

## 6.4.2. Dvostruke okrugle zagrade (())

Dvostruke okrugle zagrade se koriste za aritmetičke operacije.
Sintaksa ovakvog korištenja bi bila: `$(( ARITMETIČKE OPERACIJE ))`
Pomnožit ćemo dva broja (2 x 4) na sljedeći način:

```
echo $((2*4))

8
```

U gornjem primjeru s naredbom `echo` ispisujemo vrijednost `$` aritmetičke operacije unutar dvostrukih okruglih zagrada. Konkretno tražimo rezultat operacije množenja 2 x 4 koji dobivamo nakon pokretanja ove naredbe. Dakle dobivamo: ***8***.
Osim jednostavnih, moguće je raditi i složenije aritmetičke operacije, poput: (2+3)x4, što ćemo izvesti ovako:

```
echo $(( (2+3) *4 ))

20
```

Drugi primjer je upotreba unutar neke petlje, poput `for` petlje. Naime osim na "klasičan" način možemo petlju kreirati i korištenjem dvostrukih okruglih zagrada, na sljedeći način:

```
...
for ((i=0; i<10; i++))
...
```

Ovo znači, kako će petlja varijablu `i` povećavati od `0`, u koraku po jedan, sve dok je `i` manje od `10`.



Pogledajte i poglavlje:
**6.6.1. *For* petlja.**

**Izvori informacija: (817),(K-12), `man bash`**

### 6.4.3. Redirekcija (preusmjeravanje) i okrugle zagrade

Kako smo vidjeli, unutar okruglih zagrada, je moguće pozvati određenu naredbu ili nîz naredbi.
Dodatno, moguće je kombinirati višestruko preusmjeravanje odnosno *redirekcije* i pozivanje naredbi unutar okruglih zagrada, sve u jednom koraku, kako bi si skratili posao. Sintaksa za preusmjeravanje je upotreba naredbi unutar okvira okruglih zagrada i preusmjeravanja, bez razmaka: `<()`
Uzmimo primjer iz poglavlja: **10.7.2.2. NUMA** i to dio „***Vratimo se na naš stariji dvoprocesorski sustav***".
Ovdje smo iz dvije slijedeće datoteke:

- `/sys/devices/system/node/node0/numastat`
- `/sys/devices/system/node/node1/numastat`

izvlačili vrijednosti, prvo iz prve datoteke, potom iz druge, kako bismo ih na kraju povezali s naredbom `join`. Kada bi koristili preusmjeravanje (*redirekciju*) s okruglim zagradama (bez razmaka), to bi moglo izgledati i ovako (sve u jednom redu):

```
join  <(cat /sys/devices/system/node/node0/numastat) <(cat
/sys/devices/system/node/node1/numastat)
```

Što smo gore napravili:

- Ispisali smo sadržaj datoteke `/sys/devices/system/node/node0/numastat` s naredbom `cat` te ga preusmjerili na naredbu `join`.
- Istovremeno smo ispisali i sadržaj datoteke `/sys/devices/system/node/node0/numastat`, s naredbom `cat`, te ga isto preusmjerili na naredbu `join` koja je povezala obje datoteke u jedan ispis.

To možemo promatrati i ovako:
`NAREDBA_1 <(NAREDBA_2) <(NAREDBA_3)`

Odnosno u našem slučaju:
`join <(cat FILE-1) <(cat FILE-2)`



Važno je da između znaka preusmjeravanja odnosno redirekcije: **<** ili **>** te okruglih zagrada, nema razmaka.

**Izvor informacija: (817), `man join`, `man bash`.**

---

## 6.5. Uvjeti

***Slijedi napredno poglavlje!***
*Bash ljuska* nam nudi mogućnost korištenje uvjeta, poput onih u većini programskih jezika. Neki od češće korištenih uvjeta su:

- `if` – `else` su uvjetne naredbe (izrazi) odnosno konstrukti.
- `case` - za upravljanje tôkom uvjeta.

Postoje i uvjeti koji su definirani kao takozvani datotečno bazirani (engl. *File-based*) odnosno koji su dodatak standardnim uvjetima za slučajeve kada imamo potrebu unutar uvjeta provjeravati određena stanja datoteka ili direktorija (mapa).
Pri tome su često korišteni prekidači odnosno opcije provjere unutar ovih datotečnih uvjeta sljedeće:

- `-d` – provjera postoji li direktorij (mapa) (engl. *Directory*).
- `-e` – provjera postoji li datoteka (engl. *Existing*).
- `-f` – provjera postoji li datoteka i vrste je "obične" datoteke, dakle nije "*Block device*" ili sl.
- `-g` – provjera imaju li datoteka ili direktorij postavljen "***Set GID***" bît.
- `-r` – provjera postoji li datoteka i ima "*Read*" prava (**r**) odnosno prava čitanja.
- `-s` – provjera postoji li datoteka, te nije prazna.
- `-u` – provjera imaju li datoteka ili direktorij postavljen "***Set UID***" bît.
- `-w` – provjera postoji li datoteka i ima "*Write*" prava (**w**) odnosno prava zapisivanja.
- `-x` – provjera postoji li datoteka i ima "*Execute*" prava (**x**) odnosno prava izvršavanja.

Za posebne vrste datoteka postoje i provjere:

- `-b` – postoji li datoteka i vrste je blok uređaj (*Block device*), a obično se nalaze unutar */dev/* direktorija.
- `-c` – postoji li datoteka i vrste je karakter uređaj (*Character device*), a obično se nalaze unutar */dev/* direktorija.
- `-h` – postoji li datoteka i vrste je hard link "*Hard link*".
- `-L` – postoji li datoteka i vrste je: simbolički link.
- `-p` – postoji li datoteka i vrste je: "*Pipe*".
- `-S` – postoji li datoteka i vrste je: "*Socket*".

Osim toga postoje i logički uvjeti na stringovima, neki od njih su:

- `STRING1==STRING2` – što znači: *STRING1* je jednak *STRING2*.
- `STRING1!=STRING2` – što znači: *STRING1* nije jednak odnosno različit je od *STRING2*.
- `-z` – string je prazan (*empty*).
- `-n` – string nije prazan (*nonempty*).

I na kraju naravno postoje i aritmetički uvjeti:

- *BROJ1* `-eq` *BROJ2* – što znači kako je *BROJ1* jednak broju *BROJ2* (engl. *Equal*).
- *BROJ1* `-ne` *BROJ2* – što znači kako *BROJ1* nije jednak broju *BROJ2 (*engl. *Non equal).*
- *BROJ1* `-gt` *BROJ2* – što znači kako je *BROJ1* veći od broja *BROJ2* (engl. *Greater than*).
- *BROJ1* `-ge` *BROJ2* – što znači kako je *BROJ1* veći od ili jednak broju *BROJ2* (engl. *Greater than or equal*).
- *BROJ1* `-lt` *BROJ2* – što znači kako je *BROJ1* manji od broja *BROJ2* (engl. *Less than*).
- *BROJ1* `-le` *BROJ2* – što znači kako je *BROJ1* manji od ili jednak broju *BROJ2* (engl. *Less than or equal*).

**Primjeri**

**1.** Primjer upotrebe `if - else` uvjeta. U ovom primjeru tražimo od korisnika unos brojeva između **1** i **3** (uključujući) 3.
Za svaki broj koji smo unijeli pomoću višestrukih `if - else` uvjeta provjeriti koji broj smo upisali, a ako nije niti **1**, niti **2** niti **3** ispisat ćemo dodatnu poruku.
Kreirajmo datoteku `if-else.sh` koja će sadržavati sljedeći programski kôd odnosno retke:

```
#!/bin/bash

echo -n "Unesite broj izmedu 1 i 3 (ukljucujuci 3) : "
read broj
if [ "$broj" = "1" ]; then
    echo "Upisali ste 1."
else
    if [ "$broj" = "2" ]; then
        echo "Upisali ste 2."
    else
        if [ "$broj" = "3" ]; then
            echo "Upisali ste 3."
        else
            echo "Niste upisali broj izmedu 1 i 3 !"
        fi
    fi
fi
```

Opis važnijih redaka u našoj skripti slijedi:
`echo -n` - ispisuje poruku i ne baca nas u novi redak nego ostaje u istom retku (`-n`)
`read broj` - naredba `read` očekuje naš unos s tipkovnice (ovdje unosimo podatke) te ih ubacuje u varijablu `broj`
`if [ "$broj" = "1" ]; then` - ako je vrijednost (`$`) varijable `broj` jednaka **1** onda ide u sljedeći redak i preko `echo` naredbe ispisuje poruku.
`else` - znači: inače učini … nešto što slijedi.
`if [ "$broj" = "2" ]; then` - je uvjet koji znači da, ako je vrijednost varijable `broj` (ista provjera kao u prvom `if` uvjetu) jednaka 2, onda ide u sljedeći red i preko `echo` naredbe ispisuje poruku. …
I tako smo nanizali uvjet za uvjetom (`if` …, `else`) sve do kraja naše skripte.

Sada moramo promijeniti ovlasti kako bi naša skripta postala izvršna [*eXecutable*] te na kraju treba i pokrenuti skriptu:

```
chmod +x if-else.sh
```

Pokrenimo našu skriptu. Pokrećemo ju sa `./` jer ju pokrećemo iz trenutnog direktorija.

```
./if-else.sh
```

Skripta nam sada postavlja pitanje, a mi ćemo upisati broj `2` kako bismo vidjeli što će se dogoditi:

```
Unesite broj izmedu 1 i 3 (ukljucujuci 3) : 2
Upisali ste 2.
```

**2.** Sada napravimo *shell* skriptu koja provjerava imamo li CD/DVD-ROM na našem poslužitelju. Pod Linuxom to znači postojanje linka na posebnu datoteku `/dev/cdrom`, koja obično pokazuje na `/dev/sr0` uređaj koji je posebna vrsta datoteke (*Block device*) te označava CD/DVD-ROM uređaj. Dakle u našoj skripti provjeravamo postoji li datoteka tipa "*Block Device*": `/dev/sr0`. U slučaju da postoji ovakva datoteka, ispisujemo poruku kako postoji, a ako ne postoji ispisujemo poruku da ne postoji.

Kreirajmo datoteku (našu buduću *shell* skriptu): `if-special-file.sh` sljedećeg sadržaja:

```
#!/bin/bash

if [ -b /dev/sr0 ];
then
   echo "CD/DVD-ROM postoji"
else
   echo "CD/DVD-ROM ne postoji"
fi
```

Promijenimo joj ovlasti i pokrenimo ju te pogledajmo što će se dogoditi:

```
chmod +x if-special-file.sh
./if-special-file.sh

CD/DVD-ROM postoji
```

**3.** Ponovimo prvi primjer, ali upotrebom uvjeta: `case`. Datoteka će se zvati `uvjeti-case.sh` i sadržavati će sljedeće:

```
#!/bin/bash

echo -n "Unesite broj izmedu 1 i 3 (ukljucujuci 3) :  "
read broj
case $broj in
    1 ) echo "Upisali ste 1."
        ;;
    2 ) echo "Upisali ste 2."
        ;;
    3 ) echo "Upisali ste 3."
        ;;
    * ) echo "Niste upisali broj izmedu 1 i 3 !"
esac
```

Uočite u čemu su razlike u odnosu na prvi (**1.**) primjer!.

Promijenimo ovlasti skripti, za izvršavanje (sa `chmod`) te ju pokrenimo:

```
chmod +x uvjeti-case.sh
./uvjeti-case.sh
```

Skripta će nas zatražiti da upišemo broj.

```
Unesite broj izmedu 1 i 3 (ukljucujuci 3) :  2
```

Upisali smo **2**, pa se skripta nastavlja izvršavati te ispisuje sljedeće:

```
Upisali ste 2.
```

**Izvor informacija: (817),(K-12), `man bash`.**

# 6.6. Petlje

***Slijedi napredno poglavlje (6.6.x)!***
Petlje koristimo kada imamo potrebu ponavljanja određenog dijela programa sve dok se ne zadovolji neki od uvjeta.
Najčešće su u upotrebi tri vrste petlji:

- `for` – petlja se vrti u krug sve dok se ne završi unutarnji (definirani) brojač ili nîz (lista) definiran unutar sâme petlje.
- `while` – petlja se vrti u krug sve dok je uvjet točan.
- `until` – petlja se vrti u krug sve dok je uvjet netočan.

Za sve navedene vrste petlji moguće je koristiti i kondicionalne radnje:

- `break` – koristi se za izlazak iz trenutne petlje (obično uz neki uvjet) prije nego petlja završi.
- `continue` – koristi se za nastavljanje (iteracija) rada petlje (također obično uz neki uvjet).

Moguće je imati i petlju u petlji (tzv.. ugnježđivanje).
Ne zaboravite vašoj skripti dodijeliti izvršne ovlasti sa:

```
chmod +x IME-SKRIPTE.sh
```

prije pokretanja jer ju u suprotnom nećete moći pokrenuti.

Upotrebu petlji ćemo objasniti kroz primjere u sljedećoj cjelini.

**Izvor informacija: `man bash`**

### 6.6.1. *For* petlja

Petlja `for` koja se koristi u `bash` ljusci omogućava nam izvršavanje programskog kôda unutar petlje.
Možemo reći kako `for` petlja radi iterativno ponavljanje našeg programskog kôda unutar petlje, i to prema zadanim parametrima.

**Primjeri upotrebe:**
**1.** Kreirajmo nîz brojeva od 1 do 10, pozivajući vanjsku naredbu `seq` za kreiranje nîza brojeva.

```
#!/bin/bash

for NUM in `seq 1 1 10`
 do
  echo $NUM
 done
```


Pogledajte poglavlje upotrebe: **5.12.1.2 Upotreba navodnika ‘ “ ` (***jednostruki, dvostruki i jednostruki kosi***)** i to jednostrukih *kosih* navodnika jer se unutar njih u primjeru i poziva naredba `seq`.

Opis:
Varijabla `NUM` se postavlja na prvu vrijednost (u ovom slučaju je to `1`) jer je naredba `seq` krenula od `1`.
Potom se između `do` i `done` izvršava naš programski kôd, koji ispisuje vrijednost varijable (sa `$NUM`) s pomoću naredbe `echo`

- Naredba `seq` se koristi za kreiranje nîza brojeva (u našem slučaju od **1** do **10**, u koracima po jedan).
  Sintaksa joj je: `seq PRVI_BROJ INKREMENT ZADNJI_BROJ`

Prva iteracija se završila i ispisano je 1. Kreće druga iteracija u kojoj naredba `seq` povećava broj za 1 te se on dodjeljuje varijabli `NUM`. Sada varijabla `NUM` ima vrijednost 2. Ponovno se između `do` i `done` izvršava naš posao/kôd odnosno ispisuje se vrijednost varijable (sa `$NUM`) i to s naredbom `echo`, a sada se ispisuje 2.
U sljedećoj iteraciji se vrijednost ponovno povećava pa ispisuje, sve dok se ne dođe do limita koji je postavljen u naredbi `seq` kao 10, tada se odrađuje i ta zadnja iteracija i završava se izvođenje petlje.
Pogledajmo što dobivamo pokretanjem naše skripte s `for` petljom. Pokrenimo našu skriptu na sljedeći način:

```
./for.petlja.sh
```

Pogledajmo rezultat ispisa:

```
1
2
3
4
5
6
7
8
9
10
```

Isti rezultat smo mogli postići s upotrebom dvostrukih okruglih zagrada. Pogledajmo i kako bi izgledala ovakva skripta:

```
#!/bin/bash

for ((i=1; i<11; i++))
  do
    echo $i
  done
```

**2. Upotreba for petlje korištenjem vitičastih zagrada za generiranja nîza.**
U ovom primjeru ćemo `for` petlju ograničiti na iteracije u kojima će se “*vrtiti*” slova: `a`, `b`, `c`, `d` i `e`. U svakoj iteraciji će naša varijabla: `i` dobiti svoju vrijednost, koju kasnije unutar bloka između `do` i `done` koristimo odnosno konkretno ispisujemo. Dakle naša petlja će imati ukupno pet (5) iteracija odnosno prolazaka jer imamo pet slova; svako u svojoj iteraciji (`a`, `b`, `c`, `d` i `e`).
Pogledajmo kako će izgledati naša nova skripta:

```
#!/bin/bash

for i in {a..e}
do
    echo $i
done
```

Pokretanjem ove skripte dobivamo:

```
a
b
c
d
e
```

Isto je moguće uraditi i za generiranje brojeva, pa bi skripta za kreiranje nîza brojeva od **1** do **5** izgledala ovako:
```
#!/bin/bash

for i in {1..5}
do
    echo $i
done
```

Pokretanjem ove skripte dobivamo:
```
1
2
3
4
5
```
**3. Napravit ćemo petlju koja će koristiti nîz**. Petlja će prolaziti iterativno kroz svaki pojedini element nîza odnosno liste te pokretati naš željeni kôd. Naš nîz će biti dani u tjednu (`Pon`, `Uto`, `Sri`, `Cet`, `Pet`, `Sub` i `Ned`), koje ćemo iterativno ispisivati, jedan po jedan; jedan u svakoj iteraciji odnosno prolazu petlje. Elementi liste odnosno nîza se ne smiju odvajati zarezom jer će se on obrađivati kao još jedan element nîza odnosno liste. Pogledajmo ovakvu petlju u novoj skripti:
```
#!/bin/bash

for DANI in Pon Uto Sri Cet Pet Sub Ned
  do
    echo  $DANI
  done
```
I pogledajmo što smo dobili, njenim pokretanjem:
```
Pon
Uto
Sri
Cet
Pet
Sub
Ned
```
**4. Moguće je isti primjer odraditi i s listom elemenata postavljenim unutar druge varijable**.
Pogledajmo kako to izgleda u novoj skripti, koja će sadržavati:
```
#!/bin/bash

DANI="Pon Uto Sri Cet Pet Sub Ned"

for DAN in $DANI
  do
    echo $DAN
  done
```
Varijabli `DANI` (množina) smo dodijelili listu elemenata (`Pon` `Uto` `Sri` `Cet` `Pet` `Sub` `Ned`), prije nego smo krenuli s petljom. Potom u petlji koristimo novu varijablu `DAN` kojoj dodajemo listu elemenata jer joj na kraju dajemo vrijednost varijable koja sadrži listu (`$DANI`). Pošto unutar dijela između `do` i `done` bloka ispisujemo vrijednost varijable `DAN` sa `$DAN`, dobijemo po jedan element liste; i to onaj na kojemu se trenutno petlja i nalazi. Pogledajmo što smo dobili pokretanjem:
```
Pon
Uto
Sri
Cet
Pet
Sub
Ned
```

Ako koristite nîz/listu iz varijable, pazite na to kako se nîz odnosno lista definirana u varijabli obavezno nalazi između dvostrukih navodnika `" "`

5. **Korištenje Linux naredbe kao liste odnosno nîza.**
Koristit ćemo naredbu `awk` s kojom ćemo ispisati sve korisnike odnosno korisničke račune unutar našeg Linuxa.
Pogledajte poglavlje: **7.1.2. Datoteka /etc/passwd** za opis sadržaja navedene datoteke `/etc/passwd`.
Korištenjem programa `awk` ćemo zapravo dobiti listu korisnika, koji su zapisani u datoteci `/etc/passwd`.



Prisjetite se rada programa **`awk`**:
**5.7.3. Naredba *awk***.

Pogledajmo kako ćemo ga mi koristiti, pa pokrenimo sljedeću naredbu:

```
awk -F: '{print $1}' /etc/passwd
```

I dobit ćemo kao njen rezultat, sljedeći ispis:

```
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
uucp
operator
games
ftp
nobody
saslauth
postfix
sshd
tcpdump
apache
```

Tu istu `awk` naredbu ćemo koristiti kako bi nam kreirala listu/nîz, koju/i koristimo za našu `for` petlju unutar sljedeće skripte:

```
#!/bin/bash

for korisnik in `awk -F: '{print $1}' /etc/passwd`
do
 echo "Korisnički račun :" $korisnik
done
```

Pokretanjem gornje skripte, dobivamo sljedeće:

```
Korisnički račun : root
Korisnički račun : bin
Korisnički račun : daemon
Korisnički račun : adm
Korisnički račun : lp
Korisnički račun : sync
Korisnički račun : shutdown
Korisnički račun : halt
Korisnički račun : mail
Korisnički račun : uucp
Korisnički račun : operator
Korisnički račun : games
Korisnički račun : ftp
Korisnički račun : nobody
Korisnički račun : saslauth
Korisnički račun : postfix
Korisnički račun : sshd
Korisnički račun : tcpdump
Korisnički račun : apache
```

**Izvor informacija: (817),(K-12), `man bash`, `man awk`.**

### 6.6.2. *While* petlja

U ovoj cjelini ćemo se upoznati s `while` petljom kroz primjere upotrebe koji slijede.
**1.** Napravimo petlju ***brojač***: od **0** do **9**. Kada brojač dođe do **9** petlja treba završiti.
Stoga kreirajmo datoteku imena: `petlja-while.sh`, koja sadrži slijedeće naredbe odnosno kôd:

```
#!/bin/bash

brojac=0
while [ $brojac -lt 10 ]; do
    echo "Broj je :" $brojac
    brojac=$[$brojac + 1]
done
```

Opis:
`while` petlja ima za uvjet da varijabla `brojac` ima vrijednost manju ili jednaku 10 (`-lt 10`) tj. `while [ $brojac -lt 10 ]; do` sve dok je gore navedeni uvjet točan idi dalje: `echo "Broj je :" $brojac` ispiši poruku "`Broj je :`", sa vrijednosti (`$`) varijable `brojac`, pomoću: `brojac=$[$brojac + 1]`, s čime se brojač povećava za 1. Kraj petlje označava: `done`, a petlja se vrti u krug sve dok se ne zadovolji uvjet naveden unutar `while` bloka. Sada skripti promijenimo ovlasti kako bi se mogla pokrenuti odnosno kako bi postala izvršna:

```
chmod +x petlja-while.sh
```

Sada ju pokrenimo:

```
./petlja-while.sh
Broj je : 0
Broj je : 1
Broj je : 2
Broj je : 3
Broj je : 4
Broj je : 5
Broj je : 6
Broj je : 7
Broj je : 8
Broj je : 9
```

Vidimo da smo pokretanjem ove skripte dobili ispis kakav smo i očekivali.

**Izvor informacija: (817), (K-12), `man bash`.**

---

### 6.6.3. *Until* petlja

Pelja pomoću naredbe `until` je vrlo slična prethodnoj (ali i različita), stoga pogledajmo primjere upotrebe. Napravimo brojač odnosno petlju od **0** do **9**. Kada se dođe do **9**, petlja treba završiti.
Kreirajmo datoteku imena: `petlja-until.sh` koja sadrži cijeli potreban programski kôd:

```
#!/bin/bash

brojac=0
until [ $brojac -ge 10 ]; do
    echo "Broj je :" $brojac
    brojac=$[$brojac + 1]
done
```

Opis: Razlika u odnosu na `while` petlju je njena suprotna logika (sve dok je uvjet netočan). Uvjet je: sve dok je varijabla `brojac` veća ili jednaka 10 (`-ge 10`), a pošto `until` petlja čeka netočan uvjet, dobili smo isti rezultat kao u `while` petlji.

Promijenimo ovlasti skripti, kako bismo ju mogli pokrenuti.

```
chmod +x petlja-until.sh
```

Pokrenimo našu novu skriptu:

```
./petlja-until.sh
Broj je : 0
Broj je : 1
Broj je : 2
Broj je : 3
Broj je : 4
Broj je : 5
Broj je : 6
Broj je : 7
Broj je : 8
Broj je : 9
```

Vidimo kako je brojač uredno odradio sve što smo i očekivali (brojio je od `0` do `9`).

**Izvor informacija: (817), (K-12), `man bash`.**

### 6.6.4. *break* i *continue* unutar *for*, *while* i *until* petlji

Moguće je koristiti i dodatne uvjete unutar: `for`, `while` i `until` petlji. Za tu namjenu se može koristiti `break` koji se koristi za izlaz iz trenutne `for`, `while` ili `until` petlje prije nego bi ona normalno završila. Dakle ako je potrebno izaći iz petlje uslijed nekog uvjeta koristi se naredba `break`. S druge strane ako je potrebno preskočiti na slijedeću iteraciju unutar petlje koristi se naredba `continue`.

**Pogledajmo i primjere**.
Napraviti ćemo *shell* skriptu ***brojač***, koja će brojiti od **1** do **10**, koristeći `for` petlju:

```
#!/bin/bash

for i in {1..10}
 do
  echo $i
done

echo "Sada se nalazimo izvan petlje"
```

Ako pokrenemo brojač, dobit ćemo sljedeće:

```
1
2
3
4
5
6
7
8
9
10
"Sada se nalazimo izvan petlje"
```

Osim brojača, dodali smo i poruku koja će se ispisati kada se petlja završi: "`Sada se nalazimo izvan petlje`"

Primjer u kojemu želimo, kada se unutar petlje dođe do vrijednosti **9,** da se izađe iz petlje i nastavi dalje s izvršavanjem skripte. Za ovaj primjer ćemo koristiti `break` za izlaz ali unutar `if` uvjeta gdje se provjerava je li vrijednost brojača jednaka devet (**9**).

```
#!/bin/bash

for i in {1..10}
 do
  echo $i
    if [ "$i" -eq 9 ]
      then
        break
    fi
done
echo "Sada se nalazimo izvan petlje"
```

Ako pokrenemo skriptu, dobivamo:

```
1
2
3
4
5
6
7
8
9
"Sada se nalazimo izvan petlje"
```

Dakle petlja je došla do **9** i tada se aktivirao `if` uvjet unutar kojega se provjerava da li je vrijednost brojača (`i`) jednaka (`eq`) broju **9**. Kada je brojač došao do **9**, unutar `if` uvjeta, pokreće se naredba `break` koja iskače izvan cijele `for` petlje.

Nakon toga se nastavlja izvršavanje slijedećih naredbi koje se nalaze u skripti.
U našem slučaju se izvršava naredba: `echo "Sada se nalazimo izvan petlje"`.

Naredba `break` izlazi iz petlje, ali ne izlazi izvan cijele *shell* skripte, te će se skripta nastaviti izvršavati dalje, ako postoji još nešto izvan petlje što se mora odraditi.

## Ugniježđene petlje i `break`

Moguće je i eksplicitno navesti nakon koje razine pôd petlje želimo izaći iz nje. Dakle u slučajevima kada imamo petlju unutar petlje, moguće je izaći samo iz određene pôd petlje (obično uz neku uvjet) ili i iz vršne petlje uključujući sve njene pôd petlje. Napravit ćemo vršnu `for` petlju koja će "*vrtiti*" brojeve od **1** do **4**. Unutar ove petlje ćemo kreirati drugu `for` petlju koja će "*vrtiti*" slova od **A** do **C**. Želimo svaki put kada se u pod petlji koja vrti slova, dođe do slova **B**, da petlja završi, i to samo ova pôd petlja koja "vrti" slova.
Dakle u našem slučaju imamo glavnu petlju i njenu prvu (1) pôd petlju. Pogledajmo našu novu skriptu odnosno njen kôd:

```
#!/bin/bash

for i in {1..4}
 do
  echo $i
        for j in {A..C}
            do
               echo $j
               if [ $j == "B" ]
                  then
                      echo "Imamo slovo B"
                      break
               fi
        done
done
echo "Sada se nalazimo izvan petlje"
```

Očekujemo prvo izvršavanje prve iteracije prve petlje i rezultat: **1**, pa ulazak u drugu petlju i rezultat prve iteracije **A**, te slijedi sljedeća iteracija druge petlje i kao rezultat **B**. Sada se treba aktivirati uvjet (`if`) i provjeriti imamo li slovo **B** te se treba ispisati "`Imamo slovo B`". Zatim dolazimo do `break 1` odnosno izlaska iz prve razine `for` petlje (tj. naša pôd petlja koja generira slova) te slijedi povratak na glavnu petlju, koja nastavlja sa slijedećom iteracijom, pa dobivamo vrijednost: **2**, potom ponovno imamo ulazak u drugu petlju, koja opet kreće iz početka te dobivamo slovo: **A**, … .

U svakom slučaju je vidljivo kako sa `break` izlazimo samo iz trenutne razine petlje odnosno petlje u kojoj se nalazimo, a ne izlazimo i iz glavne petlje. Pogledajmo što dobivamo kada pokrenemo skriptu:

```
1
A
B
Imamo slovo B
2
A
B
Imamo slovo B
3
A
B
Imamo slovo B
4
A
B
Imamo slovo B
Sada se nalazimo izvan petlje
```

Ako želimo izaći i iz vršne petlje, moramo povećati razinu djelovanja `break` naredbe. U ovom slučaju bi to bila razina 2.
Promijenimo samo liniju s `break` u sljedeću:

```
break 2
```

Pokrenimo skriptu pa pogledajmo njen ispis odnosno rezultat:

```
1
A
B
Imamo slovo B
Sada se nalazimo izvan petlje
```

Čim je prva iteracija u prvoj petlji završila (rezultat : **1**) te smo ušli u prvu pod petlju, odrađuju se iteracija pod petlje.
Prva iteracija pôd petlje (rezultat : **A**) pa druga (rezultat : **B**). Potom je okidač uvjet da smo dobili slovo **B.**
Ispisuje se poruka te sa `break 2` iskačemo iz pôd petlje i iz glavne petlje (koja je s točke pod petlje, razina **2**).

## Primjeri upotrebe naredbe `continue`

Upotreba naredbe `continue` je suprotna upotrebi naredbe `break`. `continue` će nastaviti s radom petlje odnosno vratiti nas na početak petlje i to na slijedeću iteraciju na kojoj je petlja stala. Pogledajmo isti primjer s naredbom `continue`:

```
#!/bin/bash

for i in {1..4}
do
  echo $i
        for j in {A..C}
        do
             echo $j
             if [ $j == "B" ]
                 then
                     echo "Imamo slovo B"
                     continue
             fi
        done
done
echo "Sada se nalazimo izvan petlje"
```

Pokretanjem ove skripte dobivamo sljedeće:

```
1
A
B
Imamo slovo B
C
2
A
B
Imamo slovo B
C
3
A
B
Imamo slovo B
C
4
A
B
Imamo slovo B
C
Sada se nalazimo izvan petlje
```

Dakle ponašanje je slično kao kod naredbe `break` s time da kada smo došli u unutarnjoj petlji do slova **B**, tada `continue` nastavlja trenutnu petlju odnosno njenu slijedeću iteraciju; ovdje je to slovo **C**. I za `continue` je moguće koristiti skok u vršnu petlju (ili koju već razinu vršne petlje želimo).

**Izvor informacija: (817),(K-12), `man bash`.**

# 6.7. Nizanje i ulančavanje naredbi i statusni kodovi

Nizanje naredbi koristimo kada je potrebno pozivati naredbe u nîzu, jednu za drugom. Pri tome ovo nizanje ili ulančavanje naredbi zapravo znači povezivanje naredbi za izvršavanje u slijedu koji ovisi o operatorima koje smo koristili za sâmo nîzanje odnosno ulančavanje. Ovdje se ne radi sâmo o povezivanju naredbi, kao što smo to radili pomoću operatora `|` odnosno pomoću ***Pipe funkcionalnosti*** (**poglavlje. 5.10.**).

Nîzati i ulančavati naredbe možemo koristeći neke od sljedećih, takozvanih *list operatora*:

- `&` – operator koji se koristi za pokretanja programa u pozadini (primjerice kao servisa).



Pogledajte primjere upotrebe ovog operatora u poglavljima:
**9.2. Procesi i signali koje im možemo poslati.**
**9.5. Linux poslovi (*jobs*).**

- `;` – operator s kojim možemo nîzati naredbe sekvencijalno. Što znači da se završetkom izvršavanja prve naredbe u nîzu pokreće druga, nakon njenog završetka sljedeća i tako dalje.
- `||` – odnosno logički ***ILI*** (engl. *OR*) operator je koji se koristi za nîzanje naredbi, ali pri tome je važno je li prijašnja naredba uspješno završila ili nije. Naime sâmo i isključivo ako prijašnja naredba nije uspješno završila (odnosno ako ima statusni kôd različit od `0`) tek i sâmo tada se kreće na izvršavanje sljedeće naredbe u nîzu.



Pogledajte primjere upotrebe ovog operatora u poglavlju:
**7.3.1.1. *Init* sistemski servisi (*daemoni*) detaljnije.**

- `&&` – odnosno logički ***I*** (engl. *AND*), operator je za nîzanje naredbi, ali na malo drugačiji način. Samo, ako je prva naredba uspješno završila (*return status code=0*), tek i isključivo tada se kreće s izvršavanjem sljedeće naredbe u nîzu. U slučaju kad prva naredba u nîzu nije uspješno izvršena, ne nastavlja se dalje izvršavanje sljedeće naredbe u nîzu.



*Važno je razumjeti da svaka naredba nakon izvršavanja šalje svoj statusni kôd znan i kao **Exit** odnosno **Return code**.*

Statusni kôd koji je nula (`0`) označava da se sve uredno završilo. Sve druge vrijednosti ukazuju na neku vrstu greške.
Statusni kôd možemo pročitati i sa posebnom varijablom `$?` koja nam daje statusni kôd zadnje izvršene naredbe.

**Pogledajmo tablicu statusnih kôdova**:

| Statusni kôd: (broj) | Značenje | Komentar |
|---|---|---|
| 0 | Program je uredno izvršen | Ovo je normalan statusni kôd, ako je sve u redu odnosno, ako je naredba uredno izvršena i nije vratila sustavu nikakvu grešku. |
| 1 | Sve općenite greške | Primjerice dijeljenje s nulom ili druge "*nemoguće*" operacije. |
| 2 | Nepropisna upotreba ugrađenih *BASH* funkcionalnosti | Nedostaje ključna riječ ili naredba. Može biti i problem s ovlastima (*permissions*). Primjerice naredba `diff` daje ovaj kôd greške kod usporedbe binarnih datoteka. Također naznačuje da nedostaje odredišna/tražena datoteka (ili direktorij). |
| 126 | Pozvana naredba se ne može izvršiti | Problem s ovlastima (*permissions*) ili naredba nije izvršna (*execute*). |
| 127 | "*command not found*" Naredba nije pronađena | Problem sa **PATH** varijablom (putanjom) ili tipografska greška. |
| 128 | "*Invalid argument to exit*" Krivi argument za izlaz | Statusni kôd može biti samo cjelobrojna vrijednost od 0 do 255 |
| 128+n | „*Fatal error signal* **n**" Fatalna greška | Primjerice tijekom pokretanja: `kill -9 $PPID`, provjera sa `$?` vraća 137 (128 + 9) |
| 130 | Skripta je terminirana sa **Control-C** | *Control-C* pripada u "***fatal error signal 2***", (130 = 128 + 2). Pogledati gore. |
| 255* | "*Exit status*" izvan dosega | Statusni kôdovi su cjelobrojni od 0 do 255!. |

Vratimo se na nîzanje i ulančavanje naredbi. Sintaksa pri nîzanju naredbi je sljedeća:
`naredba1` ***`list-operator`*** `naredba2` ***`list-operator`*** `naredba3` …

### Slijede primjeri nîzanja i ulančavanja (povezivanja) naredbi

**1. Upotreba operatora `&&`**

**1.1** Pokrenimo dvije naredbe u nîzu, i to samo, ako je prva uspješno završila:
Ako postoji datoteka `/dev/sr0` (ovo konkretno znači da postoji CD/DVD-ROM uređaj na sustavu), ispišimo poruku "CD/DVD-ROM postoji". Stoga pokrenimo sljedeće dvije naredbe u ovakvom nizu:

```
ls -alh /dev/sr0 && echo "CD/DVD-ROM postoji"
brw-rw---- 1 root cdrom 11, 0 Jun  6 15:35 /dev/sr0

CD/DVD-ROM postoji
```

Naredba `ls -alh /dev/sr0` je vratila status `0` (OK) što znači kako tražena datoteka postoji te je sustav zatim pokrenuo slijedeću naredbu `echo "CD/DVD-ROM postoji"` i ispisao nam poruku: `CD/DVD-ROM postoji`

**1.2** U suprotnom, kada CD/DVD-ROM odnosno datoteka `/dev/sr0` ne bi postojala, dobili bi sljedeću grešku

```
ls -alh /dev/sr0 && echo "CD/DVD-ROM postoji"

ls: cannot access /dev/sr0: No such file or directory
```

Dakle u ovom slučaju druga naredba u nîzu (`echo "CD/DVD-ROM postoji"`) se nije izvršila, jer prva u nîzu ima grešku.
Pogledajmo i statusni kôd zadnje izvršene naredbe, u ovom slučaju:

```
echo $?

2
```

Dobili smo statusni kôd broj `2` dakle grešku, a koji upućuje na to kako nešto nedostaje. Ovdje je nedostajala tražena datoteka.

**2. Upotreba operatora `||`**

Upotrebom operatora `||` nîžemo odnosno ulančavamo naredbe, ali suprotnom logikom nego sa operatorom `&&`.
Probajmo ponoviti primjer iz točke **1.2**, ali upotrebom operatora `||` te malom izmjenom poruke u drugoj naredbi u nîzu:

```
ls -alh /dev/sr0 || echo "CD/DVD-ROM NE postoji"
ls: cannot access '/dev/sr1': No such file or directory
CD/DVD-ROM NE postoji
```

Pošto u ovom primjeru datoteka `/dev/sr0` ne postoji, sustav nam vraća statusni kôd broj dva (`2`), a koristili smo operator `||` koji nalaže izvršavanje slijedeće naredbe u nîzu <u>samo i isključivo ako</u> je statusni kôd različit od nule (`0`), što smo ovdje zadovoljili. Stoga, jer smo imali grešku nakon izvršavanja prve naredbe, sustav pokreće slijedeću naredbu u nîzu, pa dobivamo poruku: `CD/DVD-ROM NE postoji`. Dakle ponašanje operatora `||` je po logici rada suprotno od ponašanja operatora `&&`.

**3. Upotreba operatora `;`**

Upotrebom ovog operatora možemo nîzati naredbe, kao da ih izvršavamo u ljusci (*shellu)*, jednu za drugom, neovisno o njihovom uspješnom ili neuspješnom završetku, kao što je bio slučaj s operatorima `||` i `&&`.
Pokrenut ćemo naredbu `cd /root/` s kojom ćemo ući u direktorij `/root`, a potom `ls -al` da ispišemo sadržaj tog direktorija, sve u jednom retku, gdje će se obje naredbe izvršiti jedna za drugom, bez obzira kakav je status prethodne naredbe:

```
cd /root/ ; ls -al
```

**Izvor informacija: (817),(K-12), `man bash`.**

# 6.8. Funkcije

***Slijedi napredno poglavlje!***
Kao i većina programskih jezika i *bash* ljuska može koristiti funkcije. Funkcije implementiraju određene radnje ili operacije koje ćemo potencijalno koristiti više puta unutar programa odnosno skripte. Korištenjem funkcija činimo programski kôd:

- Čitljivijim: kod takvog rada svaka funkcija bi trebala odrađivati određeni zadatak.
- Iskoristivijim: programski kôd se ne bi trebao ponavljati, već pozivati funkciju.

Funkcije možemo koristiti na dva načina, sintaksa je pri tome slijedeća:

```
function  ime_funkcije {

naredbe
naredbe
}
```

ili

```
function  ime_funkcije () {

naredbe
naredbe
}
```

Osim toga, ne moramo niti koristiti ključnu riječ `function`, pa će to izgledati ovako:
```
ime_funkcije {

    naredbe
    naredbe

}
```
ili
```
ime_funkcije () {

    naredbe
    naredbe

}
```
Dodatno, možemo i ugnježđivati funkciju unutar funkcije.

**Primjeri**
1. Napravimo skriptu koja će imati dvije funkcije: jednu za zbrajanje dva broja i drugu za množenje dva broja. Skripta mora tražiti unos od korisnika: prvi broj, a potom drugi broj, te naziv matematičke operacije koju se želi izvršiti: zbrajanje ili množenje. Za ovu potrebu kreirajmo datoteku `funkcije1.sh` sljedećeg sadržaja:
```
#!/bin/bash

echo "Upisi prvi broj pa drugi broj te operaciju : zbrajanje ili mnozenje:"
read broj1 broj2 operacija

function zbrajanje () {
         echo $(($broj1+$broj2))
}

function mnozenje () {
         echo $(($broj1*$broj2))
}

case $operacija in
    zbrajanje ) zbrajanje ;;
    mnozenje ) mnozenje ;;
    * ) echo "Niste upisali : zbrajanje ili mnozenje"
esac
```
Pogledajmo opis redova programskog kôda koji smo koristili (samo trenutno zanimljive dijelove):
`echo "Upisi prvi broj…` → ispiše ovaj tekst unutar navodnika.
`read broj1 broj2 operacija` → očekujemo unos od korisnika, koji će se zapisati u tri varijable:
- `broj1` → ovo će biti naš prvi broj.
- `broj2` → ovo će biti naš drugi broj.
- `operacija` → ovo će biti matematička operacija (`zbrajanje` ili `mnozenje`).
```
function zbrajanje () {
         echo $(($broj1+$broj2))
}
```
- ✓ Ovdje kreiramo funkciju imena `zbrajanje` koja preko `echo` naredbe (u drugom redu) zbraja vrijednosti varijabli: `broj1` i `broj2` te ispisuje rezultat.

Isto se ponavlja i kod funkcije `mnozenje`, s time da ona množi ta dva unesena broja i ispisuje rezultat množenja:
```
case $operacija in
```
Sada pozivamo uvjet `case` koji provjerava što smo upisali kao treću vrijednost (prve dvije su bili brojevi), a za treću očekujemo da bude opisna matematička operacija: `zbrajanje` ili `mnozenje`.

Objasnit ćemo redak po redak:
`zbrajanje ) zbrajanje ;;` → ovdje provjeravamo je li upisano `zbrajanje` (prva riječ u ovom primjeru) i ako je, pozivamo funkciju `zbrajanje` (druga riječ u ovom primjeru).
`mnozenje ) mnozenje ;;` → ovdje provjeravamo je li upisano `mnozenje` i ako je, tada pozivamo funkciju `mnozenje`
`* ) echo "Niste upisali : zbrajanje ili mnozenje"`. Ako nisu zadovoljena prva dva uvjeta, tada ispisujemo poruku `Niste upisali : zbrajanje ili mnozenje`
Sada promijenimo ovlasti skripti i pokrenimo ju. Prvo joj promijenimo ovlasti, da postane izvršna:
```
chmod +x funkcije1.sh
```

Potom pokrenimo skriptu te unesimo brojeve `2` i `4` te operaciju `zbrajanje`
```
./funkcije1.sh
Upisi prvi broj pa drugi broj te operaciju : zbrajanje ili mnozenje:
2 4 zbrajanje
6
```
Pošto smo unijeli brojeve `2` i `4` te `zbrajanje`, pa je pozvana funkcija koja je zbrojila dva broja i ispisala kao rezultat: `6`.

**Izvor informacija: (817), `man bash`.**

# 6.9. Lokalne varijable

Moguće je određene varijable definirati kao lokalne; primjerice unutar naše funkcije.
Pogledajmo kako možemo definirati lokalne varijable uz primjer sličan prethodnom.



Varijablu definiramo lokalnom pozivanjem ključne riječi: `local` ispred imena varijable

Sada pogledajmo primjer ovakve skripte. Kreirajmo novu skriptnu datoteku imena: `funkcije2.sh` iz programa `vi`:

```
vi funkcije2.sh
```

I u nju ćemo ubaciti sljedeći kôd:

```
#!/bin/bash

x=100
y=200

zbrajanje(){
  local x=$1
  local y=$2
  echo "Rezultat je :" $(( $x + $y ))
}
echo "Pozovimo globalnu varijablu x: $x"
echo "Pozovimo globalnu varijablu y: $y"
echo "Pozovimo funkciju za zbrajanje uz varijable koje smo unjeli (lokalne
varijable):"
zbrajanje $1 $2
```

**Opis:** Na početku smo definirali globalne varijable `x` i `y` sa sljedećim programskim kôdom:

```
x=100
y=200
```

Nakon toga kreiramo funkciju koja će zbrajati dva broja.

Unutar te funkcije smo definirali lokalne varijable, čija vrijednost traje samo unutar ove funkcije:

```
local x=$1
local y=$2
```

Podsjetimo se: `$1` i `$2` su sistemske varijable koje označavaju prvi i drugi argument koji upisujemo; primjerice uz pokretanje ove skripte. Potom ispisujemo rezultat:

```
echo "Rezultat je :" $(( $x + $y ))
```

Izvan naše funkcije ispisujemo globalne varijable:

```
echo "Pozovimo globalnu varijablu x: $x"
echo "Pozovimo globalnu varijablu y: $y"
```

Na kraju pozivamo našu funkciju, kojoj prosljeđujemo argumente s kojima ćemo pokrenuti našu skriptu (`$1` i `$2`).
Dakle vrijednosti `$1` i `$2` se prosljeđuju unutar naše funkcije koju pozivamo (pokrećemo):

```
zbrajanje $1 $2
```

Promijenimo ovlasti našoj skripti, kako bismo ju mogli pokrenuti (izvršiti):

```
chmod +x funkcije2.sh
```

Sada pokrenimo našu skriptu i odmah joj proslijedimo argumente, koji su u našem slučaju brojevi `2` i `4`.

```
./funkcije2.sh 2 4

Pozovimo globalnu varijablu x: 100
Pozovimo globalnu varijablu y: 200
Pozovimo funkciju za zbrajanje uz varijable koje smo mi unjeli (lokalne varijable
unutar funkcije):
Rezultat je : 6
```

Vidimo kako smo dobili željeni rezultat (`6`).

**Izvor informacija: (817), (K-12), `man bash`.**

# 6.10. Povezivanje više *shell* skripti međusobno

***Slijedi napredno poglavlje!***
Vrlo često postoji potreba da povežemo više *shell* skripti međusobno tako da iz jedne skripte pozivamo funkcije ili varijable koje smo postavili odnosno definirali u drugoj skripti. Slično kao što u programskom jeziku ***C*** s naredbom: `#include` nalažemo *predprocesoru* da uključi biblioteke funkcija sadržane u nekoj drugoj datoteci, u izvorni kôd naše datoteke. Važno je razumjeti da u *bash* ljusci (*shellu*) ovakvim povezivanjem ne uključujemo izvorni programski kôd druge *shell* skripte, već sâmo učitavamo varijable i funkcije koje su definirane odnosno postavljene u njoj te izvršavamo naredbe iz te skripte.
Naime uvijek se teži tome da svaka *shell* skripta odrađuje određenu zadaću, funkcionalnost ili dio funkcionalnosti, zbog lakšeg razvoja i analize rada samih skripti. Jedan od primjera ovakvog rada su *shell* skripte koje koristi sustav za inicijalizaciju mrežnih sučelja odnosno mrežnih kartica.
Za ovu primjenu, ovdje imamo vršni direktorij `/etc/sysconfig/network-scripts/` unutar kojeg imamo primjerice datoteku `network-functions` u kojoj su definirane mnoge varijable i funkcije pomoću koji se postavljaju IP(v4) parametri, poput: informacija o mrežnom sučelju, postavljanja imena računala (*hostname*), dodavanje ruta i slično. Zatim imamo druge *shell* skripte koje su specifične za vrstu mrežnog sučelja, te one koje su zadužene za dodavanje statičkih ruta i tako dalje.

Dakle svaka pojedina *shell* skripta je napravljena za svoju namjenu, a pri tome one pozivaju funkcije ili čitaju varijable; jedne iz drugih, prema potrebi. ***Važno je znati da shell skripte koje služe sâmo kao funkcijske skripte, ne moraju biti izvršne odnosno mogu biti bez postavljene „X“*** (eXecute) ***ovlasti.***

Napravimo jednu funkcijsku skriptu imena `moje_funkcije.sh` koja će sadržavati sljedeći programski kôd:

```
#!/bin/bash
VARIJABLA1=proba
tko_sam_ja(){
   id -un
   }
```

Dakle u ovoj (funkcijskoj) datoteci smo napravili sljedeće:

- Postavili smo vrijednost varijable: `VARIJABLA1` na `proba`.
- Potom smo definirali funkciju imena: `tko_sam_ja` da pokreće naredbu `id -un` koja nam daje ime trenutno prijavljenog (*logiranog*) korisnika na sustavu.

Sada ćemo kreirati novu (glavnu) *shell* skriptu iz koje ćemo pomoću naredbe `source` pozvati našu funkcijsku *shell* skriptu.
Stoga kreirajmo našu novu *shell* skriptu imena `skripta.sh` koja će sadržavati sljedeći kôd odnosno retke:

```
#!/bin/bash
source moje_funkcije.sh
echo "Vrijednost varijable iz funkcijske datoteke je:" $VARIJABLA1
echo "Ja sam korisnik:"
tko_sam_ja
```

Zatim ćemo ovoj novoj skripti dodati ovlasti izvršavanja (***X***), kako bismo ju mogli pokrenuti. To ćemo napraviti sa:

```
chmod a+x skripta.sh
```

I potom ju pokrenimo, pa ćemo vidjeti što ćemo dobiti:

```
./skripta.sh

Vrijednost varijable iz funkcijske datoteke je: proba
Ja sam korisnik:
root
```

Vidimo da smo u prvom redu ispisali vrijednost varijable (`proba`) postavljene u funkcijskoj datoteci `moje_funkcije.sh`. Nadalje, na kraju smo i pozvali funkciju `tko_sam_ja` pomoću koje smo pozvali naredbu `id -un`, a koja nam je ispisala da smo mi (koji smo pokrenuli ovu skriptu), korisnik imena `root`.

Osim pozivanja funkcijskih ili drugih *shell* skripti unutar svoje skripte, moguće je pozvati iste skripte i direktno iz ljuske.
Pa to i napravimo; odnosno pozovimo našu funkcijsku skriptu imena `moje_funkcije.sh` i to direktno iz ljuske:

```
source moje_funkcije.sh
```

Umjesto pozivanja naše funkcijske skripte imena `moje_funkcije.sh` pomoću naredbe `source`, to je moguće izvesti i na brži način (za ***bash*** ljusku):

```
. moje_funkcije.sh
```

Dakle upisali smo `.` te razmak i tek potom funkcijsku skriptu i na taj način ju učitali isto kao što bi ju učitali s naredbom `source`. Sada kada smo učitali funkcijsku skriptu, možemo pozivati njene funkcije ili čitati vrijednosti varijabli iz nje (iz naše funkcijske datoteke).

Da bi to dokazali pozvat ćemo funkciju `tko_sam_ja` koja je definirana samo u njoj:

```
tko_sam_ja
root
```

Vidimo da smo ju ispravno pozvali i da je ona odradila što je trebala; konkretno je pokrenula naredbu: **`id -un`**.



Naredba `source`, dolazi unutar *bash* ljuske odnosno ugrađena je u nju (engl. *Shell builtin*).

**Ako za neku naredbu želimo saznati je li ugrađena u ljusku ili dolazi kao zaseban program, to možemo provjeriti upotrebom naredbe `type`. Pogledajmo primjer provjere za naredbu `source`.**

```
type source

source is a shell builtin
```

Za konkretnu naredbu smo dobili odgovor, kako je ona ugrađena u naredbeni redak odnosno ljusku (*shell*).

**Izvori informacija: (753),(754),(755),(817), `man bash`, `info source`, `help source`.**

# 6.11. Tijek izvršavanja skripti i naredba `set`

Naredba `set` koristi se za postavljanje ili poništavanje određenih zastavica i postavki koje određuju ponašanje skripti i pomažu u izvršavanju zadataka bez ikakvih problema, unutar naredbene ljuske ili skripte. Ona se može koristiti i za promjenu ili prikaz atributa i parametara rada ljuske. Ova naredba već je ugrađena u ***bash*** ljusku, ali i druge ljuske (***sh***, ***csh***, ***ksh*** i neke druge).
Ako ju pokrenemo bez opcija ili prekidača, prvo ćemo dobiti ispis svih varijabli sustava, a potom i sve druge postavke i funkcije sustava (ispis ovisi o inačici ***bash*** ljuske).
S njom možemo definirati i sistemske varijable; primjerice povećati *History* na pamćenje 2.000 zadnjih izvršenih naredbi:

```
set HISTSIZE=2000
```

Međutim mi ćemo se sada fokusirati na njene mogućnosti koje se odnose na rad unutar ***shell*** skripti.
Pogledajmo logički jednu ***shell*** skriptu:

```
#!/bin/bash
Naredba1
Naredba2
export NOVA=$PTH
Naredba3 | grep
```

**Greške u izvršavanju naredbi**
U slučaju kada se naredba (`Naredba1`) nije izvršila kako treba, skripta bi nastavila dalje s izvršavanjem slijedeće naredbe (`Naredba2`) i ostatkom skripte. Međutim tu nam može pomoći naredba `set` tako da ju dodamo u drugi redak, ovako:

```
#!/bin/bash
set -e
Naredba 1
. . .
```

Naime s opcijom `-e`, nalažemo ***bash*** ljusci, da ukoliko skripta naiđe na grešku, da se njeno dalje izvršavanje zaustavi.

**Druga mogućnost su greške prilikom pozivanja varijabli koje ne postoje**
Ako primjerice koristimo nepostojeću varijablu (u primjeru: `$PTH` ne postoji) dobit ćemo grešku, ali će skripta nastaviti dalje.
I ovdje je moguće djelovati (nova opcija `-u`) na način da također u drugi redak skripte dodamo sljedeće:

```
set -eu
```

Sada ćemo u ovakvom slučaju dobiti grešku „***unbound variable***“ i skripta se neće nastaviti izvršavati.

**Treća mogućnost je detekcija problema kod upotrebe** *pipea*
U slučaju da ako primjerice u našoj gornjoj skripti naredba `Naredba3` ne postoji, a mi pozivamo: `Naredba3 | grep` mi bi tada dobili novu grešku, koja će zaustaviti dalje izvršavanje skripte, samo ako koristimo:

```
set -euo pipefail
```

U slučaju da nismo ovo koristili, skripta bi se uobičajeno nastavila izvršavati, bez obzira na prvotnu grešku.

Postoje i mnoge druge korisne opcije, a jedna od njih je `-x` koju možemo koristiti, ako želimo vidjeti sve detalje prilikom pokretanja skripte to jest tijekom pozivanja svake naredbe te njenih argumenata i opcija unutar skripte.
Za ovakav slučaj (obično za debugiranje) možemo u drugi redak skripte dodati sljedeće:

```
set -x
```

I sada ćemo u ispisu vidjeti detaljan ispis o tijeku izvršavanja naše skripte; redak po redak, kako se izvršava.

**Izvori informacija: (753),(754),(755),(817),(1199),(1200), `man bash`, `help set`.**

# 7. Administracija Linux sustava

Za potpunu administraciju Linux sustava sa svim pravima, potrebno je biti korisnik imena: `root`, koji je u UNIX i Linux operativnim sustavima ekvivalent administratoru u *Windows* operacijskim sustavima. Pri tome `root` korisnik ima ***User ID*** (***UID***) odnosno identifikacijski broj korisnika nula (***0***). I drugi korisnici mogu dobiti *root* ovlasti (ako znaju *root* lozinku), upotrebom naredbe `su` (engl. *Substitute User*) ili naredbe `sudo`, i to samo ako im je to dozvoljeno od strane administratora.

# 7.1. Rad s korisničkima računima, grupama i lozinkama

Pošto je Unix/Linux svijet višekorisnički, posebnu pažnju treba posvetiti korisničkim računima te korisničkim grupama na sâmom sustavu. Dakle u svakom trenutku, moguće je da više korisnika koristi sustav u isto vrijeme. U primjerima oko ovlasti (poglavlje: ***4.3.***) vidljiva je veza između korisničkih računa i korisničkih grupa. Naime da bi se uopće mogli prijaviti na sustav (računalo) odnosno logirati, prvo nam je potreban korisnički račun (engl. *account*) s pripadajućim korisničkim imenom koje nosi određene ovlasti i prava pristupa. Svaki korisnik uz sâmo korisničko ime (engl. *username*) na razini sustava, identificira se jedinstvenim korisničkim brojem zvanim ***UID*** (engl. *User ID*). Isto vrijedi i za svaku korisničku grupu, koja ima jedinstveni identifikacijski broj grupe odnosno ***GID*** broj (engl. *Group ID*). Korisnik osim svog korisničkog imena mora pripadati i minimalno jednoj (primarnoj) korisničkoj grupi. Kako smo već rekli, u UNIX/Linux svijetu `root` korisnik `ID:0` je ekvivalent *administratoru*. On također pripada primarnoj grupi imena `root` koja ima broj korisničke grupe **0** (`GID:0`).



Kod kreiranja i rada sa svakim novim korisničkim računom, inicijalno se čitaju konfiguracijske datoteke s definicijom parametara rada te osnovnih postavki, definiranim u poglavlju: **7.1.6. Sigurnosne postavke i ograničenja sustava.**

Postanimo ***root*** korisnik i preuzmimo njegove postavke pomoću naredbe `su` (engl. *Substitute user*) te pomoću prekidača `-` s kojim ćemo pokupiti i sve postavke njegovog okruženja (varijable i sl.). To ćemo napraviti na sljedeći način:

```
su - root
```

Svaki korisnički račun (engl. *account*) je upisan u posebnu datoteku: `/etc/passwd`, a njegova lozinka (*hash*) je zapisana u datoteku: `/etc/shadow`. Sve korisničke grupe koje postoje na sustavu, zajedno s pripadnostima: *korisnik-grupa* se nalaze zapisani u datoteci: `/etc/group`. Pogledajmo što se nalazi u kojoj od navedenih sistemskih datoteka, malo detaljnije:

- `/etc/passwd` - sadrži ime korisničkog računa (*accounta*), osobne podatke (ime, prezime, …) te ljusku (*shell)* koja će biti pokrenuta tijekom prijavljivanja (*logiranja*) korisnika na sustav, kao i druge informacije (o njima kasnije).
- `/etc/shadow` - sadrži *hash* vrijednost lozinke za svaki korisnički račun.
- `/etc/gshadow` - sadrži *hash* vrijednost lozinke za svaku korisničku grupu (ako je ova lozinka uopće definirana).
- `/etc/group` - ovdje su definirane sve korisničke grupe i pripadnosti grupama, za svakog pojedinog korisnika.

O ovim datotekama nešto kasnije, tek kada naučimo kako se radi s korisnicima i korisničkim grupama pomoću naredbi za tu namjenu.



**Sve promjene vezane za korisničke račune ili korisničke grupe se zapisuju u datoteku:** `/var/log/secure`.

## 7.1.1. Osnovne naredbe za rad s korisničkim računima i grupama

U ovoj cjelini prvo ćemo naučiti kako kreirati korisničke račune i korisničke grupe te kako ih mijenjati.
Lista osnovnih naredbi za rad s korisničkim računima i korisničkim grupama je slijedeća (uz kratki opis svake naredbe):

- `id` – ispiše tko je trenutni logirani korisnik, koji je njegov *User ID* (`UID`) i *Group ID* (`GID`) i kojim grupama pripada.
- `lslogins` – ispiše sve korisnike i podatke o njima.
- `gpasswd` – koristi se za rad s korisničkim grupama i lozinkom za korisničke grupe
- `groups` – ispiše kojim korisničkim grupama pripada trenutni korisnik.
- `groupadd` – dodaje/kreira novu korisničku grupu.
- `groupdel` – obriše korisničku grupu.
- `groupmod` – za modifikacije nad korisničkim grupama.
- `newgrp` – za privremeno pridruživanje u željenu (primarnu) korisničku grupu.
- `useradd` – dodavanje novog korisnika. Za povezivanje sa ***SELinux*** korisnikom pogledajte poglavlje: **28.2**.
- `userdel` – brisanje postojećeg korisnika.
- `usermod` – za modifikacije nad korisničkim računom.
- `passwd` – za promjenu lozinke korisnika odnosno korisničkog računa.
- `chpasswd` – za promjene lozinki više korisnika istovremeno: ručno ili iz neke skripte.

Navedene naredbe za nas mijenjaju već spomenute konfiguracijske datoteke: `/etc/passwd`, `/etc/shadow` i `/etc/group` u koje i zapisuju sve potrebne informacije. Kroz primjere ćemo vidjeti konkretnu primjenu navedenih naredbi.

Sve navedene naredbe dolaze u softverskom paketu imena **`shadow-utils`**.

Prvo ispišimo tko je trenutni korisnik s kojim smo se prijavili na sustav (logirali), pomoću naredbe `id` i to na sljedeći način:

```
id
uid=0(root) gid=0(root) groups=0(root)
```

**Opis:** `uid 0` znači da je korisnički ***ID*** jednak `0`, a taj *uid* `0` označava kako je to *root* korisnik (*administrator*). Primarna grupa ovog korisnika je `0`, a ime te grupe je `root`. Ovaj korisnik pripada sâmo istoj sekundarnoj grupi imena: `root` (***gid* 0**).



***Ime korisnika i njegova primarna grupa se zapisuju tijekom kreiranja datoteka ili direktorija (mapa) u polje s ovlastima (*permissionima*). Uz korisnički ID (UID), primarna grupa (GID) se koristi kao korisnička grupa kod pristupanja i/ili promjena na datotečnom sustavu.***

## Osnovni rad s korisnicima i korisničkim grupama

**1.** Ispišimo kojim sve grupama pripada trenutni korisnik `root`, pomoću naredbe `groups`:

```
groups

root
```

← Vidimo da korisnik `root` pripada samo korisničkoj grupi imena `root`.

**2.** Ispišimo sve aktivne korisnike na sustavu, pomoću naredbe `lslogins` (skratili smo ispis):

```
lslogins

UID USER      PWD-LOCK PWD-DENY  LAST-LOGIN GECOS
  0 root             0        0    12:04:11 root
  1 bin              0        1             bin
  2 daemon           0        1             daemon
  3 adm              0        1             adm
  5 sync             0        1             sync
  6 shutdown         0        1 2020-Oct23  shutdown
48 apache            0        1             Apache
```

**2.1.** Ispišimo sve podatke o korisniku `root` također pomoću naredbe `lslogins`, ali sada na sljedeći način:

```
lslogins root

Username:                           root
UID:                                0
Gecos field:                        root
Home directory:                     /root
Shell:                              /bin/bash
No login:                           no
Password is locked:                 no
Primary group:                      root
GID:                                0
Last login:                         12:04:11
Last terminal:                      pts/0
Last hostname:                      192.168.1.131
Password expiration warn interval:  7
Password changed:                   Feb03/01:00
Maximal change time:                99999
Running processes:                  513
Last logs: 08:28:41 systemd[1]: Started Session 688204 of user root.
```

Iz ovog ispisa vidimo polja koja definiraju:

- Koje je njegovo korisničko ime – polje: `Username:` te početni/radni (*home*) direktorij – polje: `Home directory:`
- Ljuska u koju će se korisnik logirati je vidljiva u polju: `Shell:`, a ovdje je to `/bin/bash` ljuska.
- Primarna grupa kojoj pripada korisnik je vidljiva u polju: `Primary group:`, a ovdje je to grupa `root`.
- Je li korisniku korisnički račun zaključan; što je vidljivo u polju: `Password is locked:`
- Kada je korisnik zadnji puta logiran, što je vidljivo u polju: `Last login:`. Vidljivo je i s koje IP adrese se zadnji puta korisnik (udaljeno) spajao na sustav, kako je vidljivo u polju: `Last hostname:`

Sve što je vidljivo iz prethodnog ispisa za traženog korisnika su zapravo informacije prikupljene iz datoteka: `/etc/passwd`, `/etc/shadow` i `/etc/group`.

**3.** U slučaju kada kao `root` (*admin*istrator) korisnik želimo promijeniti lozinku za korisnika `pero` to možemo napraviti sa:

```
passwd pero
```

Ako želimo promijeniti lozinku sâmi za sebe (kao logirani korisnik), to možemo napraviti samo pozivanjem naredbe `passwd` nakon koje će nas sustav prvo tražiti staru lozinku, a potom i novu, koju moramo još jednom potvrditi.



Imamo li potrebu resetiranja izgubljene lozinke za **root** korisnika (administratora), pogledajte poglavlje:
**11.1.1.5.1. Rad s GRUB-om prilikom pokretanja sustava (*rescue*).**

## Rad s korisničkim računima (kreiranje, modifikacije i brisanje)

Prilikom kreiranja korisničkog računa na sustavu, dodjeljuje mu se jedinstveni identifilator koji se zove *User ID* (*UID*).
Linux, kao i većina UNIX-a prilikom kreiranja korisničkih računa koristi određene ***UID*** brojeve odnosno njihove raspone za određene svrhe. Pogledajmo ih:

- `UID 0` – *UID 0* korisničkog računa administratora (tzv. root). Root korisnik ima apsolutnu kontrolu nad sustavom i može izvoditi privilegirane radnje poput instaliranja softvera, upravljanja korisnicima i pristupanja važnim datotekama sustava.
- `UID 1-200` – *UID-i* korisničkih računa s posebnim sistemskim privilegijama, primjerice to su: ***daemon***. ***lp***, ***sync***, ***mail***,....
- `UID 201-999` – UID-i dodijeljeni posebnim procesima sustava, uobičajeno su to razni servisi. Primjerice u ovom opsegu su: ***sssd*** (za *System Security Services Daemon*), ***polkit*** (*PolicyKit* servis/daemon), odnosno sistemski korisnici koji pokreću servise poput web poslužitelja, baza podataka i slično.
- `UID 1000` – raspon *UID-a* za dodjelu normalnim, neprivilegiranim korisnicima.

Dakle prilikom kreiranja novog običnog korisnika, njegov *UID* će biti prvi slobodan broj od **1000**. Dok će primjerice prilikom kreiranja posebnih korisničkih računa, recimo za *Apache web poslužitelj*, njegov *UID* biti prvi slobodan broj iz opsega od **201** do **999**.

*User ID* (**UID**) brojevi odnosno njihov opseg je definiran u datoteci: `/etc/login.defs`.U ovoj datoteci su definirani *UID* brojevi za korisničke račune prilikom kreiranja korisnika u dvije kategorije: za sistemske servise i za sve ostale (normalne) korisnike, kako smo već spomenuli:

- `SYS_UID_MIN   201` → minimalni *UID* broj za sistemske korisničke račune.
- `SYS_UID_MAX    999` → maksimalni *UID* broj za sistemske korisničke račune.
- `UID_MIN   1000` → minimalni *UID* broj za normalne korisničke račune
- `UID_MAX  60000` → maksimalni *UID* broj za normalne korisničke račune

**4.** Kreirajmo korisnika `pero` koji će dobiti ***UID*** (*User* ID) `100` i pripadati će primarnoj grupi koja ima ***GID*** (*Group ID*) `1001`. *Home* direktorij će mu biti `/home/pero` te će mu pokrenuta ljuska biti `bash`.
U ovom primjeru *home* direktorij će biti automatski kreiran od strane operativnog sustava. Početni odnosno *home* direktoriji (mape) korisnika se obično nalaze unutar vršnog direktorija `/home/`.
Sada kreirajmo navedenog korisnika `pero` s naredbom `useradd` na sljedeći način:

```
useradd pero -u 100 -g 1001 -d /home/pero -s /bin/bash
```

Pogledajmo sumirano neke od korisnih prekidača naredbe `useradd`:

- `-d` *DIREKTORIJ* – kreiraj direktorij (s putanjom) za novi korisnički račun; pr. `/home/pero/`.
- `-u` *UID* – kreiraj korisnički račun koji će imati točno definirani *UID* broj; pr. `1024`.
- `-g` *GRUPA* – kreiraj korisnički račun kojem će primarna grupa biti [*GRUPA*]; pr. `korisnici`.
- `-G` *GRUPA* – kreiraj korisnički račun kojem će sekundarna grupa biti [*GRUPA*]; pr. `podaci`.
- `-s` *SHELL* – kreiraj korisnički račun koji će koristiti ljusku [*SHELL*]; pr. `/bin/bash`.

**Dodavanje korisnika u sekundarnu grupu**
**5.** Dodajmo istog korisnika (`pero`) i u dodatnu grupu (`-a`) `korisnici2`, ali ne mijenjajući mu primarnu grupu (`- G`):

```
usermod -a -G korisnici2 pero
```

**Promjena pripadnosti primarnoj grupi**
**6.** Korisniku `pero` promijenimo pripadnost **primarnoj** (`-g`) grupi u grupu: `korisnici2` na sljedeći način:

```
usermod -g korisnici2 pero
```



Vezano za detalje oko korisničkih računa i grupa, pogledajte poglavlja:
**7.1.2. Datoteka /etc/passwd →** za korisničke račune**.**
**7.1.4. Datoteka /etc/group →** za korisničke grupe**.**
**7.1.6.2. Datoteka /etc/login.defs.**

**Brisanje korisničkog računa**

**7.** U slučaju kada nam više ne treba primjerice korisnik `pero` možemo ga obrisati.
Ako želimo možemo pri tome obrisati i njegov *home direktorij* i sve datoteke u njemu (`-r`), čak iako je već logiran (`-f`).
To možemo postići na sljedeći način:

```
userdel -f -r pero
```



Korisnički račun možemo zaključati pomoću naredbe `passwd`, ako uz ime korisnika koristimo prekidač **`-l`** odnosno otključati, koristimo li prekidač **`-u`**. Zaključavanje korisničkog računa se preporučuje umjesto brisanja računa.

Preporuka je korisničke račune ne brisati već zaključati, ako više nisu u upotrebi. To možemo napraviti sa:

```
passwd -l pero
```

Ili sa sljedećom naredbom

```
usermod -L pero
```



Ako brišemo korisnika koji je vlasnik određenih datoteka, sâme datoteke čijeg vlasnika brišemo neće se automatski obrisati. Međutim, datoteke će biti bez imena vlasnika odnosno na pozicij ovlasti za vlasnika (engl. owner) će ostati samo **UID** broj korisnika koji više ne postoji na sustavu. Stoga postoji potencijalan sigurnosni problem: ako kreiramo novog korisnika, koji će potencijalno dobiti isti **UID** broj, jer će on imati sva prava bivšeg korisnika koji je bio vlasnik tih datoteka.
Stoga je najbolje ne brisati korisničke račune, već ih zaključavati.

## Rad s korisničkim grupama

Svaki korisnik mora pripadati minimalno jednoj korisničkoj grupi, koja se naziva primarna grupa, a uz svoj naziv ima i svoj identifikaciski broj odnosno *Group ID* (*GID*). I sve ostale korisničke grupe uz ime (naziv) grupe moraju imati pripadajući *GID* broj. *Group ID* (***GID***) brojevi odnosno njihov opseg koji se koristi prilikom kreiranja grupa je definiran u datoteci: `/etc/login.defs`. U ovoj datoteci su definirani *GID* brojevi za korisničke grupe prilikom njihovog kreiranja.
Vidljive su kategorija za sistemske servise te kategorija za sve ostale (normalne) korisničke grupe:

- `SYS_GID_MIN   201` → minimalni *GID* broj za sistemske servise.
- `SYS_GID_MAX    999` → maksimalni GID broj za sistemske servise.
- `GID_MIN   1000` → minimalni GID broj za normalne korisničke grupe
- `GID_MAX   60000` → maksimalni GID broj za normalne korisničke grupe.

To znači da primjerice, ako kreiramo normalnu korisničku grupu, da će njen *GID* broj biti prvi slobodan broj od **1000** na dalje.
Odnosno ako se kreira korinička grupa za neki sistemski servis, da će ona dobiti prvi slobodan broj iz opsega od **201** do **999**.

**8.** Kreirajmo nove korisničke grupe: `korisnici`, `korisnici2` i `novi` za koje želimo da redom imaju identifikatore grupa (***GID***) brojeve, kako slijedi: `1001` i `1002` te `1003`. To postižemo s naredbom `groupadd` na sljedeći način:

```
groupadd -g 1001 korisnici
groupadd -g 1002 korisnici2
groupadd -g 1003 novi
```

**9.** Želimo promijeniti ***GID*** (*Group ID*) grupe broj: `1003` (grupa: `novi`) u ***GID:*** `2003`. Ovo se radi s naredbom `groupmod`

```
groupmod -g 2003 novi
```

**10.** Odlučili smo grupu `novi` preimenovati u `novikorisnici` što ćemo isto napraviti s naredbom `groupmod`

```
groupmod -n novikorisnici novi
```

**11.** Ako ipak želimo obrisati grupu `novikorisnici` jer nam više ne treba, to ćemo postići s naredbom `groupdel`

```
groupdel novikorisnici
```



***Ako brišemo korisničku grupu koja je primarna grupa barem jednom korisniku, sustav nam to neće dozvoliti sve dok ne očistimo (popravimo) ovu ovisnost!***
Obratite pažnju na datoteke i direktorije kojima je to bila vlasnička grupa, jer će ona tada ostati zapisana samo prema *GID* broju (bez imena grupe). To i nije problem samo po sebi, međutim prva slijedeća grupa koja će biti kreirana potencijalno će dobiti isti *GID* broj te će stoga imati ista prava pristupa odnosno ovlasti kao prijašnja grupa s istim *GID* brojem.

## Napredni rad s korisničkim grupama

Postoje i slučajevi kada imamo potrebu za željenu korisničku grupu kreirati lozinku. Ova lozinka se pohranjuje u datoteku: `/etc/gshadow` na isti način kako se i lozinke za korisnike pohranjuju u datoteku: `/etc/shadow`. Naime svaki korisnik mora pripadati minimalno jednoj primarnoj korisničkoj grupi i eventualno drugim sekundarnim korisničkim grupama.

Kada imamo potrebu da određeni korisnik, koji nije administrator, sâm, privremeno promjeni primarnu korisničku grupu koju želimo zaštititi, to možemo napraviti tako da za željenu korisničku grupu kreiramo lozinku.

Tada će bilo tko, tko poznaje lozinku za tu korisničku grupu, moći promijeniti svoju pripadnost toj (sada primarnoj) korisničkoj grupi. Tako će taj korisnik imati sva prava kod pristupanja i rada s datotekama ili direktorijima, koja ima navedena (zaštićena) korisnička grupa.

**12.** Za korisničku grupu imena `novi` (kao korisnik `root`), kreirajmo lozinku, pomoću naredbe `gpasswd`:

```
gpasswd novi

Changing the password for group novi
New Password:
Re-enter new password:
```

Sada trebamo upisati novu lozinku te ju ponovno potvrditi. Od tog trenutka korisnička grupa `novi` ima postavljenu lozinku, koja će biti zapisana u datoteku: `/etc/gshadow`.

Unos (redak) za grupu: `novi`, u ovoj datoteci, izgleda ovako:

```
grep novi /etc/gshadow

novi:$6$nc0dKhdsW6$HUKiz4JQMJRckJTbaps7vI.HT1sw02lRihLNVFhSmT7LcJjyGNRYdmHX.o5MQunSmY
LOm2ZCh6UfZfZ.Th4Rf/::
```

Dakle od sada je ova korisnička grupa zaštićena lozinkom.

**12.1.** Logirajmo se na sustav kao korisnik `pero`, te pogledajmo njegove pripadnosti grupama:

```
id pero

uid=100(pero) gid=1001(korisnici) groups=1001(korisnici),1002(korisnici2)
```

Vidimo da korisnik `pero`, pripada primarnoj grupi: (`korisnici`,s `gid=1001`) te ima pripadnost i sekundarnoj grupi: (`korisnici2`,s `gid=1002`). Sada, već logirani kao korisnik `pero`, privremeno promijenimo primarnu grupu u: `novi`, za što će nas sustav tražiti lozinku za ovu korisničku grupu.

Za tu namjenu, koristimo naredbu: `newgrp`, na sljedeći način.

```
newgrp novi
```



*Sve dok za neku grupu nismo kreirali lozinku, kao običan korisnik ju ne možemo koristiti kao svoju primarnu grupu, na gore navedeni način s naredbom `newgrp`.*
*Međutim korisničke grupe standardno imaju isključenu upotrebu lozinki i ovu mogućnost!*

Potom ponovno provjerimo pripadnost primarnoj grupi za korisnika: `pero`.

```
id
uid=100(pero) gid=1003(novi) groups=1003(novi),1001(korisnici),1002(korisnici2)
```

**Vidimo da sada kao korisnik `pero`, <u>privremeno</u> pripadamo novoj grupi `novi`, koja je zaštićena s lozinkom.**



Pohranjivanje sâme lozinke u **`/etc/gshadow`** je identično s **`/etc/shadow`**, pa pogledajte poglavlje: **7.1.3. Datoteka `/etc/shadow`.**

**Izvori informacija**: (K-12), (K-14), **`man id`**, **`man lslogins`**, **`man groups`**, **`man groupadd`**, **`man groupdel`**, **`man groupmod`**, **`man useradd`**, **`man userdel`**, **`man usermod`**, **`man passwd`**, **`man libuser.conf`**, **`man gpasswd`**, **`man newgrp`**.

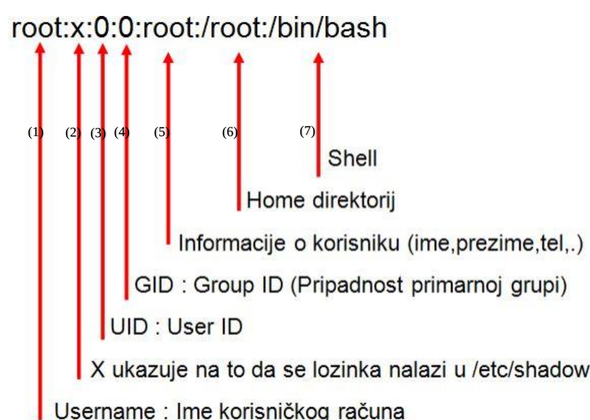
## 7.1.2. Datoteka `/etc/passwd`

***Slijedi napredno poglavlje!***
Datoteka `/etc/passwd` sadrži važne informacije koje su potrebne za prijavljivanje (*logiranje*) svakog korisnika na sustav, poput imena korisničkog računa, njegovog radnog (*home)* direktorija, njegove radne ljuske (*shella)* i slično. U ovoj datoteci nalaze se stupci konfiguracije odvojeni znakom za razgraničavanje odnosno delimitiranje `:`. Dakle format ove datoteke je ***CSV***.
Sve što smo radili u primjerima od prije, s gotovim naredbama, a vezano je za detalje oko korisničkog računa, zapravo se zapisuje u ovu datoteku. To znači da svaki puta kada pozivamo naredbe poput `lslogins`, `useradd`, `userdel` ili `passwd`, da se te promijene snimaju (i) u ovu datoteku. Tijekom logiranja na sustav, prvo se provjerava postoji li uopće određeni korisnik, a što je definirano u prvom stupcu u ovoj datoteci, pod ***username*** poljem koje označava ime korisničkog računa.
Pri tome ime korisničkog računa mora biti jedinstveno na sustavu.
Drugi stupac, koji je na današnjim sustavima uvijek postavljen na `x`, definira da se kriptirana lozinka; zapravo provjerni zbroj lozinke odnosno tzv. *hash*, nalazi pohranjen u datoteci `/etc/shadow` koja će biti kasnije objašnjena.

Osim toga važno je znati kako svaki korisnik može pripadati samo jednoj primarnoj grupi (pogledajte ***GID*** [*Group ID*] **(4)** stupac na slici dolje) te u više sekundarnih korisničkih grupa, koje su opisane u poglavlju o `/etc/group` datoteci.

Sadržaj datoteke `/etc/passwd` je vidljiv i opisan na slici **20.** (dolje):

**Slika 20. Sadržaj datoteke: `/etc/passwd`**




**(1)** - U prvom stupcu (s lijeve strane) stoji korisničko ime (*username*) koje, odnosno s kojim se logiramo (spajamo) na sustav odnosno Linux računalo i to lokalno ili udaljeno.
**(2)** - U drugom stupcu, u inačicama Linuxa i Unixa koji koriste softverski paket `shadow-utils`, a to je većina njih, stoji oznaka `x` koja označava da je lozinka pohranjena u vanjskoj odnosno drugoj datoteci: `/etc/shadow`.
**(3)** - U trećem stupcu se definira identifikacijski broj korisnika odnosno takozvani ***UID*** (engl. *User ID*) broj korisnika.
**(4)** - U četvrtom stupcu se definira broj primarne korisničke grupe kojoj ovaj korisnik pripada odnosno takozvani ***GID*** (engl. *Group ID*) broj grupe(sekundarne grupe se navode **drugdje**).
**(5)** - U petom stupcu se upisuju podaci o korisniku: ime, prezime, adresa, broj telefona i slično, odvojeni zarezom (`,`).

**(6)** - U šestom stupcu se definira putanja do korisnikovog početnog (kućnog) direktorija (engl. *home directory*); primjerice: `/home/pero`. U ovaj direktorij ovaj korisnik ima puna prava kreiranja pôddirektorija (pôd mapa) i datoteka kao i mijenjanja istih: od promjena ovlasti nad njima, promijene njihovog sadržaja ili njihovog brisanja.

**(7)** - U sedmom stupcu se definira program koji će se pokrenuti svaki puta kada se korisnik uspješno logira; ovdje se (obično) navodi ljuska (*shell)*: primjerice `/bin/bash` ili neka druga, koja će biti pokrenuta za korisnika u trenutku kada se logira.

Pogledajmo i skraćeni ispis sâmo jednog dijela ove datoteke:
```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
...
```
Kako je vidljivo i ova datoteka je u ***CSV*** formatu, u kojem je konkretno *delimiter* za stupce, znak `:`.
Objasnit ćemo unose za korisnika `root`, koji je upisan u prvi redak ove datoteke.
***Napomena: Jedan redak datoteke = definicija jednog korisničkog računa odnosno korisnika!***

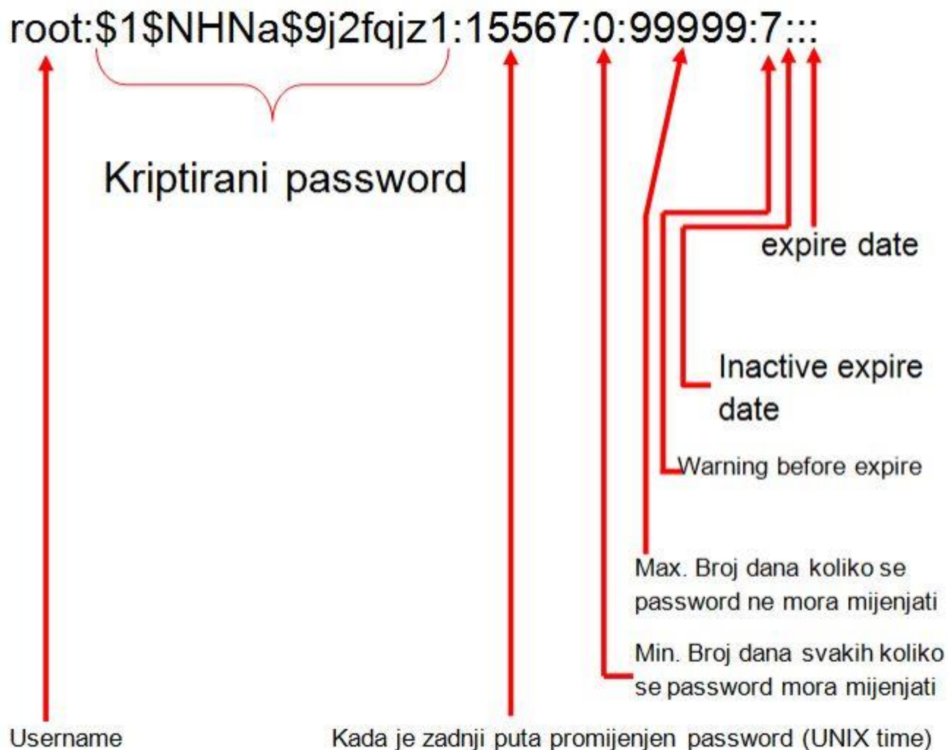
- U prvom stupcu vidimo kako je njegovo korisničko ime: `root`.
- U drugom stupcu je to standardno postavljeno `x` (definira da se ovdje ne pohranjuje lozinka).
- U trećem stupcu je `0`, što znači kako je identifikacijski broj (***UID***) za ovog korisnika broj nula (`0`).
- U četvrtom stupcu je isto `0`, što znači kako je identifikacijski broj primarne grupe (***GID***) za ovog korisnika nula (`0`). Pripadnost korisnika sekundarnim (drugim) grupama se definira u datoteci: `/etc/group`, a o njoj kasnije.
- U petom stupcu je oznaka `root`, što znači kako je za ime (prezime, adresu i slično) upisano samo `root`.
- U šestom stupcu je oznaka `/root`, što znači kako je početni (*home*) direktorij ovog korisnika, upravo vršni direktorij: `/root/`. Svi drugi korisnici imaju svoje početne (*home*) direktorije unutar vršnog direktorija: `/home/`.
- I konačno u zadnjem sedmom stupcu vidimo kako je radna ljuska (*shell*) koja će se pokrenuti svaki puta kada se ovaj korisnik logira na sustav biti ***bash*** ljuska, odnosno pokrenut će se program: `/bin/bash` koji je sâma ***bash*** ljuska.

**Izvori informacija: (584),(585),(K-12),(K-14), `man 5 passwd`.**

### 7.1.3. Datoteka `/etc/shadow`

***Slijedi napredno poglavlje!***
Sve naredbe koje smo koristili, a namijenjene su za definiranje i promjenu lozinke korisnika se zapravo zapisuju u ovu datoteku. Dakle datoteka `/etc/shadow` sadrži provjerni zbroj (engl. *Hash/checksum*) korisničke lozinke, kao i druge parametre vezane za korisnički račun. Naime nakon prvog koraka odnosno u trenutku logiranja korisnika na sustav i provjere postoji li uopće korisnik odnosno korisnički račun, sustav će vas zatražiti da upišete lozinku. Za lozinku se tada izračunava provjerni zbroj (*Hash*) i uspoređuje s onim već snimljenim, za konkretnog korisnika, a zapisanom upravo u ovoj datoteci (`/etc/shadow`) u drugom stupcu. Stupci su i ovdje odvojeni delimiterom `:`. Ako se provjerni zbroj upisane lozinke u trenutku pokušaja logiranja i onaj snimljen u ovu datoteku poklapaju, pristup sustavu se omogućava za konkretnog korisnika.
Izgled datoteke `/etc/shadow` je vidljiv na slici 21. dolje.

**Slika 21. Sadržaj datoteke:** `/etc/shadow`



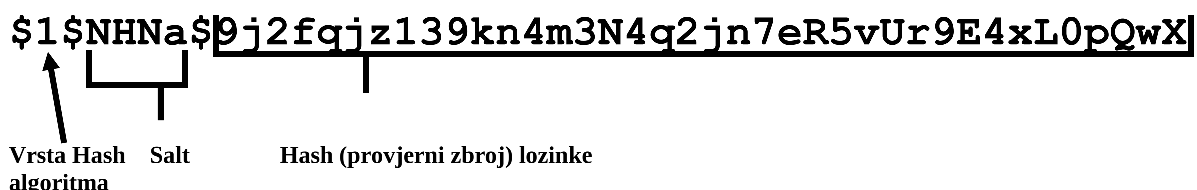

(1) - U ovom prvom stupcu s lijeve strane se nalazi ime korisničkog računa (engl. *Username*).
(2) - U drugom stupcu se nalazi tzv. *hash* vrijednost lozinke, ali u nekim slučajevima tu mogu biti i sljedeće vrijednosti:

- `!!` - znači kako je ovaj korisnički račun zaključan, ali mu se još može pristupiti, primjerice preko SSH protokola, upotrebom SSH ključa ali ne i lozinke.
- `*LK*` ili `*` - znači kako je ovaj korisnički račun zaključan, ali mu se još može pristupiti primjerice preko *SSH* protokola, i to upotrebom *SSH* ključa, ali ne i lozinke.

(3) - Na trećoj poziciji (stupcu) se u [UNIX vremenu (*epohi*)]() zapisuje kada je zadnji puta promijenjena lozinka.
(4) - Na četvrtoj poziciji (čija standardna vrijednost je **0**) se definira unutar koliko dana se lozinka ne mora mijenjati. Nula (0) znači kako ju je moguće mijenjati bilo kada.
(5) - Na petoj poziciji se definira nakon koliko dana se lozinka mora promijeniti. Vrijednost 99999 znači kako se lozinka ne mora (dugo) mijenjati.
(6) - Na šestoj poziciji je broj dana koliko će se korisnika upozoravati da mora promijeniti lozinku.
(7) - Na ovoj sedmoj poziciji se definira broj dana nakon što je lozinka istekla, ali će još uvijek korisnički račun biti aktivan.
(8) - Na osmoj poziciji se definira broj dana od **1.1.1970** od kako je korisnički račun isključen (ovo je tzv. ***Unix epoha***).
(9) - Na ovoj zadnjoj poziciji ne stoji ništa odnosno prazna je za buduću primjenu.

---

Sada pogledajmo samo dio kriptirane lozinke → dio: "***Kriptirani password***", sa slike 21, ali uvećan kako bismo mogli vidjeti njegove segmente:



**Što je opće "*password hash*" odnosno provjerni zbroj?**
To je matematička metoda/algoritam pomoću koje se na osnovu nîza ulaznih znakova, primjerice naše lozinke, stvara jedinstveni nîz znakova fiksne duljine. Ovo je jednosmjeran put što znači da nije moguće na osnovi ***hash*** nîza dobiti izvornu lozinku.
Dakle u `/etc/shadow` datoteku se pohranjuje "***password hash***", a ne sâma lozinka jer se niti u slučaju kada je sustav kompromitiran, lozinka neće moći vidjeti jer se ona niti ne sprema. Dakle spremanjem ovog provjernog zbroja (*Hash*) lozinke umjesto sâme lozinke se podiže razina sigurnosti, jer se na osnovu provjernog zbroja ne može doći do izvorne lozinke.

### 7.1.3.1. Provjerni zbroj (*hash*/*checksum*)

Provjerni zbroj odnosno ***Hash*/*Checksum*** je matematički algoritam odnosno jednosmjerna kriptografska *hash* funkcija pomoću koje se na osnovu ulaznog nîza podataka: slova/brojeva/znakova/… ili ulazne datoteke, izrađuje jedinstvena vrijednost koja je za primjerice ***MD5*** algoritam, uvijek dugačka točno 128 bitova. Njega u konkretnom slučaju čini heksadecimalni broj od trideset i dvije znamenke. Drugim riječima *hash* funkcije se koriste u sigurnosne svrhe za ratificiranje izvornosti datoteka ili podataka. ***MD5*** funkciju je dizajnirao Amerikanac ***Ronald Rivest*** 1991 godine, kao zamjenu za *hash* funkciju ***MD4***.
Međutim već 1996. pronađeni su nedostaci unutar ove funkcije koji nisu bili katastrofalni, no mnogi kriptografi savjetovali su uporabu drugih *hash* funkcija kao što su **SHA-1**, **SHA-256** ili drugih.
Vjerojatno ste nekada kopirali ***ISO*** sliku (ISO *image*) za CD ili DVD te ste uz njega primijetili primjerice oznaku: ***MD5 sum***. On je napravljen stoga da nakon što ste prekopirali tu ***ISO*** datoteku s mreže na svoje računalo, možete i na svom računalu napraviti ***MD5 sum*** odnosno ***MD5*** provjerni zbroj i usporediti vrijednost koju ste dobili sami, s onom koja se nalazi na stranici s koje ste kopirali ISO sliku (*image)* datoteku. Primjer je lista ***MD5*** provjernih zbrojeva (*checksuma*) za ***CentOS*** Linux v.6.5 je:

```
83221db52687c7b857e65bfe60787838  CentOS-6.5-x86_64-bin-DVD1.iso
91018b86ca338360bc1212f06ea1719f  CentOS-6.5-x86_64-bin-DVD2.iso
8c37390fa5d932d03feb05fba13fe92e  CentOS-6.5-x86_64-LiveCD.iso
7b571e13ce6c3f61dc176bd34f8d60bb  CentOS-6.5-x86_64-LiveDVD.iso
```

Kako ***MD5*** provjerni zbroj (*checksum* ili *hash*) napraviti na Linuxu, na datoteci: *`CentOS-6.5-x86_64-bin-DVD1.iso`* (to je ***CentOS*** instalacijska DVD *ISO slika*)?. To možemo napraviti pomoću naredbe `md5sum` na sljedeći način:

```
md5sum CentOS-6.5-x86_64-bin-DVD1.iso

83221db52687c7b857e65bfe60787838  CentOS-6.5-x86_64-bin-DVD1.iso
```

Vidimo kako smo dobili identičan *MD5* provjerni zbroj (`83221db52687c7b857e65bfe60787838`), što znači da se DVD ISO datoteka ispravno prekopirala s interneta, jer da je i jedan jedini bît podataka promijenjen i ***MD5*** izračun bi se promijenio!.

## Veza s datotekom: `/etc/shadow`

Osim ***MD5*** algoritma, postoje i drugi *hash* algoritmi za izradu provjernog zbroja, koji su pohranjeni u datoteci `/etc/shadow` u prvom stupcu (između prva dva znaka **$ $**) te su vidljivi kao slijedeći brojevi (s opisom u tablici dolje):

| Brojčana oznaka u Linuxu | *Hashing* algoritam (algoritam za provjerni zbroj) |
|---|---|
| 1 | MD5 |
| 2 | Blowfish |
| 5 | SHA-256 |
| 6 | SHA-512 |

Vratimo se na prethodno poglavlje i pogledajmo jedan unos iz `/etc/shadow` datoteke, konkretno za korisnika `root`:

```
root:$6$ulWMK97E$6rPR2q5dOpxzfUlfErcbgSbkHscskE0eHM3qJAU.lKhvJlW.hEfvHil4NljAKvbst.Wy
Hjs6JYDKZAw2/KjWW1:17842:0:99999:7:::
```

Naime u Linuxu, *hash* vrijednost lozinke se kreira kombiniranjem "***salt***-a" i željene lozinke, pomoću odabranog algoritma.
Pri tome je tzv. ***salt***, (zasoljivanje) dodatna riječ odnosno nîz znakova s kojom se lozinka dodatno šifrira odnosno „*začinjava*".

Kako provjeriti koji ***hash***-ing algoritam koristi naš Linux sustav? To možemo napraviti pomoću naredbe `authconfig`:

```
authconfig --test | grep hash
```

Rezultat našeg upita je:

```
password hashing algorithm is sha512
```

U našem slučaju koristi se *hashing* algoritam **6** odnosno **SHA-512**, pa ćemo u primjeru koji slijedi, koristiti isti algoritam.



Pogledajte poglavlje:
**7.1.6.5. Naredbe authconfig i authselect za rad s korisničkim postavkama.**

**Kako možemo ručno kreirati lozinku, korištenjem *salt*-a (*zasoljavanja*) i željenog algoritma?**
Koristit ćemo programski jezik python iz ljuske (*shella)* te pokrenuti sljedeće (u jednom retku):

```
python -c 'import crypt; print crypt.crypt("ovo_je_password","$6$ovo_je_salt")'
```

Dakle ***python*** skripta nam je kreirala sljedeću šifriranu (kriptiranu) lozinku:

```
$6$ovo_je_salt$fPiMsDoTHyyQ9R6qx3cIICWKOCqD8dfBzMTrF8rnRMHsETgsU.
```

Rezultat odnosno ispis koji smo dobili je funkcionalan dio korisničke Linux lozinke, koju smo na dvije slike prije (slika 21.) označili kao dio pod nazivom "**Kriptirani password**". Ovakav nîz karaktera odnosno znakova je potpuno funkcionalan za upotrebu u /etc/shadow datoteci na pravom mjestu; opisanom i vidljivom na istoj slici.


Važno je razumjeti kako se promjenom sâmo jednog znaka (ili čak jednog bîta podataka) u tekstu ili unutar datoteke za koju radimo izračun *hash* vrijednosti, potpuno mijenja konačno izračunata *hash* vrijednost.

Pošto ***MD5*** nije jedini algoritam za izračun *hash* vrijednosti u linuxu, stoga obično imamo i naredbe: sha1sum za *SHA-1* algoritam, sha256sum za *SHA-256* algoritam te sha512sum za *SHA-512* algoritam. Osim navedenih, postoje i još jednostavniji algoritmi poput ***CRC*** (engl. *cyclic redundancy check*) koji možemo koristit s naredbom cksum.

Pogledajmo i nekoliko primjera u kojima ćemo napraviti *hash*/*checksum* cijele datoteke /etc/shadow korištenjem različitih algoritama:

**1.** Pomoću **CRC** algoritma, uz pomoć naredbe cksum:

```
cksum /etc/shadow
685062379 1239 /etc/shadow
```

Vidimo kako smo dobili jedinstvenu vrijednost: 685062379

**2.** Pomoću **MD5** algoritma, uz pomoć naredbe md5sum:

```
md5sum /etc/shadow
af5130663645700a13ba68f7dda54022  /etc/shadow
```

Ovdje smo dobili jedinstvenu vrijednost: af5130663645700a13ba68f7dda54022

**3.** Pomoću **SHA-1** algoritma, uz pomoć naredbe sha1sum:

```
sha1sum /etc/shadow
21d954dff78b99ce093f3c781edd975fab4f481b  /etc/shadow
```

Ovdje smo dobili jedinstvenu vrijednost: 21d954dff78b99ce093f3c781edd975fab4f481b

Vidljivo je kako svaki algoritam generira jedinstvenu izlaznu vrijednost, obavezno i isključivo sâmo iste duljine:

| Algoritam | Duljina *hash* vrijednosti |
|---|---|
| **CRC** | 9 brojeva |
| **MD5** | 32 znaka (*heksadecimalno*) |
| **SHA-1** | 40 znakova (*heksadecimalno*) |
| **SHA-256** | 64 znaka (*heksadecimalno*) |
| **SHA-512** | 128 znakova (*heksadecimalno*) |


Konfiguracija opcija i parametara vezanih za rad s korisničkim računom i načinom kriptiranja lozinke definiran je u konfiguracijskoj datoteci: /etc/login.defs.


Stoga pogledajte (proučite) poglavlje:
**7.1.6.2. Datoteka /etc/login.defs.**

**Izvori informacija: (577),(578),(579),(580),(581),(K-12),(K-14),** `man 5 shadow`, `man libuser.conf`.

### 7.1.4. Datoteka `/etc/group`

***Slijedi napredno poglavlje!***
Rad svih naredbi koje smo koristili, a namijenjene su za kreiranje i promjene korisničkih grupa se zapravo zapisuje u ovu datoteku. Datoteka `/etc/group` sadrži korisničke grupe i pripadnosti korisnika tim grupama. U ovoj datoteci svaki redak predstavlja jedan unos, a unutar svakog novog retka su stupci odvojeni delimiterom koji čini znak `:`. U prvom stupcu **(1)** je naziv grupe. A u drugom stupcu **(2)** u današnjim inačicama *Linuxa* i *Unixa* stoji oznaka `x` koja označava kako je lozinka pohranjena u vanjskoj odnosno drugoj datoteci: `/etc/shadow`. Nakon toga u trećem stupcu**(3)** se definira identifikacijski broj grupe to jest ***GID*** broj. Naime svaka grupa ima svoj identifikacijski broj, koji se nalazi na ovoj trećoj poziciji. Primjerice grupa `bin` ima svoj identifikacijski broj `1` i njoj pripadaju korisnici `bin` i `daemon` koji se definiraju na četvrtoj **(4)** poziciji odnosno stupcu.

Lista korisnika se odvaja zarezom, ako ih ima više u određenoj grupi. Izgled `/etc/group` datoteke je vidljiv na slici **22.** (dolje).

**Slika 22. Datoteka `/etc/group`**

root:x:0:
bin:x:1:bin,daemon
daemon:x:2:bin,daemon

Lista korisnika koji su članovi ove grupe
GID (Group ID) od ove grupe
Password - X znači da se ne koristi (default)
Ime grupe

Pogledajmo i skraćeni sadržaj naše datoteke `/etc/group`

```
root:x:0:
bin:x:1:bin,daemon
daemon:x:2:bin,daemon
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:
mem:x:8:
wheel:x:10:
cdrom:x:11:
mail:x:12:postfix
man:x:15:
ftp:x:50:
korisnici:x:1001:ivan,pero,ana
```

Ovdje su jasno vidljivi navedeni stupci (odvojeni dvotočkom [**:**] kao znakom delimiterom):
**(1)** - U ovom prvom stupcu s lijeve strane se nalazi ime korisničke grupe (engl. *Group name*), primjerice `root:`.
**(2)** - U drugom stupcu s lijeve strane se nalazi x**:** što označava da ne postoji definirana lozinka za korisničku grupu.
**(3)** - U trećem stupcu s lijeve strane je upisan broj grupe to jest grupni ID (engl. Group ID), primjerice `0`.
**(4)** - Nakon toga slijedi lista korisnika ili korisnik koji pripada ovoj grupi to jest koji su članovi ove grupe.



Vezano za mogućnost kreiranja lozinke za grupu, pogledajte poglavlje:
**7.1.1. Osnovne naredbe za rad s korisničkim računima i grupama.**
Posebno pogledajte cjelinu: **„Napredni rad s korisničkim grupama".**

**Izvori informacija: (582),(583),(K-12),(K-14), `man 5 group`.**

## 7.1.5. Izvršavanje naredbi i prebacivanje rada s jednog na drugog korisnika

***Slijedi napredno poglavlje!***
Prema sigurnosnim preporukama, na sustav bi se trebali uvijek prijaviti sa svojim korisničkim računom, a ne kao ***root*** korisnik koji je ekvivalent administratorskom korisniku na ***Windowsima***. Stoga u određenim slučajevima imamo potrebu, iako smo logirani kao *obični* korisnik, izvršiti neku naredbu kao ***root*** ili neki drugi korisnik. Za tu namjenu postoji naredba `su` (engl. *Substitute user*) koja u doslovnom prijevodu znači: *zamijeni korisnika*. Pomoću ove naredbe možemo izvršavati naredbe ili pokretati programe s privilegijama nekog drugog korisnika. Standardnim pozivanjem ove naredbe pokreće se nova naredbena ljuska (*shell*), ali unutar našeg trenutnog radnog direktorija (mape) i s našim varijablama sustava (*environment variable*).
Ako ovu naredbu pokrećemo bez opcija ili definicije u kojeg (drugog) korisnika se želimo prebaciti, ona će nas prebaciti u korisnika ***root*** (UID=0). Stoga ju ovako i pokrenimo (prvo smo prijavljeni kao korisnik ***hrvoje***, a potom ćemo postati ***root***):

```
[hrvoje@localhost ~]$ su
Password:
[root@localhost hrvoje]#
```

Nakon što smo pokrenuli ovu naredbu, sustav će nas zatražiti lozinku za korisnika koji želimo postati (ovdje je to ***root***).
Potom nas je sustav prebacio na korisnika ***root***, a zatim dobivamo sva njegova prava odnosno ovlasti, ali nemamo postavljene i njegove sistemske varijable (`PATH`, `USER`, `MAIL`,…). Ako ipak želimo da se postave i sve sistemske varijable, ali i sve ostalo, kao da smo se inicijalno logirali kao ***root*** korisnik, tada je potrebno naredbu `su` pokrenuti na sljedeći način:

```
su - root
```

Ovdje smo sa znakom `-` naznačili sustavu da postavi sve potrebno kao da smo se logirali kako korisnik koji slijedi, tj. konkretno korisnik `root`. Nakon toga možemo raditi sve kao da smo se inicijalno logirali kao navedeni korisnik, odnosno pokretati željene naredbe i programe te pristupati datotekama koje možemo koristiti samo kao navedeni korisnik ***root***.
S ovom naredbom možemo postati i bilo koji drugi korisnik.
Na kraju se vraćamo u bivšeg korisnika s naredbom:

```
exit
```

**Što kada ne želimo postati neki drugi korisnik već samo želimo pokrenuti neku naredbu kao traženi korisnik?**
Tada nam naredba `su` nije od pomoći, već se moramo osloniti na drugi mehanizam. Naime za tu potrebu moramo koristiti naredbu `sudo` (engl. *Superuser do*) pomoću koje možemo pokretati programe/naredbe s privilegijama nekog drugog korisnika. Najčešće je slučaj da kao običan korisnik želimo pokrenuti neku naredbu za koju su nam potrebne privilegije (*ovlasti*) administratora odnosno korisnika ***root***. Međutim i s naredbom `sudo` se možemo prebaciti u željenog korisnika, iako to nije njena primarna namjena. Pogledajmo kako to napraviti, a pritom povući i sve njegove (***root***) varijable i postavke sustava:

```
sudo -i -u root
```

Sada ćemo se vratiti na normalan i uobičajeni način upotrebe naredbe `sudo`. Za razliku od naredbe `su`, s kojom kako bismo postali neki drugi korisnik, moramo upisati (poznavati) lozinku tog korisnika, s naredbom `sudo` to nije slučaj. Naime s naredbom `sudo` kada želimo pokrenuti naredbu kao neki drugi korisnik, odnosno s njegovim privilegijama/ovlastima, sustav će nas zatražiti našu lozinku, a ne lozinku korisnika s čijim privilegijama ju pokrećemo. Naredba koja će biti pokrenuta pomoću tog mehanizma, u ime drugog korisnika (obično je to korisnik ***root***), biti će pokrenuta s ***UID*** brojem tog korisnika (***root, UID=0***) te s njegovom primarnom korisničkom grupom (za ***root*** korisnika je to ***GID=0***).
Međutim, ako baš imamo takvu potrebu, moguće je zaobići upotrebu primarno definirane korisničke grupe i definirati neku drugu, pomoću prekidača `-g` nakon kojeg treba slijediti željena korisnička grupa.

### Naredba sudo

**Nije li to pokretanje programa s *root* ovlastima bez *root* lozinke potencijalni sigurnosni problem?**
Zapravo nije, jer nemaju svi korisnici prava na ovakvu upotrebu naredbe `sudo` odnosno prava na pokretanje naredbi/programa kao administratori odnosno kao korisnik ***root***. Važno je razumjeti kako *sudo* koristi napredni modularan sustav sigurnosnih politika. Inicijalna konfiguracija *sudo* sustava (i njegovih modula) nalazi se u datoteci: `/etc/sudo.conf`.

Sigurnosni modul koji je primarno u upotrebi se zove ***sudoers*** modul. Dakle kada kao *običan* korisnik želimo pomoću naredbe `sudo` pokrenuti neku naredbu za koju samo ***root*** korisnik ima pravo, sustav prvo provjerava, ima li naš korisnički račun prava na to, a kako je definirano u konfiguracijskoj datoteci: `/etc/sudoers`.

Provjerimo ima li korisnik `ivan` *sudo* prava, na sljedeći način:

```
sudo -lU ivan
User ivan is not allowed to run sudo on localhost.
```

Dakle vidimo kako konkretni korisnik `ivan` nema *sudo* prava (`not allowed`).
Vratimo se na konfiguracijsku datoteku *sudo* sustava. Ovu datoteku možemo direktno otvoriti s nekim uređivačem teksta mada je zbog sintaksne provjere preporučivo otvarati ju s naredbom `visudo`. U pravilu, novi korisnički računi i njihove privilegije se dodaju na kraj ove datoteke. Međutim prvo se upoznajmo sa osnovama sintakse ove datoteke.
Prvi veći dio ove datoteke sadrži naredbe `Defaults` uz koju slijede parametri, opcije i vrijednosti koje se postavljaju kao zadane (*default*); od varijabli koje se postavljaju: lokalne ili sistemske, te drugih postavki *sudo* sustava. Pošto ovdje postoji na stotine opcija i parametara (*aliasi, grupe naredbi, grupe korisnika i drugo*) mi ih nećemo sve spominjati.
U prvom primjeru ćemo se fokusirati samo na jedan mali dio konfiguracije vezan za definiciju rada s novim korisničkim računima kojima želimo dodati mogućnosti pokretanja **SVIH** naredbi i programa za koje su inače potrebne ***root*** ovlasti. Ako smo kreirali korisnika `hrvoje` i njemu želimo dati sva navedena prava, tada na kraj datoteke `/etc/sudoers` dodajmo sljedeće:

```
hrvoje ALL=(ALL) ALL
```

Važno je razumjeti kako se u svakom novom redu ove datoteke definira jedno pravilo.
Ovo naše pravilo (ovaj red) znači slijedeće: prva oznaka, recimo da je to stupac, označava kako slijede pravila za korisnika `hrvoje`, koji nadalje; s bilo kojeg terminala (`ALL=`) i to kao bilo koji korisnik (druga oznaka `(ALL)`) ima prava pokretati bilo koji program odnosno naredbu (treća oznaka `ALL`).



Preporuka za dodavanje novih unosa jest kreiranje zasebnih datoteka unutar direktorija: `/etc/sudoers.d/`, kako bi bilo lakše pratiti promjene na sustavu. Datoteke u ovom dirktoriju se čitaju prema numeričkom redoslijedu.

Primjerice kreirajmo datoteku: `/etc/sudoers.d/1-admini` u koju ćemo za početak dodati isti sadržaj kao u primjeru gore.
Potom moramo promjeniti ovlast za ovu datoteku te testirati sintaksnu ispravnost:

```
cd /etc/sudoers.d/
chmod 0440 1-admini
visudo -c
/etc/sudoers: parsed OK
/etc/sudoers.d/1-admini: parsed OK
```

← Vidimo da je sve prošlo uredno.

Probajmo se sada logirati kao korisnik `hrvoje`. Potom ćemo probati kreirati mrežno pôd sučelje `enp0s17:1` s pripadajućim IP parametrima, za što inače moramo imati ***root*** prava odnosno bez njih to nije moguće izvesti:

```
sudo ifconfig enp0s17:1 192.168.2.200 netmask 255.255.255.0
```

Nakon toga ćemo upisati lozinku za korisnika `hrvoje` i ova naredba će biti izvršena s privilegijama ***root*** korisnika.
U novom primjeru ćemo kreirati jednu ***sudo*** grupu korisnika: `GRUPA` u koju ćemo dodati korisnike: `hrvoje` i `ivan`.
Potom ćemo kreirati pseudonim (*alias*) grupu za naredbe u koju ćemo nanîzati naredbe koje dodjeljujemo tom *aliasu* za naredbe, imena: `GASENJE`.

Naime ovdje ćemo nanîzati naredbe (nîzanje se odvaja zarezom) s kojima se računalo može ugasiti ili restartati, a koje ćemo potom dodijeliti korisnicima *sudo* grupe: `GRUPA`.
S time će navedeni korisnici dobiti prava izvršavanja navedenih naredbi, koja do tada nisu imali.

```
User_Alias      GRUPA = hrvoje, ivan
Cmnd_Alias      GASENJE = /sbin/shutdown, /sbin/halt, /sbin/reboot, /sbin/restart
GRUPA ALL = GASENJE
```

Pogledajmo usporedbu osnovnih značajki rada naredbi `su` i `sudo`:

| Značajka | Upotreba naredbe su | Upotreba naredbe sudo |
|---|---|---|
| **Korisnički pristup** | Omogućuje pristup kao **root** ili drugi korisnik | Omogućuje pristup kao **root** ili drugi privilegirani korisnik |
| **Potreba za lozinkom** | Zahtjeva **root** lozinku | Zahtjeva korisničku lozinku |
| **Bilježenje aktivnosti** | Bilježi se samo upotreba naredbe `su` | Bilježi se pokretanje naredbi u datoteci `/var/log/secure` |
| **Granularan kontrola** | Ograničena granularna kontrola | Omogućuje granularnu kontrolu pristupa putem konfiguracije *sudoers* datoteke |
| **Povijest izvršenih naredbi** | Ne prati se | Može se pratiti, ako je tako konfigurirano |



Pogledajte i primjere upotrebe u slijedećim poglavljima:
**28.3. Sigurnosne preporuke** i to primjer broj **10.2.**
**4.5.7. Datoteka *limits.conf* i druga ograničenja sustava** (primjer na kraju).
**7.1. Rad s korisničkima računima, grupama i lozinkama.**
**6.2.2. Sistemske (*Environment*) varijable i postavke terminala** te varijablu **`EUID`**.

**Izvori informacija: (918),(919),(K-12),(K-14),** **`man su`**, **`man sudo`**, **`man sudoers`**, **`man visudo`**, **`man sudo.conf`**.

## 7.1.6. Sigurnosne postavke i ograničenja sustava

***Slijedi napredno poglavlje (7.1.6.x)!.***
U ovoj cjelini upoznati ćemo se s konfiguracijskim datotekama koje su vezane za sigurnosne i druge postavke te ograničenja sustava vezana za rad s korisničkim računima i grupama, drugim ograničenjima korisničkih računa i grupa te drugih opcija vezanih za korisničke račune. Postavke za pravila o minimalnoj složenosti tijekom kreiranja nove lozinke su definirana u datoteci: `/etc/security/pwquality.conf`, a ostala pravila ćemo spomenuti u narednim cjelinama.

### 7.1.6.1. Što se događa tijekom prijavljivanja (logiranja) na sustav

Nakon pokretanja Linuxa, u trenutku nakon što se pokrenuo prvi proces (***init*** ili ***systemd***) te kada su se inicijalizirale sve komponente sustava, sustav je spreman za rad s korisnicima. Međutim ***init*** ili ***systemd*** (ovisno o kojem sustavu se radi [***RedHat/CentOS*** 6.x ili noviji]) nisu zaduženi sâmo za pokretanje skripti koje pokreću (ili prema potrebi zaustavljaju) sistemske servise. Naime jedna od njihovih odgovornosti je i pokretanje programa koji korisnicima omogućuju prijavu na sustav. Za terminal (ili virtualnu konzolu) koriste se dva programa, a to su: `getty` odnosno novija implementacija `agetty` i program `login`. ***Getty*** je kratica za "*get terminal*" odnosno „*pozovi terminal*“. Osnovni program **`getty`** otvara terminalni uređaj, inicijalizira ga, ispisuje prijavu i čeka unos korisničkog imena. Suvremeni **`getty`** programi (postoji ih nekoliko dostupno za Linux; primjerice: `agetty`) mogu raditi i druge stvari. Ovisno o tome što se koristi za pristup terminalu.
Koji ***getty*** će se koristiti, ovisi o tome kako je definirano na sustavu:
- Za ***Init*** sustav (***RedHat/CentOS*** 6.x i Linuxe koji koriste ***SystemV***), je ***getty*** definiran u datoteci: `/etc/inittab`.
- Za ***Systemd*** sustav (***RedHat/CentOS*** 7.x i novije), je ***getty*** definiran u posebnom servisu: `getty@.service`.

Kada se netko želi prijaviti na sustav (*logirati*), tada **`getty`** odnosno obično `agetty` izvršava svoju rutinu za prijavu, na način da inicijalizira terminal, na njemu ispisuje sadržaj datoteke: `/etc/issue` (obično je to poruka koja prikazuje ime računala i inačicu kernela) te prikazuje poruku `login:`. On zatim traži od korisnika korisničko ime za prijavu putem narednog retka. Nakon što korisnik upiše korisničko ime i potvrdi ga (s *ENTER*), ***(a)getty*** pokreće program `login` te njemu prosljeđuje upravo upisano ime korisnika kao parametar. Program `login` tada prikazuje poruku da je potrebno upisati lozinku, obično sa: `password:`. Nakon toga potrebno je i upisati lozinku za konkretan korisnički račun.
Sada kada program `login` ima i korisničko ime i pripadajuću lozinku, on dalje ovisno o konfiguraciji sustava:
- Ili provjerava direktno dostupnost korisničkog računa u datoteci: `/etc/passwd` te potom napravi provjerni zbroj lozinke (*hash*) i usporedi ju s onim pohranjenim u datoteci: `/etc/shadow`. Ako se podudaraju, tada se omogućuje pristup sustavu. Konfiguracija ovog ponašanja je definirana u datoteci: `/etc/nsswitch.conf`, ali se čitaju i postavke iz datoteke: `/etc/login.defs`, poput algoritma (***MD5***/***SHA***/…) za spremanje lozinke i slično.
- Ili koristi ***PAM*** (engl. *Pluggable Authentication Modules*) mogućnosti sustava, kako je opisano u sljedećem poglavlju. Naime ovdje se prvo provjerava konfiguracijska datoteka≛: `/etc/pam.d/passwd` u kojoj su definirane metode dohvaćanja potrebnih datoteka te postavki sustava (poput algoritma za spremanje lozinke).
  Na ***RedHat/CentOS 7+*** sustavima navedena datoteka≛ poziva datoteku u kojoj su navedena sva pravila za korisničke račune, lozinke i slično, a to je datoteka: `/etc/pam.d/system-auth`, što je vidljivo u njenim≛ slijedećim linijama odnosno redcima:

```
auth        include       system-auth
account     include       system-auth
password    substack      system-auth
```


*U slučaju udaljenog pristupa na računalo, primjerice pomoću **SSH** protokola; događa se slična procedura.*
*Međutim sada **sshd** servis umjesto **getty** traži unos imena korisnika i prikazuje poruku za unos.*
*Nakon toga **sshd** pokreće program `login` za nastavak provjere, prosljeđujući mu upravo upisano ime korisnika kao parametar.*
*Zatim `login` nastavlja s već opisanim radom.*

Tijekom spajanja na lokalni terminal, ako je pristup sustavu odobren, ***(a)getty*** pokreće ljusku (*shell*), na standardan način opisan u poglavlju: **3.1.5. Bash ljuska detaljnije.** Isto se događa i tijekom spajanja s udaljenog terminala (pr. *SSH*).


Pogledajte i slijedeća poglavlja:
**7.1.6.2. Datoteka /etc/login.defs.**
**7.1.6.3. Datoteka /etc/nsswitch.conf.**
**7.1.6.4. Datoteka /etc/libuser.conf.**
**7.1.6.5. Naredbe authconfig i authselect za rad s korisničkim postavkama.**

**Izvori informacija: (938),(939),(940),(941),(K-14), `man agetty`, `man login`, `man 5 issue`, `man 5 login.defs`**

### 7.1.6.2. Datoteka /etc/login.defs

U datoteci /etc/login.defs nalaze se razne postavke vezane za korisničke račune, od trenutka prijavljivanja na sustav, kao i za kreiranje novih korisnika ili korisničkih grupa ili promjene njihovih parametara rada. Opcije i parametri definirani ovdje se primjenjuju na cijeli sustav odnosno koriste ih razne naredbe, a imaju i najveću težinu (u odnosu na druge konfiguracijske datoteke). Ova konfiguracijska datoteka dolazi unutar „*Shadow password*" softverskog paketa, a u ***CentOS***/***RedHat*** Linuxu je to shadow-utils u kojem se nalaze i osnovni mehanizmi u kojima su definirane opcije i mogućnosti koje se zapisuju u datoteku /etc/shadow, kao i druge datoteke koje koristi cijeli operacijski sustav, a tiču se postavki korisničkih računa.
Pogledajmo sâmo mali dio onoga što sve možemo definirati u datoteci /etc/login.defs:

- PASS_MAX_DAYS – maksimalan broj dana unutar kojih se lozinka smije koristiti. Nakon ovog perioda se lozinka mora promijeniti.
- PASS_MIN_DAYS – minimalan broj dana između dvije promijene lozinke. Bilo koji pokušaj promijene lozinke, prije vremena definiranog ovdje, bit će odbačen.
- PASS_WARN_AGE – broj dana, prije nego se lozinka mora promijeniti, kada će biti poslano upozorenje kako se lozinka mora promijeniti. Ako nije ništa upisano, ne šalje se nikakva poruka. Nula (0) znači kako se poruka šalje isti dan kada se mora mijenjati lozinka.
- LOGIN_RETRIES – maksimalni broj pokušaja ponovnog upisivanja lozinke, ako smo nekoliko puta upisali krivu.
- ENCRYPT_METHOD – definicija algoritma s kojim se kriptira lozinka u datoteci: /etc/shadow
  - SHA512 - *SHA512* algoritam. Pr *SHA512* će u /etc/shadow biti vidljivo kao $6$.
- Vezano za korisničke grupe i naredbe: useradd, groupadd ili newusers:
  - GID_MIN – tijekom kreiranja novih korisničkih grupa (koje se zapisuju u /etc/group), od kojeg ***GID*** (*Group ID*) broja se kreće. Postoji i SUB_GID_MIN za *namespaces* izolaciju.
  - GID_MAX – tijekom kreiranja novih korisničkih grupa (koje se zapisuju u /etc/group), do kojeg ***GID*** (*Group ID*) broja se mogu kreirati grupe. Postoji i SUB_GID_MAX za *namespaces* izolaciju.
- Vezano za **ID** korisnika i kreiranje novih korisničkih računa te naredbe: useradd i newusers:
  - UID_MIN – tijekom kreiranja novih korisnika, od kojeg ***UID*** broja se kreće (standardno je 1000).
  - UID_MAX – tijekom kreiranja novih korisnika, do kojeg ***UID*** broja se ide (standardno je 60000).
  - SUB_UID_MIN i SUB_UID_MAX – koriste se za izolirane prostore Linuxa – pr. za *Linux kontejnere*.
  - UMASK – tijekom kreiranja novih korisnika, koji se *UMASK* koristi za kreiranje njihovih *home* direktorija.

Pogledajte i detalje o postavkama vezanim za kreiranje korisnika**: 7.1.6.4. Datoteka /etc/libuser.conf.**

Osnovni rad s korisničkim računima, vezanim uz vremenske okvire rada korisničkih računa, omogućava nam naredba chage. Parametri koji utiču na rad ove i sličnih naredbi za korisničke račune su također definirani u datoteci /etc/login.defs

**Pogledajmo i nekoliko primjera**
Promijenimo vrijeme nakon kojeg korisnik pero mora promijeniti lozinku, na 60 dana, pomoću naredbe chage:

```
chage -M 60 pero
```

Natjerajmo korisnika pero da mora promijeniti lozinku nakon prvog prijavljivanja na sustav (*logiranja*):

```
chage -d 0 pero
```

## *PAM* (engl. *Pluggable Authentication Modules*)

Za naprednije funkcionalnosti, potrebna je upotreba nadogradnje ovog sustava, koja dolazi s Linux **PAM** (engl. *Pluggable Authentication Modules*) mogućnostima, a koje su danas standardno dostupne u Linuxu te koje ovdje nećemo detaljno objašnjavati. Naime upotrebom ***PAM***-a dobivamo detaljnije mogućnosti kontrole sustava autentikacije odnosno provjere autentičnosti korisnika u kombinaciji s aplikacijama i servisima (po potrebi). ***PAM*** koristi module, od kojih je svaki zadužen za određenu funkcionalnost. Linux ***PAM*** dijeli zadatke provjere autentičnosti na četiri neovisne grupe za upravljanje:

- Moduli za korisničke račune (engl. *Account modules*) provjeravaju je li navedeni korisnički račun važeći u cilju provjere autentičnosti pod određenim postavljenim uvjetima. To može uključivati uvjete kao što su istek korisničkog računa, doba dana i ima li korisnik prava pristupa traženoj usluzi (pr. pam_sss.so, pam_unix.so).
- Moduli za provjeru autentičnosti (engl. *Authentication modules*) provjeravaju identitet korisnika, primjerice tražeći i provjeravajući lozinku (zaporku), sigurnosni ključ ili neki drugi mehanizam. Također mogu proslijediti informacije o autentičnosti drugim sustavima. Neki od ***PAM*** modula ovdje su: am_localuser.so,pam_env.so.
- Moduli za zaporke/lozinke (engl. *Password modules*) odgovorni su za ažuriranje lozinki i općenito su povezani s modulima koji se koriste u koraku provjere autentičnosti. Također se mogu koristiti za nametanje jakih (kompliciranijih) zaporki. Neki od ***PAM*** modula za ovu namjenu su: pam_unix.so,pam_pwquality.so.
- Moduli sesije (engl. *Session modules*) definiraju akcije koje se izvode na početku i na kraju sesija. Sesija počinje nakon što je korisnik uspješno autenticiran. Neki od njih su: pam_limits.so,pam_keyinit.so.

Dodatno, ako su i servisi (pr. *SSH servis*) kompilirani s podrškom za ***PAM*** te, ako za njih postoji ***PAM*** modul i mogućnost kontrole, tada dobivamo još jednu razinu kontrole. Primjerice provjerimo je li naš *SSH* servis (/usr/sbin/sshd) kompiliran s podrškom za ***PAM***. To ćemo provjeriti pomoću naredbe ldd na sljedeći način:

```
ldd /usr/sbin/sshd | grep libpam.so
        libpam.so.0 => /lib64/libpam.so.0 (0x00007f5c29dc7000)
```

Pošto se koristi ***PAM*** biblioteka (libpam.so.0) to znači kako imamo podršku za ***PAM*** (nju nećemo dodatno objašnjavati).

**Izvori informacija: (586),(587),(588),(K-14), man chage,man ldd,man pam_unix,man pam_localuser, man pam_sss, man 5 login.defs.**

### 7.1.6.3. Datoteka `/etc/nsswitch.conf`

*Name Service Switch* sustav povezuje računalo s raznim izvorima podataka i mehanizmima za razrješavanje imena; primjerice s datotekama: `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/hosts`, … Konfiguracijska datoteku *Name Service Switch* sustava (***NSS***) je datoteka: `/etc/nsswitch.conf`. Ovu datoteku koriste ***GNU C*** biblioteke za određivanje izvora informacija iz kojih se dobivaju podaci važni za rad sustava vezan za korisnike, korisničke grupe, imena računala i slično.
Ova datoteka je razdvojena u nîz kategorija koje sustavu daju određene informacije. Svaka kategorija informacija identificira se nazivom baze podataka koju koristi kako bi dohvatila traženu informaciju. To je obična tekstualna datoteka u kojoj su stupci odvojeni razmacima ili znakovima tabulatora. Prvi stupac navodi ime kategorije odnosno indirektno njene baze podataka. Preostali stupci opisuju redoslijed izvora (resursa) za upit i ograničen skup radnji koje se mogu izvršiti rezultatom pretraživanja. Naime ova datoteka određuje gdje sustav može pronaći podatke kao što su korisnici i njihove lozinke, korisničke grupe, imena računala (*hostova*) i slično. Ove podatke sustav može pronaći u nekim od lokalnih datoteka, ili pomoću nekih od mrežnih usluga odnosno servisa koje mu definiramo. U ovoj datoteci kategorije se definiraju na sljedeći način:

**Kategorija resurs1 resurs2 resurs *n***

Pogledajmo i jedan mali dio datoteke `/etc/nsswitch.conf`

```
passwd:    files sss
shadow:    files sss
group:     files sss
hosts:     files dns
```

Na ispisu iz primjera datoteke `/etc/nsswitch.conf` vidimo sljedeće kategorije, od gore do dolje, redak po redak:

- `passwd:` - kada sustav traži informacije o korisnicima kao u trenutku kada se neki korisnik želi logirati na sustav, prvo će ih tražiti u lokalnim datotekama (`files`), što konkretno podrazumijeva datoteku: `/etc/passwd` u kojoj su pohranjeni lokalni korisnici. A ako tamo korisnik nije pronađen, sustav će ga pokušati pronaći pomoću lokalnog ***SSS*** servisa (engl. *System Security Services Daemon*) (`sss`), koji se konfigurira u: `/etc/sssd/sssd.conf`.
- `shadow:` - definira da kada je korisnik pronađen u prvom koraku, njegova lozinka (definirana s `files`) će se prvo tražiti u lokalnoj datoteci `/etc/shadow`, a ako se tamo ne pronađe, ponovno će se kontaktirati ***SSS*** servis jer je on naveden kao drugi resurs u nîzu (`sss`).
- `group:` - u slučaju kada je potrebno provjeriti pripadnost korisničkim grupama, također će se prvo provjeravati lokalna konfiguracijska datoteka (definirana s `files`) u kojoj su kreirane korisničke grupe i pripadnosti korisnika tim grupama, u datoteci: `/etc/group`. Ako se korisnička grupa ovdje ne pronađe, tražit će ju se pomoću ***SSS*** servisa (`sss`).
- `hosts:` – kada se pokušava razlučiti IP adresa iz imena nekog računala; primjerice onog kojemu pristupamo preko mreže, prvo se provjerava lokalna datoteka: `/etc/hosts` (definirano s `files`), a tek potom vanjski ***DNS*** poslužitelji koji se ne definira ovdje već u datoteci: `/etc/resolv.conf` (definirano s `dns` ili `sss`).

🎯 Pogledajte i detalje o postavkama vezanim za razlučivanje **IP** adresa iz imena računala ili **FQDN** imena:
**25.6. Osnovna konfiguracija mreže i mrežnog podsustava.**

Moguće je definirati i druge kategorije poput:

- `ethers` - za razlučivanje *ETHERNET* vrsta/tipova mrežnih okvira (kako je vidljivo u: `/etc/ethertypes`).
- `networks` - za razlučivanje adresa/imena poznatih mreža (kako je vidljivo u: `/etc/networks`).
- `protocols` - za prepoznavanje mrežnih protokola (kako je vidljivo u: `/etc/protocols`).
- `services` - za razlučivanje TCP/IP servis: ***broj porta – ime*** (kako je vidljivo u: `/etc/services`).

Za provjeru, radi li željena kategorija koju smo konfigurirali, to možemo napraviti s naredbom `getent`.
Tako primjerice za provjeru kategorije `passwd:` to možemo napraviti upotrebom naredbe `getent` na sljedeći način:

```
getent passwd
```

Ako smo dobili ispis korisničkih računa, poput ispisa dolje, sve je u redu (ispis smo skratili na samo nekoliko redaka):

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

Što se resursa tiče, moguće je koristiti sljedeće resurse (navest ćemo ih samo nekoliko):

- `files` - označava lokalnu konfiguracijsku datoteku, koja ovisi o kategoriji.
- `nis` - označava mrežni ***NIS*** servis.
- `nisplus` - označava mrežni ***NIS*** plus (***NIS+***) servis.
- `dns` - označava mrežni ***DNS*** poslužitelj.
- `ldap` - označava mrežni ***LDAP*** (*Lightweight Directory Access Protocol*) poslužitelj.



Svi pronađeni podaci se zbog bržeg ponovnog dohvaćanja dodatno mogu spremati u međumemoriju (*cache*), upotrebom: za ***RedHat/Centos* 6.x**: `nscd` servisa (konfiguracijska datoteka: `/etc/nscd.conf`) odnosno za ***RedHat/Centos* 7/8.x** i novijih, upotrebom `sssd` servisa.

**Izvori informacija: (599),(600),(601),(768),(K-12),(K-14), `getent --usage`,`man 5 nsswitch.conf`.**

### 7.1.6.4. Datoteka `/etc/libuser.conf`

U datoteci `/etc/libuser.conf` konfiguriraju se osnovni parametri i opcije važni kod kreiranja i rada s novim korisnicima, vezanih za korisničke grupe (***GID***), identifikacijski broj korisnika (***UID***), način na koji će lozinka biti kriptirana, kao i drugi detalji. Opcije odnosno varijable i parametri definirani ovdje se primjenjuju na cijeli sustav i imaju najmanju težinu. Pogledajmo samo neke od njih kratko nabrojane uz malu napomenu kako se postavke postavljene ovdje koriste i za druge komponente sustava, a koje su definirane u datotekama koje smo spomenuli u prethodnim cjelinama odnosno datotekama: `/etc/login.defs` te `/etc/nsswitch.conf` i drugima. Opcije odnosno varijable koje se nalaze u datoteci: `/etc/libuser.conf` su zapisane u sljedećem formatu: **`opcija_ili_varijabla =`** `vrijednost ili parametar`
Navest ćemo samo njih nekoliko:

- **`login_defs =`** `/etc/login.defs` - ovdje se definira osnovna konfiguracijska datoteka za rad s korisnicima. Konkretno je ovdje definirana datoteka: `/etc/login.defs`.
- **`default_useradd =`** `/etc/default/useradd` – ovdje se definira datoteka s osnovnim varijablama za rad s korisnicima. U definiranoj datoteci jedna od varijabli je `SKEL` unutar koje je definiran direktorij `/etc/skel/` unutar kojeg su pohranjene datoteke koje se koriste prilikom kreiranja svakog korisničkog računa. Iz tog direktorija se prilikom kreiranja korisničkih računa kopiraju sve dostupne datoteke (`/etc/skel/*`) u kućni direktorij novog korisnika.
- **`crypt_style =`** `sha512` - ovdje se definira algoritam za kriptiranje lozinki (može biti i: `md5`,`sha256`,`des`,…)
- **`create_modules =`** `files shadow` - ovdje se definira lista modula koji se koriste za autentikaciju u slučajevima kada se mijenjaju ili kreiraju korisnik ili korisnička grupa. Konkretno je postavljena lista modula koji se koriste tijekom kreiranja ili rada s korisničkim računima ili grupama; prvo modul: `files` pa `shadow`.
- **`modules =`** `files shadow` - ovdje se definira lista modula koji se koriste za autentikaciju u slučajevima kada se **ne mijenjaju** korisnik ili korisnička grupa. Vidimo listu modula; prvo modul: `files` pa `shadow`.
- Sekcija **`[files]`**, **`directory`** `= /etc` - ovdje se definira vršni direktorij (`/etc`) u kojem se podrazumijevaju datoteke: `/etc/group` i `/etc/passwd`.
- Sekcija **`[shadow]`**, **`directory`** `= /etc` - ovdje se definira vršni direktorij (`/etc`) u kojem se podrazumijeva datoteka: `/etc/shadow`.

Opcije i parametri se nalaze unutar sekcija:
`[import]`, `[defaults]`, `[userdefaults]`, `[groupdefaults]`, `[files]`, `[shadow]`, …

**Izvori informacija: `man 5 libuser.conf`,`man login.defs`.**

---

### 7.1.6.5. Naredbe authconfig i authselect za rad s korisničkim postavkama

Sve do *RedHat/CentOS 8.x* sustava, za promjene postavki vezanih za korisničke račune bila je zadužena naredba `authconfig` koja se za svoj rad oslanjala na konfiguracijske datoteke poput: `/etc/sysconfig/authconfig`, za definiciju mehanizama autentikacije, kao i: `/etc/nsswitch.conf`, a ovdje definirana pravila je primjenjivala u datotekama vezanim za korisničke račune, poput datoteka: `/etc/passwd` i `/etc/shadow`, ali i vanjske servise koji su vezani za korisničke računa poput servisa koji su se konfigurirali u datotekama poput: `/etc/ldap.conf` i `/etc/openldap/ldap.conf` (za ***LDAP***), `/etc/samba/smb.conf` (za ***SMB/CIFS***) i druge.
Ako koristite ***RedHat/CentOS*** sve do inačice 8 ili neku drugu distribuciju Linuxa koja i dalje koristi navedenu naredbu (a to je dobar dio ili većina njih), pogledajmo nekoliko primjera njene upotrebe.

Dakle za ***RedHat/Centos do 7.x*** bi za promjenu postavki korisnički računa to bilo sljedeće:
Promijenimo provjerni zbroj (*hashing)* algoritam za zapisivanje lozinki korisnika u ***SHA512*** i snimimo promjenu:

```
authconfig --passalgo=sha512 –update
```

Postavimo minimalnu duljinu lozinke na sedam znakova uz potrebu da se koristi minimalno jedno malo slovo i jedan broj te snimimo promjenu:

```
authconfig --passminlen=7 --enablereqlower --enablereqdigit –update
```

Međutim s vremenom je ***RedHat*** pa sâmim time i sve druge distribucije koje su bazirane na njemu, od inačice ***8.x*** napustio ovakav načina rada jer je naredba `authconfig` u pozadini koristila cijeli nîz konfiguracijskih datoteka, a pogotovo zato što je sve promijene zapisivala u desetke gore navedenih datoteka, pa je kasnija analiza potencijalnih poteškoća u radu bila pomalo problematična (prema nekima). Stoga je uvedena nova naredba imena `authselect` koja modificira samo sljedeće datoteke:

- Datoteku: `/etc/nsswitch.conf`
- Datoteke u direktoriju: `/etc/pam.d/`
- Datoteke u direktoriju `/etc/dconf/db/distro.d/`
- Datoteke u direktoriju `/etc/authselect/` od kojih su neke izvorne za poveznice (*simboličke linkove*) unutar direktorija: `/etc/pam.d/`.

Za ***RedHat/CentOS 8.+*** to znači kako se ova naredba i cijeli sustav sada oslanjaju na ***PAM***, čija se konfiguracija, ovisno o podkomponenti sustava koji se konfigurira, nalazi u nekoj od datoteka unutar direktorija: `/etc/pam.d/`. Tako je recimo osnovna konfiguracijska datoteka `/etc/pam.d/system-auth` upravo ta koji sadrži postavke koje se iz nje čitaju a potom i postavljaju pri radu s korisničkim računima. Naime nova naredba imena `authselect` sada koristi samo Linux ***PAM*** (*engl. Pluggable Authentication Modules*) sustav i njegove pripadajuće module koji upravljaju zadacima autentifikacije korisnika i usluga na sustavu.

Ako pogledamo konfiguraciju sustava vezanu za korisničke grupe i korisnike `/etc/nsswitch.conf` vidjet ćemo sljedeće (sa sljedećom naredbom):

```
grep -i -e ^passwd -e ^shadow -e ^group /etc/nsswitch.conf

passwd:     sss files systemd
group:      sss files systemd
shadow:     files sss
```

Dakle vidimo kako se za korisničke račune (`passwd`) i grupe (`group`) prvo koristi `sss` (engl. *System Security Services Daemon*) koji se oslanja na ***PAM*** sustav te da se za spremanje provjernog zbroja (*hash*) lozinki koristi prvo datoteka jer je definirano `files` što se odnosi na datoteku: `/etc/shadow`, pa tek potom `sss`.

Pogledajmo nekoliko primjera upotrebe ove nove naredbe.
Pri tome je važno znati da novi sustav s `authselect` koristi profile i opcije. Stoga prvo pogledajmo neke od njih u tablici:

| *Authconfig* opcija | Prevodi se u *authselect* profil: | *Authconfig* opcija | Prevodi se u *authselect* opciju profila: |
|---|---|---|---|
| **--enableldap** **--enableldapauth** | **sssd** | *--enablesmartcard* | *with-smartcard* |
| **--enablesssd** **--enablesssdauth** | **sssd** | *--enableecryptfs* | *with-ecryptfs* |
| **--enablekrb5** | **sssd** | *--enablepamaccess* | *with-pamaccess* |
| **--enablewinbind** **--enablewinbindauth** | **winbind** | *--enablewinbindkrb* | *with-krb5* |
| **--enablenis** | **nis** | *--enablemkhomedir* | *with-mkhomedir* |

To konkretno znači sljedeće. Dati ćemo vam oba primjera: stara (***RedHat/CentOS do 7.x***) i nova naredba (***RedHat/CentOS od 8.x***). Pogledajmo primjer aktivacije *smart card* sustava.

Stara naredba (***RedHat/Centos do 7.x***) bi bila:

```
authconfig --enablesssd --enablesssdauth --enablesmartcard --smartcardmodule=sssd --updateall
```

Nova naredba (***RedHat/Centos od 8.+***) bi bila:

```
authselect select sssd with-smartcard
```

Za sve navedeno potrebno je da je *sssd* servis aktiviran i pokrenut. Ako slučajno nije, to možemo napraviti sa:

```
systemctl enable sssd.service
systemctl start sssd.service
```

Za ***RedHat/Centos od 8.x*** vezano za korisničke račune i njihovu konfiguraciju, sada imamo sljedeće konfiguracijske datoteke:

- Prvo imamo datoteku: `/etc/pam.d/passwd` koja obično pokazuje (*symbolic link*) na `/etc/pam.d/system-auth`. Potom imamo datoteku: `/etc/authselect/system-auth`, koja je izvorna datoteka na koju pokazuje i simbolička poveznica (*symbolic link*) datoteke✱: `/etc/pam.d/system-auth`. U njoj se definiraju pravila vezana za rad tijekom autentikacije na sustav. Primjerice vrsta algoritma za pohranu lozinke se definira u sljedećem retku✱:
  ```
  password  sufficient  pam_unix.so sha512 shadow nullok try_first_pass use_authtok
  ```
  Dakle vidimo da je odabran `sha512` algoritam koji će se koristiti kada korisnik upiše lozinku.
- Potom imamo datoteku: `/etc/authselect/password-auth`, koja je izvorna datoteka, na koju pokazuje simbolička poveznica (*symbolic link*) datoteke: `/etc/pam.d/password-auth`.
  U njoj se definiraju pravila vezana sa samu pohranu lozinke, poput:
  ```
  password  sufficient  pam_unix.so sha512 shadow nullok try_first_pass use_authtok
  ```
  I ovdje vidimo da se koristi `sha512` algoritam koji će se koristiti tijekom zapisivanja lozinke.

Navedene dvije datoteke, vezano za spremanje i rad s lozinkom (u redu koji počinje sa: `password`) imaju navedeno da koriste Linux/Unix ***PAM*** modul imena: `pam_unix.so`. Ovaj ***PAM*** modul je zadužen za rad s lozinkama.

Objasnit ćemo što je sve ovdje postavljeno, a tiče se ***PAM*** modula: `pam_unix.so`
  - `sha512` – znači da je odabrani algoritam SHA512 (na ovoj poziciji se i mijenja ovaj algoritam).
  - `shadow` – znači da se sve zapisuje u *shadow* datoteku✱: `/etc/shadow`.
  - `nullok` – znači da se dozvoljava prazna lozinka (ako ovo nije upisano tada se ne dozvoljava).
  - `try_first_pass` – znači da će se prvo provjeriti postoji li neka druga metoda (preko nekog drugog ***PAM*** modula) preko koje bi se moglo doći do lozinke, a tek onda provjeriti u *shadow*✱ datoteci.
  - `use_authtok` – znači da će se kod pokušaja promjene lozinke provjeriti zadovoljava li ona prema složenosti; za što je zadužen drugi *PAM* modul: `pam_cracklib`, ako se koristi.
- Imamo i još nekoliko datoteka za *authselect* sustav; koje koristi ***PAM*** sustav u pozadini.



Za osnove funkcioniranja ***PAM*** sustava pogledajte poglavlje:
**7.1.6.2. Datoteka /etc/login.defs.**

Za napredne sigurnosne postavke sustava pomoću *SELinux-a* pogledajte poglavlje:
**28.2. SELinux sustav.**

**Izvori informacija**: **(881)**,**(882)**, `man authselect`,`man authconfig`,`man pam_unix`, `man 5 system-auth`.

### 7.1.6.6. Linux PAM (Pluggable Authentication Modules)

Raniji operativni sustavi slični Unixu ili su pristupali samo lokalnim datotekama ili su imali kodirana pravila za pristup datotekama ili mrežnim bazama podataka o korisnicima, korisničkim grupama ili drugim podacima.
Primjerice ako ste koristili programe, kao što je su: `passwd`, `login` ili slične, potrebne za autentifikaciju korisnika, oni bi jednostavno pročitali potrebne informacije iz datoteka: `/etc/passwd` i `/etc/shadow`. Ako je još bilo potrebno i promijeniti korisničku lozinku, bilo je potrebno urediti datoteku `/etc/passwd`. Ova jednostavna, ali nespretna metoda unosila je brojne probleme za administratore sustava, ali ponajviše za programere navedednih aplikacija. Kako su *MD5* i *shadow* lozinke postale sve popularnije, svaki program koji je zahtijevao autentifikaciju korisnika morao je znati kako dobiti odgovarajuće informacije.
Potom je tvrtka *Sun Micorsystems*, razvila takozvani *Name Service Switch* (**NSS**) sustav koji povezuje računalo s raznim izvorima uobičajenih konfiguracijskih baza podataka i mehanizama za razrješenje imena. Ti izvori uključuju datoteke lokalnog operativnog sustava kao što su: `/etc/passwd`, (`/etc/shadow`) , `/etc/group` i `/etc/hosts`), te domenski sustav (***DNS***), sustav mrežnih informacijskih usluga (***NIS*** i ***NIS+***), imenični servis (***LDAP***) i druge.



Za detalje o načinu rada *Name Service Switch* (**NSS**) sustava, pogledajte poglavlje:
**7.1.6.3. Datoteka /etc/nsswitch.conf.**

**I dolazi PAM (Pluggable Authentication Modules)!**
Ako ste htjeli promijeniti svoju shemu provjere autentičnosti korisnika, sve prethodno navedene programe morali ste promijeniti te ponovno kompajlirati. ***PAM*** eliminira ovaj nered omogućavajući programima transparentnu provjeru autentičnosti korisnika, bez obzira na način na koji su podaci o korisniku pohranjeni.

Modularni sustav za provjeru autentičnosti odnosno engleski: *Pluggable Authentication Modules* (**PAM**) čine mehanizmi za integraciju više shema provjere autentičnosti niske razine prema sučelju za programiranje aplikacija visoke razine (***API***).
*PAM* omogućuje da se programi koji se oslanjaju na autentifikaciju pišu neovisno o temeljnoj shemi provjere autentičnosti.
Prva ga je predložila tvrtka *Sun Microsystems* tijekom 1995. godine. *PAM* se prvi put pojavio u ***Red Hat*** Linuxu 3.0.4 u kolovozu 1996. u Linux *PAM* projektu. *PAM* je trenutno podržan i u operativnim sustavima: ***AIX***, ***DragonFly BSD***, ***FreeBSD***, ***HP-UX***, ***Linux***, ***macOS***, ***NetBSD*** i ***Solaris***.
Budući da tada nije postojao centralni standard koji definira PAM, došlo je do kasnijeg pokušaja standardizacije PAM-a kao dijela procesa standardizacije ***X/Open UNIX***, što je rezultiralo standardom **X/Open Single Sign-on** (***XSSO***).
Međutim ovaj standard nije ratificiran, ali je nacrt standarda poslužio kao referentna točka za kasnije implementacije PAM-a (na primjer, *OpenPAM*-a). S time da je *OpenPAM* implementacija PAM-a s BSD licencom koju koriste ***FreeBSD***, ***NetBSD***, ***DragonFly BSD*** i ***macOS*** (počevši od *Snow Leoparda*), ali ne i Linux.
*Linux Pluggable Authentication Modules* (**PAM**) čini skup biblioteka koje administratoru Linux sustava omogućuje konfiguriranje metoda za provjeru autentičnosti korisnika. On pruža fleksibilan i centraliziran način za promjenu metoda provjere autentičnosti za zaštićene aplikacije korištenjem konfiguracijskih datoteka umjesto promjene sâmog programskog kôda aplikacije.
To konkretno znači da se programima (aplikacijama) pružaju biblioteke s funkcijama koje aplikacija može koristiti da zatraži provjeru autentičnosti korisnika. To isto tako znači da upotrebom ***PAM***-a nije važno je li vaša lozinka pohranjena u `/etc/shadow` ili na mrežni poslužitelj, upotrebom nekog od servisa za tu namjenu.
Dakle, kada program treba autenticirati korisnika, *PAM* mu pruža prohramsku biblioteku koja sadrži funkcije za ispravnu shemu provjere autentičnosti. Budući da se ova biblioteka učitava dinamički, promjena shema provjere autentičnosti može se izvršiti jednostavnim uređivanjem konfiguracijske datoteke. Konfiguracijske datoteke *PAM* sustava nalaze se u vršnom direktoriju (mapi): `/etc/pam.d/`, a unutar njega se nalaze konfiguracijske datote, po jedna za svaku važniju komponentu ili servis na sustavu. Sve dostupne ***PAM*** biblioteke se u pravilu nalaze unutar direktorija: `/usr/lib64/security/`.

**Za neke od primjera konfiguracije PAM-a pogledajte prethodna poglavlja!**

## System Security Services Daemon

*System Security Services Daemon* (**SSSD**) softver je izvorno razvijen za operativni sustav Linux, a koji pruža skup servisa za upravljanje pristupom udaljenim imeničnim servisima (pr. ***LDAP***, ***Active Directory***, ***DNS***) i mehanizmima provjere autentičnosti. Svrha SSSD-a je pojednostaviti administraciju sustava autenticiranog i ovlaštenog korisničkog pristupa koji uključuje više različitih sustava, a baziran je na Linux PAM-u. Njegova osnovna namjena je pružanju mogućnosti jednostruke prijave (engl. *Single sign on*) na lokalni sustav ili na (udaljene) mrežne servise.
Naime većina sustava provjere autentičnosti konfigurirana je lokalno (na računalu), što znači da je potrebno provjeriti lokalno pohranjene korisničke račune kako bi se odredila autentičnost korisnika. Ono što *SSSD* čini jest omogućavanje lokalnim servisima dohvačanje podataka iz lokalne predmemorije *SSSD* sustava. S time da se u *SSSD* predmemoriji podaci mogu preuzeti od bilo kojeg niza udaljenih pružatelja identiteta poput: ***LDAP***, ***Active Directory*** ili ***Kerberos*** sustava.
*SSSD* također sprema te korisnike i njihove vjerodajnice, tako da ako lokalni sustav ili davatelj identiteta nije dostupan, korisničke vjerodajnice su i dalje dostupne uslugama za provjeru iz *SSSD* predmemorije.
Možemo reći i ovako: ***SSSD*** je posrednik između lokalnih klijenata (pr. **PAM** ili **NSS** servisa) i bilo kojeg vanjskog sustava za pohranu podataka o identitetu korisnika.

Osim centralnog servisa (`sssd`), za pristup na udaljene sustave i mehanizme provjere autentičnosti *SSSD* se oslanja na nekoliko dodatnih servisa, od kojih ćemo spomenuti njih samo nekoliko:

- Za spajanje na ***LDAP*** servis, korist se servisom imena `sssd-ldap`.
- Za spajanje na ***Identity Management*** (***IdM*** ili ***IPA***), koristi se servisom imena: `sssd-ipa`.
- Za spajanje na ***Kereberos*** servis koristi se servisom imena: `sssd-krb5`.
- Za spajanje na ***Active Directory*** servis, koristi se servisom imena: `sssd-ad`.
- Za spajanje na lokalne datoteke (***passwd*** i ***group***), koristi se servisom imena: `sssd-files`.

Sve log datoteke *SSSD* sustava nalaze se u vršnom direktoriju: `/var/log/sssd/`. Centralna konfiguracijska datoteka *SSSD* sustava je: `/etc/sssd/sssd.conf`. *SSSD* sustav može i ne mora biti instaliran na sustavu.
Ako ga želite instalirati, to možete napraviti sa:

```
yum -y install sssd
```

**U konačnici se PAM, NSS i SSSD sustavi nadopunjuju!**
Pogledajmo samo dio datoteke *Name Service Switch* sustava odnosno datoteku: `/etc/nsswitch.conf`:

```
passwd:     sss files systemd
shadow:     files sss
group:      sss files systemd
hosts:      files dns
```

U konkretnoj konfiguraciji vidimo sljedeće:

- Za provjeru korisničkog imena (definirano s `passwd:`) koristit će se redom; prvo `sss` (*SSSD* sustav), pa tek onda `files` to jest datoteka: `/etc/passwd` i tek na kraju `systemd` servis.
- Za provjeru lozinke korisničkog računa (definirano sa `shadow:`) koristiti će se redom: `files` (datoteka: `/etc/shadow`) i tek na kraju `sss` (*SSSD* sustav).
- Za provjeru pripadnosti korisničkoj grupi (definirano s `group:`), koristit će se redom; prvo `sss` (*SSSD* sustav), pa tak onda `files` (datoteka: `/etc/group`) i tek na kraju `systemd`.
- Za razlučivanje imena računala u IP adrese (definirano s `hosts:`) koristiti će se redom: `files` (datoteka: `/etc/hosts`), tek potom `dns` odnosno definirani **DNS** poslužitelji koji su definirani u datoteci: `/etc/resolv.conf`.

***Zamislimo slijedeći scenarij: s našeg računala se spajamo na udaljeni SSH poslužitelj imena server1***
Ovdje se događa sljedeće:

1. Kao ***root*** korisnik pokrećemo spajanje sa *ssh* klijentom na ***server1***:
   ```
   ssh root@server1
   ```
2. Naš ***ssh*** klijent preko *Name Service Switch* sustava, prije nego išta pokuša, mora razlučiti IP adresu od imena poslužitelja (**server1**). Stoga na osnovu konfiguracije (`/etc/nsswitch.conf`) i definicije (`hosts`) prvo čita (`files`) i traži postoji li unos za ime računala **server1**, u lokalnoj datoteci (`/etc/hosts`), pomoću biblioteke `/lib64/libresolv.so`. U ovom slučaju ga pronalazi i saznaje da je računalu/serveru imena **server1** pripadajuća IP adresa 192.168.1.1 na koju se on potom i spaja. Da to nije uspio pokušao bi s **DNS** upitom.
3. Sada se na strani ***server1*** poslužitelja i njegovog ***SSH*** servisa (`sshd`), na osnovu njegove:
   a. *Name Service Switch* konfiguracije (`/etc/nsswitch.conf`) i sada definicije imena korisničkog računa (`passwd`) prvo čita (`sss`), koji konkretno provjerava svoju međumemoriju ili traži postoji li korisnik ***root*** u datoteci: `/etc/passwd`. Dakle konkretno se za to koristi biblioteka `/lib64/libnss_sss.so`. Recimo da korisnik postoji, dakle odgovor je pozitivan, pa se kreće dalje.
   b. Pošto je SSH servis ***PAM*** kompatibilan on preko ***PAM-a*** traži ima li za SSH servis (`sshd`) definiciju za *ssh* servis (`sshd`), u datoteci: `/etc/pam.d/sshd`. U konkretnom slučaju ona postoji. Zatim se ponovno provjerava; ali sada preko ***PAM*** sustava, postoji li korisnik ***root***.
   c. U slučaju da SSH servis nije ***PAM*** kompatibilan, prešlo bi se na klasičnu provjeru preko ***NSS*** sustava. Za provjeru korisničkog računa i lozinke tada bi se koristila biblioteka `/lib64/libnss_files.so`.
4. Sada se prema definiciji ***PAM***-a na poslužiteljskoj strani prema klijentu traži unos lozinke za korisnika ***root***.
5. SSH poslužitelj ponovno preko ***PAM***-a, kako je definrano u datoteci: `/etc/pam.d/sshd` provjerava lozinku, ili direktno ili preko *SSSD* servisa i njegove međumemorije. Ako je točna, odobrava spajanje na sustav.

***Koraci 2 – 4 ovise o inačici i sposobnostima servisa koji se koristi odnosno jesu li NSS i/ili PAM kompatibilni!***

*Navedeni primjer upotrebe spajanja sa SSH protokolom (za SSH klijenta), možemo analizirati sa sljedećom naredbom:*

```
strace ssh root@server1
```



Proučite i sljedeća poglavlja:
**15.2. Linux *strace* naredba.**
**25.8.6.2. SSH.**
**25.8.5. DNS protokol.**

**Izvori informacija: (1239),(1240),(1241),(1242),(1243),(1244),** **`man sssd.conf`,`man sssd-ldap`,`man sssd-ipa`, `man sssd-krb5`,`man sssd-ad`,`man sssd-files`,`man pam`,`man pam.conf`,`man pam_unix`.**

# 7.2. Menadžment softverskih paketa

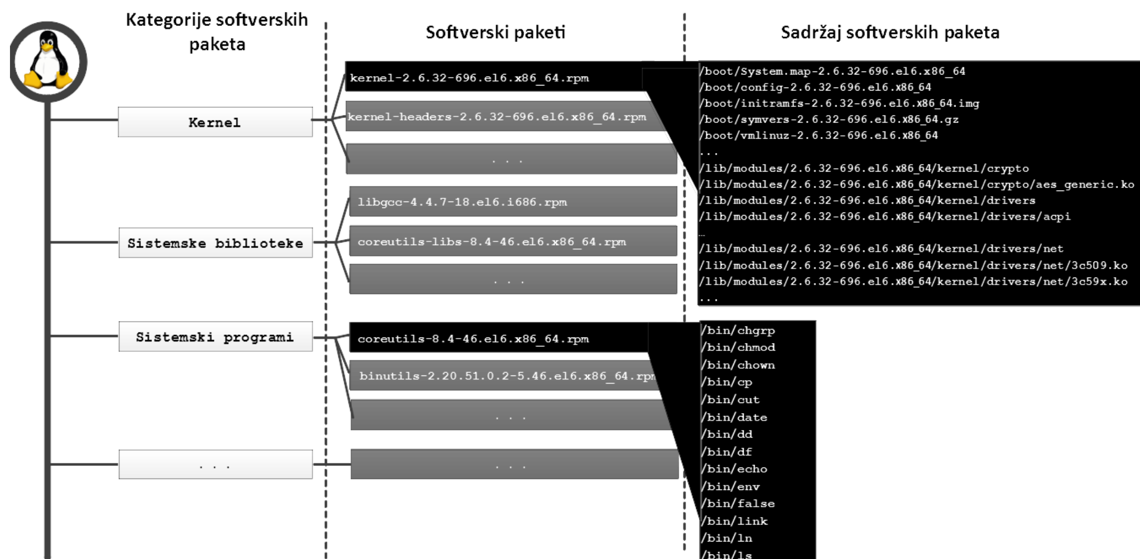
Softverski paketi u Linux terminologiji predstavljaju skup raznih programa (softvera) određene kategorije, koji se nalaze u jednom paketu, spremnom za instalaciju. Paket o kojem govorimo je u osnovi arhivska datoteka koja sadrži binarnu izvršnu datoteku odnosno sâm program/aplikaciju (ili više njih), njene konfiguracijske datoteke i biblioteke te informacije o međusobnim ovisnostima između paketa, odnosno programa i programskih biblioteka koje su programu potrebne za rad. Međutim zbog specifičnosti svake veće grane* distribucija Linuxa odnosno distribucija Linuxa koji su primjerice bazirani na ***Red Hat****, ***Debian****, ili nekoj drugoj vršnoj distribuciji, svatko od njih je razvio svoj menadžer softverskih paketa. Naravno osim menadžera softverskih paketa, morali su razviti i poseban format sâmih paketa; pr. **.rpm** format za ***RedHat***. Menadžeri softverskih paketa su razvijeni i stoga, kako bi se izbjegla potreba ručnog instaliranja, nadogradnje i deinstaliranja softvera.

U radu i primjerima koji slijede, mi ćemo koristiti *Red Hat* menadžer softverskih paketa odnosno *„Package management“* (**RPM**). On je zadužen za rad sa softverskim paketima našeg Linuxa. To znači kako pomoću njega: instaliramo, deinstaliramo i nadograđujemo softver. ***RPM*** format softverskih paketa koriste i druge distribucije Linuxa, bazirane na ***Red Hat***-u, poput:

- ***CentOS*** i **Rocky** Linuxa, kao i ***Fedora*** Linuxa te ***Oracle*** Linuxa, ali i nekih drugih distribucija Linuxa.
- ***SUSE*** i ***Open SUSE*** distribucija Linuxa.

Ekstenzija ***RPM*** paketa je `.rpm`. Pogledajmo logičku shemu distribucije softverskih paketa prema kategorijama odnosno logičkim cjelinama te njihovom sadržaju, kako je vidljivo na slici 23. dolje.

**Slika 23. Pogled na softverske pakete i kategorije softverskih paketa.**




Naime za svaku logičku cjelinu, poput cjelina: **kernel**, **sistemske biblioteke, sistemski programi** i slično, postoje pripadajući softverski paketi. Mi smo prikazali po nekoliko softverskih paketa iz nekoliko navedenih kategorija.
Tako je primjerice vidljiv paket (datoteka) u kojem se nalazi Linux kernel, konkretno u primjeru: `kernel-2.6.32-696.el6.x86_64.rpm`. U tom paketu se, kako je vidljivo u trećem stupcu (***Sadržaj softverskih paketa***) nalazi sâm linux kernel odnosno datoteka imena: `/boot/vmlinuz-2.6.32-696.el6.x86_64`, ali i konfiguracijska datoteka konkretnog kernela `/boot/config-2.6.32-696.el6.x86_64`. Dodatno, u istom paketu se nalaze i kernel moduli odnosno upravljački programi za ovaj kernel, kao i mnoge druge datoteke.
Ako pogledamo primjerice kategoriju sa sistemskim programima vidimo kako se u softverskom paketu imena: `coreutils-8.4-46.el6.x86_64.rpm` nalazi glavnina sistemskih programa odnosno osnovnih linux naredbi poput: `cat`, `chmod`, `chown`, `cp`, `cut`, `dd`, `df`, `du`, `echo`, `ln`, `mkdir`, `mv`, `rm`, `sort` i mnogih drugih, odnosno njihove pripadajuće datoteke. Dakle (*GNU*) ***coreutils*** softverski paket je jedan od vrlo važnih paketa programa na Linux sustavima.

Ako se vratimo jedan korak iznad (***Kategorije softverskih paketa***), gdje se nalaze sistemske biblioteke, možemo vidjeti kako se biblioteke potrebne za programe unutar paketa `coreutils`, u kojem se nalaze navedeni sistemski programi, nalaze u drugom paketu, imena `coreutils-libs-8.4-46.el6.x86_64.rpm` što je dosta čest slučaj. Dakle slučaj kada se biblioteke (engl. *library*) za neke programe distribuiraju u zasebnom softverskom paketu je vrlo čest, jer ih koriste odnosno na njih se oslanjaju mnogi programi s kojima se stoga te biblioteke uopće ne distribuiraju. Ovakve biblioteke se zovu dijeljene iz očitog razloga dijeljenja između više programa. Eventualni konflikt ili nedostatak određene biblioteke koju zahtjeva neki program je riješen tako da postoji lista ovisnosti o drugim paketima (engl. *Dependency*), kao i druge metode provjere potrebnih biblioteka.

Kategorije koje smo spominjali, su samo logičke, kako bismo razumjeli poveznicu između logičke kategorije, ***RPM*** paketa i njegovog sadržaja. Međutim postoji i konkretna razdioba ovih kategorija, koja je definirana u takozvanim ***grupama*** paketa. Prema tome moguće je instalirati cijelu ***grupu paketa*** (prema imenu grupe), a koja će onda u pozadini instalirati sve softverske pakete koji pripadaju toj grupi. Listu dostupnih kategorija (*grupa*) programa možete dobiti sa sljedećom naredbom:

```
yum group list
```

Za više detalja pogledajte sljedeća poglavlja!

**Izvori informacija: (1212),(K-12),(K-14), `man rpm`,`man yum`,`info coreutils`.**

### 7.2.1. RPM menadžer softverskih paketa

*RPM* menadžer softverskih paketa za Red Hat Linux odnosno engleski: „*package manager*" nam omogućava:

- Instalaciju softverskih paketa.
- Deinstalaciju softverskih paketa.
- Provjeru ispravnosti softverskih paketa odnosno provjeru i izračun provjernog zbroja (*checksuma*) paketa.
- Pretraživanje* softverskih paketa.
- Nadogradnju* softverskih paketa.

Najbliži ekvivalent Linux paketa u ***Windows*** svijetu bi bila instalacija nekog programa u formatu `.msi` ili `.exe` poput: `Setup.exe` ili `Setup.msi`. Mada Linux paketi osim ove osnovne funkcionalnosti koja je dostupna pod operativnim sustavom ***Windows***, nude mnogo više*. Softverski paketi se osim na instalacijskom mediju (pr. DVD-ROM) nalaze i na takozvanim *repozitorijima* odnosno web stranicama posebno pripremljenim za smještaj softverskih paketa.
Pri tome postoje sljedeće kategorije *repozitorija*:

- Standardni odnosno osnovni *repozitoriji* (engl. ***Base***), kao što je primjerice za 64. bitni **CentOS 6.x** *repozitorij* s osnovnim paketima, koji se nalazi na adresi: https://vault.centos.org/6.10/os/x86_64. Ovaj repozitorij je standardno uključen. Dok je primjerice za ***CentOS 7*** to: http://mirror.centos.org/centos/7/os/x86_64/Packages/ i tako dalje.
- Dodatni prošireni, poput: ***EXTRAS***, koji je također standardno uključen, a sadrži dodatne funkcionalnosti (pakete).
- Poseban sadržaj koji potencijalno mijenja bazni sustav: ***CentOSPlus***, koji je konfiguriran, ali standardno isključen.

S druge strane, postoje i dodatni repozitoriji, poput ***EPEL*** repozitorija za ***RedHat***/***CentOS***. Konkretno, ***EPEL*** repozitorij za ***RedHat*** bazirane Linuxe, inačice **7.x**, i to za 64.bitnu x86 platformu procesora, je primjerice dostupan na adresi: https://archives.fedoraproject.org/pub/archive/epel/7/x86_64/. Naime repozitorij je posebno pripremljen direktoriji na web poslužitelju koji sadrži softverske pakete, u obliku datoteka s ekstenzijom `.rpm` te ostale specifične datoteke. S repozitorija možete kopirati (napraviti *download*) željene dostupne softverske pakete. Podržani načini automatskog kopiranja, preko paketnog menadžera su preko: **HTTP** ili **FTP** protokola. Dodatno, svaki ***RPM*** paket može biti i kriptografski verificiran pomoću:

- **GPG** (*GNU Privacy Guard*) mehanizma (on je alternativa PGP mehanizmu).
- ***MD5*** provjeru provjernog zbroja (*checksuma*).

Vratimo se paketima i onome što se nalazi unutar svakog paketa. Dakle unutar svakog ***RPM*** paketa (datoteke) se nalaze:

- ***Zaglavlje (Header)*** koje sadrži informacije o samom paketu: inačica, arhitektura, opis, lista datoteka u paketu, …
- ***Podaci (Payload)*** odnosno arhiva sâmih datoteka (programi/aplikacije, konfiguracijske datoteke, biblioteke) obično u `cpio` formatu, koja je komprimirana sa **gzip**-om, a noviji formati podržavaju i druge kompresije ili podformate.
- ***Potpis (Signature)*** koji osigurava integritet paketa [koriste se ***GPG*** ili ***MD5***].

U trenutku kada radimo neku operaciju s paketima, određene informacije o svakom paketu se snimaju/čitaju u ili iz interne ***RPM*** baze podataka, na sâmom računalu. Navedena interna ***RPM*** baza podataka sadrži listu instaliranih paketa na našem sustavu. *RPM* bazu možemo pretraživati po raznim kriterijima te pomoću nje možemo provjeriti i inačice instaliranih paketa.

**Primjeri**

**1.** Ispišimo inačicu softverskog paketa imena `openssh-server` (to je *SSH servis*) pomoću naredbe `rpm`:

```
rpm -q openssh-server
```

**2.** Izlistajmo (`-q`) sve instalirane (`-a`) softverske pakete na našem sustavu (računalu):

```
rpm -qa
```

**3.** Upotrebom *regularnog izraza* `*` ispišimo sve instalirane pakete i pronađimo samo onaj koji u imenu ima riječ `telnet*` dakle riječ `telnet` iza koje može pisati bilo što:

```
rpm -qa "telnet*"
```

**4.** Pronađimo koji paket je instalirao točno određenu datoteku, s apsolutnom putanjom. Primjerice zanima nas datoteka imena `ssh` koja se nalazi u direktoriju `/usr/bin/ssh`. Konkretno nas zanima u kojem softverskom paketu ona izvorno dolazi:

```
rpm -qf /usr/bin/ssh

openssh-clients-5.3p1-84.1.el6.x86_64
```

Vidimo kako je pronađen softverski paket imena: `openssh-clients-5.3p1-84.1.el6.x86_64` u kojem je ona došla.

**5.** Izlistajmo sve datoteke koje su se instalirale sa softverskim paketom `coreutils` (prekidač `-ql`) :

```
rpm -ql coreutils
```

**6.** Ručno instalirajmo (`-i`) softverski paket imena `mc` (*Midnight Commander*) direktno s repozitorija, navodeći punu adresu (*URL*) do ***RPM*** paketa/datoteke, uz detaljan prikaz tijeka instalacije (`-vh`). To ćemo postići na sljedeći način:

```
rpm -ivh http://centos.t-2.net/6/os/x86_64/Packages/mc-4.7.0.2-6.el6.x86_64.rpm
```

U konkretnom primjeru nedostajat će nam paketi o kojima ovisi instalacija ovog paketa odnosno takozvane ovisnosti (engl. *Dependencies*) te se paket stoga neće instalirati. Ovaj primjer ćemo stoga ponoviti u slijedećem poglavlju, s drugom naredbom.

**7.** Provjerimo kojoj *RPM* grupi/kategoriji softverskih paketa pripada softverski paket imena: `kernel`

```
rpm -q --qf %{group} kernel
```

**7.1.** Ispišimo sve grupe *RPM* paketa, koje imamo instalirane na sustavu, te ih sortirajmo po abecedi s naredbom `sort`:

```
rpm -qa --qf '%{group}\n' | sort -u
```

*! Gore navedeni repozitoriji odnosno njihove adrese se mogu s vremenom mijenjati !*

Pogledajmo koji su osnovni prekidači naredbe `rpm`:

- `-i` *IME* – instaliraj paket naziva *IME*, pri tome (*IME)* može biti i URL adresa: ***FTP*** ili ***HTTP*** (*engl. Install*).
- `-U` *IME* – nadogradi paket naziva *IME* (*engl. Upgrade*).
- `-e` *IME* – obriši ili deinstaliraj paket naziva *IME* (engl. *Erase*).
- `-V` *IME* – provjeri paket naziva *IME* (*engl. Verify*). Pri tome se provjerava:
    - Vlasnik *(engl. Owner*).
    - Vlasnička grupa *(engl. Group*).
    - *Ovlasti (engl. Permisons/Modes)* (*r/w/x*).
    - Provjerni zbroj *MD5 (engl. Checksum*).
    - *Veličina (engl. Size).*
    - Inačice; veća i manja: v. **X.y** (*engl. Major and Minor Version number).*
    - Simbolički linkovi/poveznice (*engl. Symlink*).
    - Vrijeme promjene *(engl. Modification time).*

Dodatni korisni prekidači su:

- `-h` – ispis napretka tijeka instalacije, sa znakovima # (*engl. Hash marks - #)*
- `-v` – detaljniji ispis tijeka instalacije ili neke druge operacije (*engl. Verbose*).

Korisni prekidači za pretraživanje (malo slovo **Q**):

- `-qi` IME – ispiši opis paketa (ovdje je i ime grupe kojoj pripada).
- `-qg` IME – ispiši listu paketa koji pripadaju određenoj grupi.
  Primjerice: `rpm -qg "System Environment/Daemons"`
- `-ql` IME – ispiši listu datoteka unutar paketa. Primjerice: `rpm -ql openssh-clients`
- `-qlp` IME *RPM* paketa – ispiši listu datoteka koje se nalaze u konkretnom RPM softverskom paketu. Paket može biti na lokalnom disku pa je potrebno upisati cijelu putanju do njega ili primjerice na udaljenom repozitoriju, poput:

```
rpm -qlp http://centos.t-2.net/6/os/x86_64/Packages/mc-4.7.0.2-6.el6.x86_64.rpm
```

- `-qc` IME – ispiši konfiguracijske datoteke određenog paketa.
- `-qs` IME – ispiši stanje datoteka unutar paketa, poruke mogu biti sljedeće:
    - *Normal* – datoteka je instalirana.
    - *Not installed* – datoteka iz paketa nije instalirana.
    - *Replaced* – datoteka je zamijenjena.

I na kraju je važno znati nešto i o nedostacima `rpm` menadžera paketa. Najveći nedostatak ***RPM***-a jest činjenica da on ne može riješiti ovisnosti jednog paketa o nekom drugom paketu (engl. *dependencies*) odnosno ovisnost određenog programa unutar paketa o nekom drugom ili o nekoj biblioteci (*Library*) koja mu je potrebna za rad.
Jedino što `rpm` (program) može napraviti po pitanju ovisnosti o nekom drugom programskom paketu, jest prije instalacije provjeriti je li taj paket instaliran i upozoriti vas kako vam nedostaju određeni paketi, ali on ih za vas neće moći instalirati.
To morate sami … mada i za to postoji rješenje koje ćemo opisati u sljedećem poglavlju, a zove se ***YUM***.

**Izvori informacija: (K-12),(K-14), `man rpm`.**

## 7.2.2. *YUM* menadžer softverskih paketa

***YUM*** (engl. *Yellow dog Update Modified*) nam daje nadogradnju na mogućnosti *RPM* menadžera paketa: `rpm`. Možemo reći i da je ***YUM*** dodatno sučelje više razine za *RPM* menadžer paketa. ***YUM*** omogućava praćenje ovisnosti jednog paketa o drugim paketima, a on ih automatski i rješava. Naime, on međusobne ovisnosti softverskih paketa rješava uvođenjem dodatka na *RPM* repozitorij u obliku meta podataka, koji sadrže dodatnu bazu podataka koja pohranjuje informacije o svakom paketu te svim njihovim međuovisnostima; primjerice: za paket X, potrebno je prvo instalirati paket Y i slično.


Svaki puta kada pokrenete program **yum**, on se spaja na sve one repozitorije (na internetu), koji su vam dodani odnosno konfigurirani na sustavu.

Nakon pokretanja, program `yum` povlači sve *YUM* baze podataka s definiranih repozitorija, na vaše računalo. Na taj način u svakom trenutku, usporedbom instaliranih paketa i onih koji su u *YUM* bazi podataka, *YUM* zna je li se pojavila novija inačica nekog programa, uz njegovu osnovnu funkcionalnost: da su mu poznate međuovisnosti svih tih softverskih paketa.

### Kako se instaliraju paketi pomoću YUM-a i što se događa u pozadini

Procedura pri instalaciji softverskih paketa je sljedeća:
**1.** Po potrebi (nije nužno); dodajete novi *repozitorij*, ako nije dodan, a baš na njemu se nalazi paket koji trebate. Ako je *repozitorij* već dodan ili se program koji želite instalirati nalazi na standardnim *repozitorijima* (onima koji su već definirani), onda preskačete ovaj korak.
**2.** Tijekom svakog pokretanja ***YUM***-a, sinkroniziraju se metapodaci s *Web repozitorija* na vaše (lokalno) računalo.
**3.** *YUM* kopira *RPM* pakete te provjerava njihove ovisnosti. Ako ovisnosti postoje, prvo se kopiraju i instaliraju svi softverski paketi koji su prethodno potrebni, a tek onda oni koji slijede nakon njih.

**Izvor informacija: (K-12), `man yum`.**

### 7.2.2.1. Rad s *YUM*-om

Naredba `yum` se koristi za napredni, a sada standardni, rad sa softverskim paketima. Njene osnovne opcije i prekidači su:

- `yum update` – napravi nadogradnju **apsolutno svih instaliranih** softverskih paketa (engl. *Full system update*).
- `yum install` *IME* – instaliraj novi softverski paket imena (*IME*).
- `yum info` *IME* – zatražite informacije o softverskom paketu imena (*IME*) [pr. inačicu, veličinu, detaljniji opis,…].
- `yum erase` *IME* – deinstaliraj softverski paket imena (*IME*).
- `yum search` *IME* – pretraži sve softverske pakete u potrazi za ključnom riječi (*IME*).
- `yum list` – ispiši popis softverskih paketa; kako instaliranih, tako i onih dostupnih za instalaciju na svim repozitorijima koji su aktivni na našem računalu:
  - `yum list installed` – ispiši samo popis softverskih paketa koji su instalirani na našem računalu
- `yum repolist` – ispiši listu svih aktivnih repozitorija koje koristimo na našem računalu.
- `yum whatprovides needs-restarting` – ispiše sve softverske pakete koji traže restart sustava.
- `yum grouplist` – ispiše popis svih instaliranih grupa paketa, primjerice ispišimo sve grupe softverskih paketa, sa sljedećom naredbom:
  ```
  yum grouplist
  ```
- `yum groupinstall` *IME* – instaliraj grupu paketa naziva *IME*, pa bi tako ono naziva: `Development Tools` instalirali na sljedeći način:
  ```
  yum groupinstall "Development Tools"
  ```
  - `yum groupremove` *IME* – deinstaliraj grupu paketa naziva *IME*, primjerice:
    ```
    yum groupremove "Development Tools"
    ```

**Kako nadograditi cijeli sustav i apsolutno sve instalirane softverske pakete na sustavu, pomoću programa/naredbe** `yum`
```
yum update -y
```

**Primjeri dodavanja novog repozitorija**
Kako dodati novi *repozitorij* na sustav; primjerice ***EPEL*** 64 bitni *repozitorij*.
Prvo moramo u web pregledniku otići na **EPEL** *repozitorij* na adresi:
https://archives.fedoraproject.org/pub/archive/epel/6/x86_64/ te pronaći paket, imena poput `epel-release-6-8.noarch.rpm`. Naime na većini *repozitorija* postoji *RPM* paket, koji kada instaliramo, konkretni *repozitorij* se automatski instalira odnosno nadoda u postojeću grupu *repozitorija*, te dodaje njegov **GPG ključ** koji će se ubuduće koristiti, kod instalacije novih softverskih paketa. **GPG je zadužen za provjeru autentičnosti i verifikaciju integriteta paketa.**
*Točna adresa (URL), kao i ime gore navedene datoteke se s vremenom može mijenjati (ovisno o inačici).*

U novije vrijeme, neki od priznatijih *repozitorija*, upravo poput ***EPEL***-a, već imaju svoj softverski paket na standardnim *repozitorijima*, pa ga najjednostavnije možemo dodati (*instalirati*) na sljedeći način (za ***CentOS*** **6**.x/**7**.x/**8**.x):
```
yum -y install epel-release
```

**Metoda 1 - ručno: poluautomatski; primjer (URL) je za CentOS 6.x**
Korištenjem naredbe `wget` koja će napraviti download tog *RPM* paketa (datoteke) koju ćemo instalirati, pomoću *RPM* menadžera paketa, odnosno programa `rpm` u drugom koraku (drugi redak):
```
wget https://archives.fedoraproject.org/pub/archive/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm
```

**Metoda 2 - ručno: automatski; primjer (URL) je za CentOS 6.x**
Korištenjem *RPM* menadžera paketa koji ionako podržava *HTTP* i *FTP* protokole za dohvaćanje datoteka direktno s mreže:
```
rpm -ivh https://archives.fedoraproject.org/pub/archive/epel/6/x86_64/epel-release-6-8.noarch.rpm
```

Provjerimo je li novi ***epel*** *repozitorij* sada dodan (treba se pojaviti `epel` u listi repozitorija):
```
yum repolist

base                    CentOS Linux 6 - BaseOS
epel                    Extra Packages for Enterprise Linux 6 - x86_64
extras                  CentOS Linux 6 - Extras
```
Vidimo kako je `epel` izlistan te da je napravljeno dohvaćanje podataka s njega, dakle to je u redu. U slučaju kada **`epel`** ili neki drugi *repozitorij* nije uredno dodan u listu aktivnih *repozitorija*, možemo ga ručno uključiti sa sljedećom naredbom:
```
yum --enablerepo=epel
```

Isto tako, moguće je i isključiti pojedini *repozitorij*, sa sličnom naredbom: `--disablerepo=XX` (`XX` je ime *repozitorija* koji isključujemo). Sada instalirajmo program *Midnight Commander* (*Norton/Total Commander* klon za Linux), ime paketa je `mc`
```
yum install mc
```

Zatim, za probu, deinstalirajmo program *Midnight Commande*r koji smo upravo instalirali (ime softverskog paketa je `mc`):
```
yum erase mc
```

Potom ispišimo sve instalirane grupe softverskih paketa, ali i sve dostupne preko svih *repozitorija*, koji su aktivirani na računalu:
```
yum grouplist
```

S gore navedenom naredbom, dobit ćemo listu skupina softverskih paketa (skratili smo ju zbog jednostavnijeg prikaza):

```
Installed Groups:                          Available Groups:
   E-mail server                               Additional Development
   Legacy UNIX compatibility                   Backup Client
   Networking Tools                            Base
   Security Tools                              CIFS file server
   Storage Availability Tools                  Client management tools
   System administration tools                 Compatibility libraries
   Web Server          . . .                   Console internet tools    . . .
```

**Izvor informacija: (K-12),(K-14),** **`man yum`**.

---

### 7.2.2.2. Napredna konfiguracija *YUM*-a

***Slijedi napredna cjelina!***

Konfiguracija *YUM*-a se nalazi u nekoliko osnovnih kategorija:

- Globalna konfiguracija je zapisana u konfiguracijskoj datoteci `/etc/yum.conf`
- Opcionalne konfiguracije, koje se nalaze u direktoriju (mapi) `/etc/yum/`
- Konfiguracije svakog zasebnog repozitorija, koje se nalaze unutar direktorija (mape) `/etc/yum.repos.d/`

## Datoteka /etc/yum.conf

Konfiguracijska datoteka: `/etc/yum.conf` sadrži globalne postavke programa `yum`.
Ova datoteka mora sadržavati minimalno jednu sekciju : `[main]`, koju ćemo upravo pogledati.
Pogledajmo osnovni primjer ove konfiguracijske datoteke (u kojoj u većini slučajeva nećete ništa morati mijenjati):

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=5
bugtracker_url=http://bugs.centos.org/set_project.php?project_id=19&ref=http://bugs.c
entos.org/bug_report_page.php?category=yum
distroverpkg=centos-release
```

Upoznat ćemo se s nekoliko osnovnih parametara odnosno opcija koje vidimo ovdje:

- `cachedir=` – označava direktorij u koji će *YUM* spremati privremene podatke, ali i držati svoju bazu podataka s listom softverskih paketa. Za ovu namjenu, mogu se koristiti i posebne varijable, poput:
  - `$basearch` – je varijabla koja označava baznu arhitekturu sustava:
    - `i686` ili `i386` – označavaju 32 bitnu inačicu (verziju).
    - `x86_64` – označava 64 bitnu inačicu (verziju).
  - `$releasever` – je varijabla koja označava glavnu inačicu, odnosno generaciju ***RedHat***/***CentOS*** Linuxa
- `keepcache=` – definira svojstvo čuvanja privremenih podataka odnosno meta podataka i softverskih paketa, nakon što je svaki pojedini softverski paket instaliran:
  - `0` – znači kako se trebaju brisati nakon instalacije.
  - `1` – znači kako trebaju ostati na disku i nakon instalacije.
- `debuglevel=` – definira koliko detaljno će se ispisivati podaci u radu sa softverskim podacima; vrijednost `2` je standardni, `1` je minimalno, a `10` je maksimalno.
- `logfile=` – datoteka u koju će se logirati poruke o radu sa softverskim paketima; koji je instaliran, deinstaliran i slično.
- `exactarch=` – znači hoće li `yum` provjeravati točnu arhitekturu kod nadogradnje softverskih paketa, pri tome je `1` standardno da se provjerava, a `0` bi bilo da se ne provjerava (nije preporučeno).
- `obsoletes=` – označava hoće li paket koji je označen da nasljeđuje neki drugi paket (odnosno kako je taj drugi *engl. Obsolete*), onda će ovaj noviji pregaziti onaj koji se nasljeđuje. Standardno je ova opcija uključena (`1`).
- `gpgcheck=` – ovo je opcija koja traži ***GPG*** *(engl. GNU Privacy Guard)* koji je zamjena za ***PGP***, provjeru autentičnosti softverskih paketa, pomoću elektroničkog sustava potpisivanja, prije nego se paketi uopće mogu/smiju instalirati. Standardno je ova opcija uključena (`1`).
  Vezano za **GPG** pogledajte poglavlje: **7.2.5. GNU Privacy Guard (GPG) i repozitoriji** .
- `plugins=` – opcija koja omogućava (vrijednost `1`) upotrebu takozvanih *plugina*, koji mogu nadograditi određene funkcionalnosti `yum`-a. Standardno je ova opcija uključena (`1`).
- `installonly_limit=` – opcija je koja govori koliko različitih inačica softverskih paketa smijemo istovremeno imati na sustavu. Ovo se posebice odnosi na instalirane inačice kernela, kojih obično imamo više, jer želimo mogućnost pokretanja sustava i sa nešto starijim kernelom, Standardna vrijednost je `5`. Ostale opcije nisu toliko važne.

# Direktorij /etc/yum.repos.d/

Unutar direktorija `/etc/yum.repos.d/` nalaze se konfiguracije svakog pojedinog repozitorija. Ove konfiguracijske datoteke mogu sadržavati i opcije koje su već navedene u globalnoj konfiguracijskoj datoteci `/etc/yum.conf` te će u konačnici, ako postoje opcije definirane ovdje, pregaziti one već definirane u datoteci `/etc/yum.conf`.

Naime svaki puta kada dodajemo novi repozitorij, pomoću metode 1 ili 2, opisane prije dvije stranice, zapravo se za svaki pojedini repozitorij, kreira po jedna datoteka, unutar direktorija: `/etc/yum.repos.d/` s imenom koje mora imati ekstenziju: `.repo`.

Tako primjerice za standardni osnovni repozitorij (***BASE***) za *CentOS* linux, postoji datoteka imena: `/etc/yum.repos.d/CentOS-Base.repo`. Važno je razumjeti kako i većina repozitorija ima i nekoliko pôd kategorija koje možemo promatrati kao pôd repozitorije, koji pak sadrže drugačije kategorije softverskih paketa, pa tako mogu postojati neke od kategorija poput:

- Standardne kategorije – obično nosi osnovno ime repozitorija (sadrži standardne softverske pakete).
- ***Debug*** kategorija – s alatima/programima za debugiranje.
- ***Updates*** kategorija – sa softverskim paketima za nadogradnu sustava (kod nadogradnje primjerice, sa CentOS v.6.8 na v.6.9).
- ***Source*** kategorija – s izvornim kôdom programa iz svih gornjih kategorija.

Datoteka u kojoj se definira repozitorij mora imati minimalno jednu sekciju s nekoliko opcija unutar nje, poput:

```
[repozitorij]
name=repository_name
baseurl=repository_url
```

- `[repozitorij]` je sekcija koja označava kategoriju unutar repozitorija, minimalno mora postojati samo jedna.
- `name=` je naziv ove kategorije ili repozitorija.
- `baseurl=` je web adresa (URL) tog konkretnog repozitorija, pri čemu su moguće tri mogućnosti:
  - Za ***HTTP/S*** web adresu: adresa mora biti u formatu: `http://putanja/do/repozitorija`



Pri upotrebi ***HTTPS*** protokola, *URL* (jedinstvena adresa) obično izgleda poput:
**`baseurl=`**`https://putanja/do/repozitorija/releases/$releasever/server/$basearch/os/`

  - Za ***FTP*** web adresu: adresa mora biti u formatu: `ftp://putanja/do/repozitorija`
  - Za lokalni repozitorij (***na lokalnom disku***), što je isto moguće. Pri tome adresa mora biti u formatu: `file:///putanja/do/lokalnog/repozitorija`

Postoje i dodatne opcije koje se također često koriste, a mi ćemo spomenuti njih samo nekoliko:

- `enabled=` ovo označava je li ovaj repozitorij omogućen. Standardno je vrijednost `1`, dok isključen ima vrijednost `0`
- `gpgcheck=` ovdje se definira, koristi li se provjera autentičnosti pomoću *GPG* algoritma, standardno je uključena (`1`) te obično sadrži i ključ:
  - `gpgkey=` je lokacija *GPG* javnog ključa, na lokalnom disku. Ovaj ključ se kopira prvi puta, sa repozitorijskog poslužitelja, a obično izgleda poput: `gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6`.
    Za detalje o **GPG** pogledajte poglavlje: **7.2.5. GNU Privacy Guard (GPG) i repozitoriji** .
- `mirrorlist=` URL – ovdje se definira lista kopija odnosno poslužitelja koji su kopije izvornog repozitorija, što se često koristi. Ovo je korisno jer u slučaju nedostupnosti jednog repozitorijskog poslužitelja, sustav traži poslužitelje koji su njegove *zrcalne* kopije (*engl. Mirror*).
  - `failovermethod=` - ovo je metoda prema kojoj će sustav, u slučaju kada ne može dohvatiti softverski paket s prvog repozitorijskog poslužitelja, pokušavati tražiti i na drugim „*mirror*" poslužiteljima. Standardno je postavljeno na: `priority` što znači kako se već nakon prvog neuspješnog pokušaja dohvaćanja softverskog paketa s primarnog repozitorijskog poslužitelja, odmah prebacuje na drugi u listi poslužitelja koji su zrcalne kopije (*mirror*).

Sada pogledajmo kako izgleda naša datoteka: `/etc/yum.repos.d/CentOS-Base.repo`

```
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os&infra=$infra
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
#released updates
[updates]
name=CentOS-$releasever - Updates
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=updates&infra=$infra
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
```



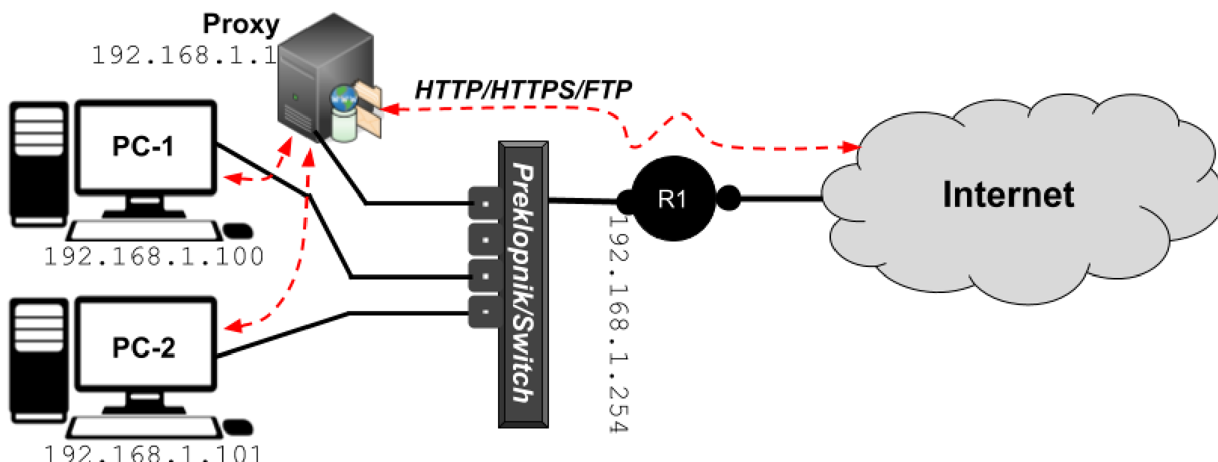
Novije inačice **RedHat/CentOS** (v.**8**.x) uvele su takozvani ***DNF*** menadžer softversih paketa, kao zamjenu za ***YUM***, iako je ***YUM*** na njima i dalje aktivan (kao naredba). ***DNF*** menadžer koristi istoimenu naredbu `dnf`.

**Izvori informacija: (287),(288),(K-14), `man rpm`, `man yum`, `man dnf`, `man 5 yum.conf`.**

### 7.2.2.3. Upotreba posredničkog (*Proxy*) poslužitelja

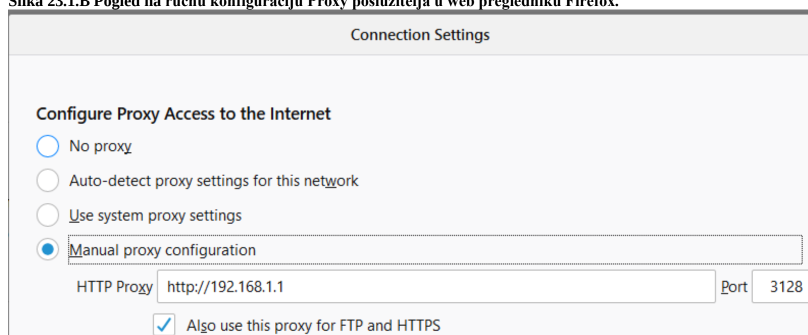
***Slijedi napredna cjelina!.*** U računalnim mrežama, proxy poslužitelj je poslužiteljska aplikacija ili uređaj koji djeluje kao posrednik za zahtjeve klijenata koji traže resurse od poslužitelja koji pružaju te resurse. ***Proxy*** odnosno posrednički poslužitelj radi u ime klijenta kada zahtijeva uslugu, potencijalno prikrivajući istinsko podrijetlo zahtjeva poslužitelju resursa. Umjesto da se izravno poveže s poslužiteljem koji može ispuniti traženi resurs, poput datoteke ili web stranice, klijent usmjerava zahtjev na proxy poslužitelj, koji provjerava zahtjev te izvodi potrebne mrežne transakcije. To služi kao metoda za pojednostavljivanje, kontrolu, provjeru prava pristupa ili pružanje dodatnih pogodnosti poput uravnoteženja opterećenja, privatnosti kao i dodatne sigurnosti. Proxy poslužitelji (pr. [SQUID proxy]) nude i napredne metode filtriranja web stranica (pr. uz [squidGuard]), potencijalno prema nepoželjnom sadržaju i drugome, kao i mogućnost spremanja sadržaja web stranica u privremenu memoriju (*cache*), iz koje se sadržaj može dohvatiti sljedeći puta kada netko pristupa istoj web stranici. Pohrana jednom otvorenih web stranica ubrzava rad i rasterećuje vezu prema internetu. Sve navedeno u praksi znači da se računala, poput ovih na slici 23.1.A, umjesto direktnog spajanja na web stranice na internetu, spajaju na proxy poslužitelj, koji se u njihovo ime spaja na tražene web stranice. U konfiguraciji na slici je vidljivo da oba računala imaju konfiguriran podrazumijevani usmjerivač (***default gateway***) s IP adresom: ***192.168.1.254***, preko koje bi inače imali izlaz na Internet. Međutim u ovakvim mrežama se zabranjuje izlaz na Internet za pr. *HTTP/HTTPS/FTP* i slične protokole, za sve IP adrese osim IP adrese Proxy poslužitelja. Stoga su ova računala konfigurirana da za pristup web sadržaju (web stranicama) koriste ***proxy*** poslužitelj na IP adresi: ***192.168.1.1***.

**Slika 23.1.A Pogled na Proxy poslužitelj u mreži.**




Pogledajmo ručnu konfiguraciju proxy poslužitelja u web pregledniku ***Firefox*** (slika 23.1.B) {*OPTIONS→ GENERAL→ Network Settings*}:

**Slika 23.1.B Pogled na ručnu konfiguraciju Proxy poslužitelja u web pregledniku Firefox.**




U Linuxu se postavke ***proxy*** poslužitelja mogu definirati na globalnoj razini; za sve aplikacije na sustavu, što uključuje i menadžer softverskih paketa (**YUM** ili **DNF**), i to definiranjem *proxy* postavki u globalnoj datoteci: `/etc/profile` ili u datotekama specifičnim za svakog pojedinog korisnika. Ovdje zapravo definiramo globalne *proxy* varijable na koje se oslanjaju mnogi programi i komponente sustava. Mi ćemo koristiti navedenu datoteku u koju ćemo dodati sljedeće retke:

```
export http_proxy=http://192.168.1.1:3128
export https_proxy=http://192.168.1.1:3128
export ftp_proxy=http://192.168.1.1:3128
```

U slučaju kada proxy poslužitelj zahtjeva i autentikaciju korisnika, tada je potrebno definirati ove varijable u postavkama korisnika (pr. u datotekama korisnika: `.bashrc` ili `.bash_profile`), na sljedeći način (opisujemo samo jednu od njih):

```
export http_proxy=http://hrvoje:lozinka@192.168.1.1:3128
```

Pri tome je `hrvoje` ime korisničkog računa a `lozinka` je njegova lozinka, a **`192.168.1.1:3128`** su IP adresa i port (`3128`) proxy poslužitelja. **U slučaju kada proxy poslužitelj želimo ili moramo koristiti samo za menadžer softverskih paketa (*YUM*), tada otvorimo datoteku: `/etc/yum.conf`, te unutar sekcije `[main]` dodajmo sljedeće retke konfiguracije:**

```
proxy=http://192.168.1.1:3128
proxy_username=hrvoje
proxy_password=lozinka
```

Za detalje oko konfiguracije i rada **Proxy** servisa pogledajte poglavlje: **26.4.5. Proxy poslužitelji.**

**Izvori informacija: (949),(950),(951),(K-12),(K-14), `man 5 yum.conf`.**

### 7.2.3. Rad s DNF-om

U nekim od distribucija Linuxa baziranim na ***Red Hat*** Linuxu, paketni menadžer ***YUM*** zamijenjen je s paketnim menadžerom naziva ***DNF*** jer su mnogi dugotrajni problemi u *Yumu* ostali neriješeni. Naziv ***DNF*** došao je iz engleskog jezika, a znači „*Dandified YUM*“ to jest uljepšani ***YUM***. Međutim, u ***Red Hat v.8.x. YUM*** je i dalje u upotrebi, ali u inačici četiri (YUM v4.) koja je bazirana na ***DNFu***. Problemi koje ***DNF*** rješava uključuju slabu izvedbu, pretjerano korištenje RAM memorije, usporavanje tijekom rješavanja paketne ovisnosti i druge nedostatke. **DNF** koristi takozvanu ***libsolv*** programsku biblioteku za rješavanje ovisnosti među paketima, koju je razvio i održava *SUSE*, radi poboljšanja performansi, a sâm je primarno razvijen u programskim jezicima ***C*** i ***C++*** te su samo neki dijelovi pisani u jeziku *Python*. S druge strane ***YUM*** je napisan uglavnom u programskom jeziku *Pythonu* i ima svoj način rješavanja ovisnosti među paketima, koji nije optimalan, te za sobom povlači ograničenja ovog programskog jezika, po pitanju performansi. Osim toga, njegov ***API*** nije u potpunosti dokumentiran, a njegov sustav proširenja dopušta samo *Python* dodatke. Kako što smo već naveli, u ***DNF*** su dodana i mnoga poboljšanja, od kojih ćemo nabrojati samo neke:

- **API** je vrlo dobro dokumentiran, te je jednostavno razvijati nove funkcionalnosti i ekstenzije (ne samo u *Pythonu*).
- Programski kôd ***DNFa*** je gotovo dva puta manji (29.000 prema 59.000 linija kôda) te je optimiziran, a samim time zahtjeva manje resursa računala u radu. Osim toga manji i čitljiviji programski kôd je lakše održavati.
- U slučaju nedostupnosti repozitorija, prelazi se na prvi sljedeći u dohvaćanju sadržaja.
- Razlučivanje međusobne ovisnosti softverskih paketa je brže i pouzdanije.

U konačnici i ***YUM*** i ***DNF*** se koriste za upravljanje paketima na sustavu koji se temelji na ***rpm*** paketima (koji koriste distribucije Linuxa poput: **Red Hat**, ***CentOS***, ***Rocky***, ***Fedora***, ***Oracle***), uključujući instalaciju, nadogradnju, pretraživanje i uklanjanje softverskih paketa. Naredba s kojom ga pozivamo je `dnf`, a njena sintaksa je kompatibilna sa sintaksu naredbe `yum`.
Pogledajmo listu nekih od opcija unutar naredbe `dnf`:

- `dnf update` - napravi nadogradnju **apsolutno svih instaliranih** softverskih paketa (engl. *Full system update*).
    - `dnf update` *IME* - napravi nadogradnju samo softverskog paketa imena (*IME*).
- `dnf install` *IME* - instaliraj novi softverski paket imena (*IME*).
- `dnf reinstall` *IME* - reinstaliraj novi softverski paket imena (*IME*).
- `dnf upgrade-to` *IME* - nadogradi softverski paket imena (*IME*), na točnu inačicu, primjerice:
  **`dnf upgrade-to httpd-2.4.27-3.fc26`**
- `dnf info` *IME* - zatraži informacije o softverskom paketu imena (*IME*) [*pr. inačicu, veličinu, detaljniji opis,…*]
- `dnf provides` *IME* – provjerimo u kojem softverskom paketu dolazi naredba ili datoteka imena (*IME*); primjerice
  **`dnf provides /etc/shadow`**
- `dnf erase` *IME* ili `dnf remove` *IME* - deinstaliraj softverski paket imena (*IME*).
- `dnf download` *IME* – samo prekopiraj softverski paket imena (*IME*), ali ga ne instaliraj.
- `dnf search` *IME* - pretraži sve softverske pakete (sa svih repozitorija) u potrazi za ključnom riječi (*IME*).
- `dnf list` - ispiši popis softverskih paketa; kako instaliranih, tako i onih dostupnih za instalaciju na svim repozitorijima koji su aktivni na našem računalu:
    - `dnf list installed` - ispiši samo popis softverskih paketa koji su instalirani na našem računalu.
- `dnf repolist` - ispiši listu svih aktivnih repozitorija koje koristimo na našem računalu.
- `dnf whatprovides needs-restarting` - ispiše sve softverske pakete koji traže restart sustava.
- `dnf grouplist` - ispiši popis svih instaliranih grupa paketa, primjerice ispišimo sve grupe softverskih paketa, sa sljedećom naredbom: **`dnf grouplist`**
- `dnf groupinstall` *IME* - instaliraj grupu paketa naziva *IME*, pa bi tako ono naziva: `Development Tools` instalirali na sljedeći način: **`dnf groupinstall "Development Tools"`**
    - `dnf groupremove` *IME* - deinstaliraj grupu paketa naziva *IME*, primjerice:
      **`dnf groupremove "Development Tools"`**
- `dnf makecache` – kreirajmo predmemoriju s listom softverskih paketa i podataka o njima (za ubrzavanje pretrage).
    - `dnf clean all` – obrišimo (lokalnu) predmemoriju s listom softverskih paketa.
- `dnf history` – ispišimo povijest uporabe odnosno pozivanja naredbe `dnf`.
- `dnf autoremove` – obrišimo softverske pakete koji koji nam sada više ne trebaju, a trebali su nam jer su oni u nekom trenutku bili potrebni za neki drugi paket. Dakle to su oni paketi koji su nekada bili instalirani kako bi zadovoljili ovisnost o nekom drugom paketu, ali sada nam više nisu potrebni.
- `dnf distro-sync` – za sinkronizaciju svih paketa na zadnju stabilnu inačicu naše distribucije Linuxa.

***DNF*** koristi direktorij `/etc/dnf/` za pohranu svoje globalne konfiguracije (`/etc/dnf/dnf.conf`) te mnoge poddirektorije i datoteke unutar njega za proširenja; to jest za *module* i *plugine*. Međutim konfiguracija svih repozitorija se i dalje (zbog kompatibilnosti unatrag) pohranjuje unutar direktorija: `/etc/dnf/yum.repos.d/`, u već poznatom formatu, kako je objašnjeno u prethodnim poglavljima.
Za ubrzavanje dohvaćanja paketa, dodajte sljedeće dvije opcije (retka) u datoteku `/etc/dnf/dnf.conf`. One omogućuju dohvaćanje do 10 paketa preko mreže istovremeno te odabir najbržeg repozitorija:

```
max_parallel_downloads=10
fastestmirror=True
```

**Izvori informacija:(1180),(1180.1),** **`man dnf`**, **`man dnf.conf`**, **`man yum`**.

### 7.2.4. Paketni menadžment na Debian kompatibilnim distribucijama Linuxa

Paketni menadžment na *Debian* kompatibilnim distribucijama Linuxa drugi je najpopularniji paketni menadžment uz onaj ***Red Hat-ov*** (RPM). Naime *Debian* kompatibilne distribucije Linuxa koriste svoj paketni menadžment.

Zadatak i ovog paketnog menadžera odnosno upravitelja paketa ili sustava za upravljanje paketima je automatizacija procesa instaliranja, nadogradnje, konfiguriranja i uklanjanja računalnih programa za računalo (Linux) na dosljedan način.
Kao takav i ovaj upravitelj paketa bavi se paketima, distribucijom softvera i podacima u arhivskim datotekama.
Svi Linux programski paketi, pa tako i ovi, sadrže *metapodatke*, kao što je naziv softvera, opis njegove svrhe, broj inačice, ime onoga tko je izradio paket, kontrolni zbroj (po mogućnosti kriptografska *hash* funkcija) i popis ovisnosti potrebnih za ispravno funkcioniranje softvera. Nakon instalacije, *metapodaci* se pohranjuju u lokalnu bazu podataka paketa. Upravitelji paketa obično održavaju ovu bazu podataka o softverskim ovisnostima i informacijama o inačici kako bi spriječili nepodudaranje softvera ili nedostajuće preduvjete.
Primjerice nedostajuće programske biblioteke ili nekompatibilne inačice programa i programskih biblioteka i slično.
I ovaj upravitelj paketa komunicira s repozitorijima softvera.

Za *Debian* kompatibilne distribucije Linuxa (pr. Ubuntu), koristi se `dpkg` (engl. *Debian Package*) kao upravitelj paketa, i on je poput `rpm` alata, alat niže razine, za instaliranje, izgradnju, uklanjanje i upravljanje *Debian* paketima. Sam `dpkg` se u potpunosti kontrolira putem parametara naredbenog retka, koji se sastoje od točno jedne akcije i nula ili više opcija.
Međutim primarno i lakše sučelje za *dpkg* iz naredbenog retka (CLI) je program `apt` (engl. *Advanced Package Tool*), a kao **TUI** (program korisničkog sučelja iz terminala) inačica, tu je dostupan program `aptitude`.

Pogledajmo usporednu tablicu ***RPM*** i ***DPKG*** paketnih menadžera:

| RPM | DPKG | Opis |
|---|---|---|
| **rpm** | **dpkg** | Paketni menadžer niže razine. |
| **yum** ili **dnf** | **apt** | Paketni menadžer više razine. |
| | **aptitude** | Paketni menadžer više razine. **aptitude* ima korisničko sučelje u terminalu. |

A sada pogledajmo tablicu s razlikama u upotrebi i mogućnostima između `dpkg` i `apt` naredbi na *Debian Linuxu*:

| Značajke | dpkg | apt |
|---|---|---|
| Namjena | Upravljanje individualnim *Debian* paketima na niskoj razini. | Upravljanje paketima na višoj razini, uključujući ovisnosti. |
| Instalacija paketa | **dpkg -i paket.deb** | **apt install paket** |
| Uklanjanje paketa (bez konfiguracijskih datoteka) | **dpkg -r paket.deb** | **apt remove paket** |
| Uklanjanje paketa (uključujući konfiguracijske datoteke) | **dpkg -P paket.deb** | **apt purge paket** |
| Preuzimanje paketa iz mrežnog repozitorija | Nije moguće (radi samo s lokalnim **.deb** datotekama)! | Automatski preuzima pakete iz definiranih repozitorija! **apt install paket** |
| Potraga za novim inačicama programa | Nije moguće! | **apt update** |
| Ažuriranje sustava | Nije moguće! | **apt upgrade** ← ažurira sve instalirane pakete! |
| Prikaz informacija o paketu | **dpkg -s paket** | **apt show paket** |
| Pretraga za paketima | Nije moguće! | **apt search paket** |
| Lista instaliranih paketa | **dpkg -l** | **apt list --installed** |
| Popravljanje ovisnosti (*dependencies*) paketa | Nije moguće! | **apt install -f** ← pokušava popraviti „*pokidane*“ ovisnosti. |
| Interaktivno sučelje | Nema interaktivno sučelje. | Nema interaktivno sučelje (za to postoji **aptitude**). |
| Zaključavanje paketa za nadogradnju | Nije moguće! | **apt-mark hold paket** |
| Prikaz paketa sa dostupnim nadogradnjama | Nije moguće! | **apt list --upgradable** |
| Konfiguriranje repozitorija | Nije moguće! | **apt edit-sources** |

**Slijede primjeri!**
**1.** Osvježavanje liste dostupnih programa, preko dostupnih repozitorija na lokalnoj mreži ili internetu:
```
apt update
```

**2.** Nadogradnja svih instaliranih paketa na najnovije inačice:
```
apt upgrade
```

**3.** Nadogradnja svih instaliranih paketa i automatsko uklanjanje zastarjelih:
```
apt full-upgrade
```

**4.** Instalacija programa `mc` (*Midnight Commander*) i svih ovisnosti (biblioteka i drugih potprograma nužnih za njegov rad):
```
apt install mc
```

**4.1.** Instalacija više programa odjednom (*Apache* i *Apache alata*), s automatskim odgovorom (*Yes*) prilikom pitanja o instalaciji:
```
apt install apache2 apache2-utils -y
```

**4.2.** Instalacija specifične verzije paketa (`apache2` inačice `2.2.14-5ubuntu8.7`):
```
apt install install apache2=2.2.14-5ubuntu8.7
```

**5.** Uklanjanje paketa, ali zadržavanje konfiguracijskih datoteka (za `mc` (*Midnight Commander*):
```
apt remove mc
```

**6.** Popravljanje pokidanih ovisnosti između paketa:
```
apt install -f
```

## Konfiguracija repozitorija

*Debian* repozitoriji su skladišta softvera koja sadrže pakete, uključujući aplikacije, programske biblioteke, sigurnosna ažuriranja i sistemske alate. Ovi repozitoriji su centralno održavani i organizirani tako da pružaju jednostavan i siguran način za instalaciju i nadogradnju softvera na *Debian* sustavima, identično kao što se koriste repozitoriji na *Red Hat Linuxu*.

**Vrste Debian repozitorija:**

- **Main** – označava glavni repozitorij koji sadrži sav slobodan softver koji je u skladu s *Debian* smjernicama za slobodan softver (DFSG).
- **Contrib** – sadrži slobodan softver koji ima ovisnosti o neslobodnom softveru (koji nije *open source*) koji se nalazi izvan repozitorija.
- **Non-free** – sadrži softver koji nije u skladu s DFSG, što može uključivati vlasnički softver i upravljačke programe.

*Debian* repozitoriji se konfiguriraju putem datoteke `/etc/apt/sources.list` i eventualno putem datoteka u direktoriju `/etc/apt/sources.list.d/`. Ove datoteke sadrže URL-ove repozitorija uz nužne specifikacije koje vrste paketa i distribucija treba koristiti. Pogledajmo primjer jedne ovakve datoteke (`/etc/apt/sources.list`):
```
deb http://deb.debian.org/debian/ bookworm main contrib non-free
deb-src http://deb.debian.org/debian/ bookworm main contrib non-free

deb http://security.debian.org/debian-security bookworm-security main contrib non-free
```
Sintaksa unutar ove datoteke je:

- `deb` – oznaka za binarne pakete (koje sustav koristi).
- `deb-src` - oznaka za pakete s programskim kôdom (koji se mogu koristiti za izgradnju binarnih paketa).
- `http://deb.debian.org/debian/:` - URL (adresa) repozitorija.
- `bookworm` – naziv distribucije (može biti stabilna, nestabilna, testna ili kodno ime kao *bookworm, bullseye* itd.). *Debian* vršne inačice naziva kodnim imenima (pr. *Debian 12 - kodno ime: Bookworm, Debian 11 - Bullseye*, …).
- `main contrib non-free` – sekcije repozitorija koje definiraju vrstu softvera (pr. *non-free*).

**Upravljanje repozitorijima**

Ako želite dodati specifične repozitorije, poput onih koji nisu službeni ili su specijalizirani (npr. za određeni softver), to se može učiniti dodavanjem redaka u datoteku `/etc/apt/sources.list` ili kreiranjem nove datoteke u direktoriju `/etc/apt/sources.list.d/`, kao obične tekstualnu datoteku s ekstenzijom `.list`.

**Provjera i dodavanje GPG ključeva**.

Većina repozitorija zahtijeva da dodate njihov GPG ključ kako bi se osigurala autentičnost paketa. To možete učiniti naredbom:
```
wget -qO - https://repo.primjer.com/key.gpg | sudo apt-key add -
```
Ili pomoću preporučene (novije) metode koju ćemo ubrzo opisati.

**Pogledajmo primjer dodavanja repozitorija (za Debian v.12. [*bookworm*]) za multimediju**

Kreirat ćemo novu datoteku: `/etc/apt/sources.list.d/multimedia.list` u koju ćemo dodati repozitorij odnosno:
```
deb http://www.deb-multimedia.org bookworm main non-free
```
← Ovim korakom smo kreirali definiciju repozitorija u zasebnu datoteku, te si olakšali kasniji rad s tim repozitorijem.
Sada trebamo napraviti proces spajanja na sve definirane repozitorije i dohvaćanje sadržaja s njih:
```
apt update
Err:4 http://www.deb-multimedia.org bookworm InRelease
  The following signatures couldn't be verified because the public key is not available:
NO_PUBKEY 5C808C2B65558117
```
← Ovdje dobivamo grešku jer moramo dodati javni **GPG** ključ koji odgovara privatnom ključu korištenom za potpisivanje paketa u ovom repozitoriju. To možemo napraviti instalacijom paketa u kojem je GPG ključ ili ručno (*detalji ovise o inačici sustava*).
Uz preduvjet da imamo program `curl` i `gpg`, te direktorij `~/.gnupg`, ručno ćemo uvesti GPG certifikat sa:
```
curl -fsSL https://www.deb-multimedia.org/pool/main/d/deb-multimedia-keyring/deb-multimedia-keyring_2016.8.1.dsc | gpg --dearmor -o /usr/share/keyrings/deb-multimedia-keyring.gpg
```
← I sada možemo nastaviti koristiti ovaj novi repozitorij. **Za detalje o GPG pogledajte sljedeću cjelinu!**

**Izvori Informacija: (1734),(1735), `man dpkg`, `man apt`, `man aptitude`, `man sources.list`, `man gpg`.**

### 7.2.5. GNU Privacy Guard (GPG) i repozitoriji

*GNU Privacy Guard* (**GPG** ili **GNU PG**) odnosno konkretno naredba `gpg` je alat za enkripciju i digitalno potpisivanje podataka, temeljen na *OpenPGP* standardu. *GPG* omogućuje korisnicima sigurno šifriranje podataka, verifikaciju integriteta podataka, kao i autentifikaciju pomoću digitalnih potpisa.
GPG program je kompatibilan s *Open PGP* specifikacijom definiranom u: **RFC 4880** te novijim **RFC 9580**.

Osnovne značajke *GNU Privacy Guard* (**GPG**) su:

- **Kriptiranje (šifriranje) podataka** – GPG omogućava kriptiranje podataka kako bi se osigurala njihova povjerljivost. Samo osoba koja posjeduje odgovarajući privatni ključ može dešifrirati podatke koji su šifrirani s odgovarajućim javnim ključem.
- **Digitalno potpisivanje** – GPG može dodati digitalne potpise na podatke ili datoteke. Digitalni potpis služi za autentifikaciju izvora podataka (ili datoteka) te osigurati da podaci nisu mijenjani nakon što su potpisani.
- **Verifikacija potpisa** – GPG omogućuje provjeru autentičnosti podataka, odnosno provjeru je li datoteka ili poruka zaista potpisana od strane očekivanog autora i je li netaknuta. GPG se koristi za verifikaciju softvera kako bi se osiguralo da softver dolazi iz pouzdanog izvora i da nije izmijenjen. To je ključno kod instalacije softvera sa **repozitorija**, koje koriste distribucije poput ***Red Hat*** ili *Debian/Ubuntu* i većina drugih.
- **Upravljanje ključevima za kriptiranje** – GPG koristi sustav parova javnog i pripadajućeg privatnog ključa. Javne ključeve možete slobodno dijeliti, dok su privatni ključevi tajni i koriste se za šifriranje i potpisivanje.



GPG se često koristi i za šifriranje elektroničke pošte i drugih oblika komunikacije kako bi se osigurala povjerljivost i integritet poruka. Najpoznatija aplikacija za ovo je **GnuPG** u kombinaciji s klijentima elektroničke pošte poput **Thunderbird** koji koristi **Enigmail** dodatak.
GPG se koristi i za šifriranje pojedinačnih datoteka ili arhiva podataka, čime se osigurava da samo autorizirani korisnici mogu pristupiti sadržaju tih datoteka.

Nadalje, GPG se koristi i u sustavima za verzioniranje koda poput **Git**-a, gdje se koristi za potpisivanje ***commit**-ova*, čime se osigurava integritet i autentičnost promjena u kôdu.

## GPG - repozitoriji i paketi

Kada programeri ili osobe koje održavaju softverske pakete kreiraju novi paket za distribuciju na **repozitoriju**, taj paket se digitalno potpisuje privatnim ključem. Ovaj digitalni potpis služi kao dokaz autentičnosti, dokazujući da je paket doista kreiran od strane očekivanog izvora.

Kada korisnik preuzima paket putem upravitelja paketa (`apt` u slučaju *Debiana/Ubuntua* ili `yum`/`dnf` u slučaju *Red Hat* Linuxa), u pozadini se koristi GPG javni ključ konkretnog repozitorija za provjeru digitalnog potpisa paketa. Ako je potpis ispravan, to znači da je paket autentičan i nije mijenjan od strane treće strane.
Za detalje o stvaranju i verifikaciji digitalnih potpisa pogledajte **RFC 4880** i to cjelinu: **5.2: "Signatures".**

Osim provjere autentičnosti, GPG također omogućuje provjeru integriteta paketa. Kada je paket digitalno potpisan, uz njega dolazi i *hash* vrijednost (kao svojevrsni kriptografski otisak) sadržaja paketa. Pri preuzimanju paketa s ***repozitorija***, GPG izračunava *hash* vrijednost preuzetog paketa i uspoređuje ju s *hash* vrijednošću koja je potpisana i poslana zajedno s paketom. Ako se te vrijednosti podudaraju, to znači da paket nije bio mijenjan i da je njegov integritet očuvan.

Naime, bez ove provjere, zlonamjerni napadači bi mogli zamijeniti legitimne pakete zlonamjernim inačicama, a GPG to sprječava, jer bi svaka promjena paketa promijenila i njegov digitalni potpis. Stoga korisnici mogu biti sigurni da softver koji instaliraju dolazi iz pouzdanog izvora (***repozitorija***) te da nije neovlašteno mijenjan.

**Kako to radi?**
Kada dodajemo novi repozitorij, sustav obično zahtijeva da preuzmemo i instaliramo GPG javni ključ tog repozitorija. Upravo taj ključ se koristi za provjeru potpisa svih paketa koje želimo instalirati iz tog repozitorija.

Kada ažuriramo popis paketa (`apt update` ili `yum update`) ili instalirate paket (`apt install` ili `yum install`), `apt` i `yum`/`dnf` koriste GPG za automatsku provjeru potpisa svakog paketa.
Ako GPG verifikacija ne uspije (zbog nedostajućeg ključa ili promijenjenog paketa), sustav će nas upozoriti ili izbaciti grešku te neće instalirati taj paket, čime se korisnik štiti od potencijalno opasnog softvera.

## Debian (Ubuntu)

**Gdje se pohranjuju GPG ključevi**
Tradicionalna spremišta GPG ključeva za Debian Linux su u `/etc/apt/trusted.gpg` ili direktoriju `/etc/apt/trusted.gpg.d/`. Dakle ovdje su pohranjeni ključevi koji se koriste za provjeru potpisa repozitorija. Ova metoda se smatra zastarjelom jer svi ključevi u ovom spremištu imaju implicitno povjerenje za sve repozitorije, što nije najsigurnije. Novo spremište odnosno preporučena metoda je pohrana GPG ključeva u `/usr/share/keyrings/` direktorij. Ovdje se ključevi pohranjuju specifično za svaki pojedini repozitorij, što omogućava bolju sigurnost i izolaciju između repozitorija.

**GPG** ključevi na *Debian* sustavima obično su bili pohranjeni u tzv. ***ASCII-armored*** formatu, koji koristi `.asc` ili `.gpg` ekstenziju. Ovaj format je *Base64* kodirani format s "BEGIN PGP PUBLIC KEY BLOCK" zaglavljem, što ga čini pogodnim za slanje putem emaila ili pohranu u tekstualnim datotekama te kompatibilnim s drugim programima poput `apt-key`. Druga opcija (od ***Debian 9.x***) je binarni format koji je kompaktniji i direktno čitljiv iz `gpg` programa.

**Kako se importiraju ključevi**
Za uvoz (importiranje) ključeva za *Debian* (ili *Ubuntu*) distribucije Linuxa postoji nekoliko metoda:
Korištenje `apt-key` (zastarjelo) – ova metoda se koristi za dodavanje ključeva u `/etc/apt/trusted.gpg` ili u direktorij `/etc/apt/trusted.gpg.d/`. To možemo postići primjerice sa:

```
apt-key adv -keyserver keyserver.ubuntu.com -recv-keys <KEY_ID>
```

🡐 Pri tome je `<KEY_ID>` ID ključa skraćeni identifikator GPG ključa, izveden iz punog *fingerprint-a* ključa, koji se koristi za preuzimanje i identifikaciju javnih ključeva s poslužitelja koji sadrže GPG ključ. *Fingerprint* (otprilike prevedeno kao "otisak prsta") GPG ključa je jedinstveni niz heksadecimalnih znakova koji predstavlja kriptografski otisak cijelog javnog ključa. *Fingerprint* se koristi za jednoznačnu identifikaciju GPG ključa a on je jedinstven za svaki ključ i služi kao sigurnosna provjera. Pogledajmo i drugu (stariju) metodu importiranja *GPG* ključa:

```
curl -fsSL <URL_do_ključa> | apt-key add -
```

Odnosno pogledajmo noviju metodu koja se i preporuča:

```
curl -fsSL <URL_do_ključa> | gpg --dearmor | tee /usr/share/keyrings/<repo>-keyring.gpg > /dev/null
```

Nakon toga uobičajeno je potrebno dodati putanju do GPG ključa; u retku u kojem definiramo repozitorij.
Dakle u `sources.list` ili `.list` datoteci navedite repozitorij uz opciju `signed-by`, poput:

```
deb [signed-by=/usr/share/keyrings/<repo>-keyring.gpg] <URL> <distribution>
<components>
```

Listu svih uvezenih GPG ključeva za *Debian* Linux, možemo vidjeti s naredbom:

```
gpg --list-keys
```

Dok za detalje o ključevima možete koristiti naredbu:

```
gpg --fingerprint
```

## Red Hat kompatibilne distribucije (pr. Fedora, Oracle, Rocky, …)

**Gdje se pohranjuju GPG ključevi i kako se uvoze (importiraju)**
Tradicionalna spremišta GPG ključeva kod ovih distribucija Linuxa su u direktoriju: `/etc/pki/rpm-gpg/`.

*Red Hat* je počeo prelaziti s *ASCII-armored* formata zapisa GPG ključeva na binarni format s RHEL 5 i 6, a RHEL 7 i noviji uobičajeno koriste binarni format.

U *Red Hat* baziranim distribucijama Linuxa je situacija po pitanju dodavanja *GPG* ključeva automatizirana jer će pri prvom pokušaju instalacije paketa iz novo dodanog repozitorija, `yum` ili `dnf` automatski zatražiti potvrdu za uvoz GPG ključa ako je opcija `gpgcheck=1` postavljena u `.repo` datoteci (što je zadano ponašanje).
Pogledajmo primjerice već dodani repozitorij imena **ELRepo**, to jest prvi dio njegove konfiguracijske datoteke: `/etc/yum.repos.d/elrepo.repo`:

```
[elrepo]
name=ELRepo.org Community Enterprise Linux Repository - el9
baseurl=http://elrepo.org/linux/elrepo/el9/$basearch/
        http://mirrors.coreix.net/elrepo/elrepo/el9/$basearch/
mirrorlist=http://mirrors.elrepo.org/mirrors-elrepo.el9
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-elrepo.org
```

Uočite zadnja dva retka:

- `gpgcheck=1` – označava da se *GPG* provjera ključeva treba raditi (što je zadano ponašanje).
- `gpgkey=` – označava da je *GPG* ključ već dohvaćen i snimljen u datoteku: `/etc/pki/rpm-gpg/RPM-GPG-KEY-elrepo.org`.

To konkretno znači da je repozitorij **ELRepo**, već dodan, te da je nakon prve naredbe **`yum update`** ili **`yum install`** `XY` uvezen *GPG* ključ i pohranjen unutar datoteke: `/etc/pki/rpm-gpg/RPM-GPG-KEY-elrepo.org`.
Za detaljan ispis podataka o svim dostupnim repozitorijima možete pokrenuti naredbu:

```
yum repolist -v
```



Za detalje o kriptografiji, pogledajte i poglavlje:
**26.4.4. Secure Socket Layer, Transport Layer Security i kriptografija.**

**Izvori informacija: (1736),** **`man apt`**, **`man yum`**, **`man gpg`**, **`RFC 4880`**, **`RFC 9580`**.

# 7.3. Stanja rada Linuxa (*Runlevels*)

Tijekom pokretanja operativnog sustava Linux ili UNIX, sustav se pokreće na određenu razinu rada odnosno takozvani ***runlevel*** prema *init* terminologiji, kako je opisano u **Unix System V** definiciji.
Razina ili stanje rada svakog Unixa ili Linuxa ovisna je o potrebama ili trenutnoj namjeni rada sustava:

- Za poslužitelje (*servere*) se standardno pokreće ***runlevel 3*** koji nam omogućava višekorisnički rad s aktivnom mrežom i svim mrežnim servisima, kao i svim ostalim funkcionalnostima sustava.
- S druge strane za osobna (*desktop*) računala se obično pokreće ***runlevel 5*** koji ima sve mogućnosti kao ***runlevel 3***, ali ima i grafičko sučelje koje se u UNIX/Linux sustavima zove ***X Window System***.
- Osim navedenih razina rada, za posebne namjene se koriste i ostala stanja rada odnosno *runleveli*:
  - **Runlevel 1** se koristi za održavanja, kada ne želimo da se itko može spojiti na sustav preko mreže, već samo lokalni administrator (`root` korisnik). Koristi se za potrebe popravljanja većih grešaka na sustavu, tvrdom disku i slično pa stoga ne želimo mrežne servise niti spajanje drugih korisnika na sustav.
  - **Runlevel 0** se koristi za gašenje sustava, ovaj *runlevel* se brine o tome kako bi se svi programi i servisi uredno ugasili (tijekom gašenja sustava).
  - **Runlevel 6** se koristi za restartanje sustava, način rada mu je sličan kao `runlevel 0`.

Popis i malo više detalja o *runlevelima* pogledajmo u tablici:

| Run Level | Mode | Opis |
|---|---|---|
| 0 | Halt | Gasi operativni sustav (shutdown) |
| 1 | Single-User Mode | Ne konfigurira mrežne interface-e , ne pokreće servise (daemon), dozvoljava samo pristup root korisniku |
| 2 | Multi-User Mode | (Ovisno o Linux distribuciji ) ali obično podiže mrežne interface-e ,ali ne pokreće mrežne servise (daemons), dozvoljen je pristup svim korisnicima |
| 3 | Multi-User Mode with Networking | Normalno pokreće operativni sustav |
| 4 | Undefined | Često ne definirano / moguće je definirati za svoje potrebe (Slackware Linux ga koristi kao Runlevel 5) |
| 5 | X11 | Starta runlevel 3 + display manager(X ) [pokreće X Window grafički sustav] |
| 6 | Reboot | Restarta operativni sustav |

Naime tijekom svakog pokretanja operativnog sustava, nakon što se učita Linux kernel, pokreće se prvi proces odnosno program koji se zove: `init`. On potom čita datoteku: `/etc/inittab` u kojoj je definirana *runlevel* razina u koju se sustav potom treba pokrenuti.

Pogledajmo izgled datoteke: `/etc/inittab` u slučaju kada se sustav treba pokrenuti u **init 3**:
`id:3:initdefault:`
Dakle broj `3` označava *init* broj `3` odnosno *runlevel* broj tri.

Pogledajmo i strukturu procesa odnosno pokrenutih programa i servisa, s pogledom na ***init*** proces, koju smo dobili s naredbom: `pstree -a` koja dolazi u softverskom paketu `psmisc`. Ispis je skraćen zbog lakšeg razumijevanja:

```
init
  ├─auditd
  │   └─{auditd}
  ├─crond
  ├─login
  │   └─bash
  ├─mingetty /dev/tty2
  ├─ . . .
  ├─rsyslogd -i /var/run/syslogd.pid -c 5
  │   ├─{rsyslogd}
  ├─sshd
  │   └─sshd
  │       └─bash
  │           └─pstree -a
  └─udevd -d
      ├─udevd -d
      ├ . . .
```



Vidljivo je kako `init` proces pokreće sve ostale servise, kao i sve programe (procese) na sustavu. Nadalje, `init` proces se i sâm pokreće kao servis. Pošto je on prvi program (proces) na Linuxu, on ima identifikator procesa to jest ***PID*** broj jedan (**1**). ***PID*** brojevi su jedinstveni brojčani identifikatori svakog pokrenutog programa ili servisa na Linuxu ili UNIXu.

On (`init`) na osnovi *runlevel* broja s kojim se treba pokrenuti sustav, pokreće skripte koje se nalaze u direktorijima unutar glavnog vršnog direktorija za sve *runlevel-e*, koji je: `/etc/rc.d/`.
Pri tome je osnovna struktura navedenih direktorija ispod vršnog direktorija: `/etc/rc.d/` sljedeća:

```
drwxr-xr-x 2 root root 4.0K Nov 11 15:05 rc0.d
drwxr-xr-x 2 root root 4.0K Nov 11 15:05 rc1.d
drwxr-xr-x 2 root root 4.0K Nov 11 15:05 rc2.d
drwxr-xr-x 2 root root 4.0K Nov 11 15:05 rc3.d
drwxr-xr-x 2 root root 4.0K Nov 11 15:05 rc4.d
drwxr-xr-x 2 root root 4.0K Nov 11 15:05 rc5.d
drwxr-xr-x 2 root root 4.0K Nov 11 15:05 rc6.d
```

Važno je razumjeti da pri tome direktorij (mapa):
**`/etc/rc.d/rc0.d/`** - sadrži sve skripte koje se moraju pokrenuti, ako sustav ulazi u ***init 0*** stanje rada.
**`/etc/rc.d/rc1.d/`** - sadrži sve skripte koje se moraju pokrenuti, ako sustav ulazi u ***init 1*** stanje rada.
**`/etc/rc.d/rc3.d/`** - sadrži sve skripte koje se moraju pokrenuti, ako sustav ulazi u ***init 3*** stanje rada, ... i tako dalje.

Pogledajmo sadržaj direktorija za ***runlevel 3***, s naredbom: `ls -al`**.** Dakle govorimo o direktoriju `/etc/rc.d/rc3.d/`.
Skripte odnosno skriptne datoteke smo pri tome označili **`podebljano`**:

```
lrwxrwxrwx 1 root root 22 Nov 11 15:04 K15htcacheclean -> ../init.d/htcacheclean
lrwxrwxrwx 1 root root 13 Jul 24 16:06 K35nmb -> ../init.d/nmb
lrwxrwxrwx 1 root root 13 Jul 24 16:06 K35smb -> ../init.d/smb
lrwxrwxrwx 1 root root 20 Jul 24 16:06 K50netconsole -> ../init.d/netconsole
lrwxrwxrwx 1 root root 15 Nov 11 15:05 K50snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root root 19 Nov 11 15:05 K50snmptrapd -> ../init.d/snmptrapd
lrwxrwxrwx 1 root root 14 Nov 11 15:05 K74nscd -> ../init.d/nscd
lrwxrwxrwx 1 root root 15 Nov 11 15:04 K75netfs -> ../init.d/netfs
lrwxrwxrwx 1 root root 19 Nov 11 15:04 K75udev-post -> ../init.d/udev-post
lrwxrwxrwx 1 root root 15 Nov 11 15:04 K89rdisc -> ../init.d/rdisc
lrwxrwxrwx 1 root root 18 Jul 24 16:06 S08iptables -> ../init.d/iptables
lrwxrwxrwx 1 root root 21 Jul 24 16:06 S08modules_dep -> ../init.d/modules_dep
lrwxrwxrwx 1 root root 17 Nov 11 15:04 S10network -> ../init.d/network
lrwxrwxrwx 1 root root 17 Nov 11 15:05 S12rsyslog -> ../init.d/rsyslog
lrwxrwxrwx 1 root root 14 Jul 24 16:06 S55sshd -> ../init.d/sshd
lrwxrwxrwx 1 root root 18 Jul 24 16:06 S80sendmail -> ../init.d/sendmail
lrwxrwxrwx 1 root root 15 Nov 11 15:04 S85httpd -> ../init.d/httpd
lrwxrwxrwx 1 root root 15 Jul 24 16:06 S90crond -> ../init.d/crond
lrwxrwxrwx 1 root root 11 Nov 11 15:04 S99local -> ../rc.local
```

Vidimo kako su sve datoteke čije ime datoteke počinje sa **`S`**`*` i **`K`**`*` zapravo simbolički linkovi na datoteke koje se nalaze unutar direktorija `/etc/rc.d/init.d/`.
To je tako zbog toga što sve servise (tzv. *daemone*) tj. njihove skripte kreiramo samo na jednom mjestu i to u direktoriju: `/etc/rc.d/init.d/`. Tek potom u željenom***** runelevel direktoriju: `/etc/rc.d/rc*.d/` trebamo kreirati simbolički link na postojeću datoteku u direktoriju: `/etc/rc.d/init.d/`.
Pri tome, ako ime simboličkog linka počinje s **`K`**, ta skripta (datoteka) će biti pokrenut s parametrom `stop`, a ako joj ime počinje sa **`S`** bit će pokrenut s parametrom `start`.



Ono što je važno kod izvornih skripti koje se nalaze unutar direktorija: `/etc/rc.d/init.d/` jest to da one moraju imati minimalno implementirane tri metode: `stop`, `start` i `restart`.

Osim toga, u slučaju *Red Hat* baziranih Linuxa, poput primjerice *CentOS/Rocky* Linuxa, ako skripta sadrži ključnu riječ: `# chkconfig:` parametri iza nje će se koristiti za automatsko kreiranje `/etc/rc.d/rc*.d` simboličkih poveznica (linkova), pa ih prema tome ne moramo kreirati sâmi. Ovakav način rada je standardan kod normalne instalacije servisa (*daemona*), poput: *apache (web poslužitelj)*, *ssh (SSH servis)*, *mariadb (SQL baza podataka)* i ostalih.



Novije distribucije ***Red Hat*** (**v.7+**) i kompatibilnih distribucija Linuxa (kao i neke druge) više ne koriste (**System V**) **init** sustav, već noviji **systemd** sustav, o kojem ćemo govoriti u sljedećim poglavljima.

**Izvori informacija: (849),(K-12), `man runlevel`.**

## 7.3.1. *Init* i sistemski servisi (*daemoni*)

***Slijedi napredno poglavlje (7.3.1.x), za sustave koji koriste INIT proces!***
Do sada smo naučili kako je na većini *UNIX* operativnih sustava, kao i na svim klasičnim *Linuxima*, upravo `init` proces taj koji se prvi pokreće prije svih drugih procesa odnosno programa. Važno je znati kako je ***PID*** broj (*Proces ID*) od ***init*** procesa, broj jedan (**1**), pošto je on prvi proces koji pokreće sve ostale procese i brine se o njima. Standardno se tijekom pokretanja operativnog sustava, sustav pokreće u ***runlevel 3***, kod Linuxa koji ne pokreću grafičko sučelje (***X Window)***, a za one koji pokreću grafičko sučelje, sustav se pokreće u ***runlevel 5***.

Ako je u pitanju pokretanje standardnog ***runlevel 3***, događa se sljedeće:

**1.** Nakon što je operativni sustav pokrenut i učitao se kernel, pokreće se program koji se zove `init` i on ulazi u direktorij (*mapu*): `/etc/rc.d/rc3.d/`

**2.** Iz tog direktorija pokreću se sve skripte (datoteke) čije ime počinje sa slovom **`K`** i to prema rednom broju. Dakle prvo s najmanjim brojem primjerice: **K1..**, **K2..** pa **K3** te sve do najvećeg broja i to tako da se svaka od njih poziva s parametrom `stop`. Dakle prvo datoteka/skripta **`K1`**... `stop` pa **K2**... `stop`, … Primjerice datoteka/skripta imena: `K15htcacheclean` se pokreće sa: `K15htcacheclean stop` i to tako za sve njih (**K***) koliko god ih ima.



To znači da se prvo zaustavljaju (*stopiraju*) svi servisi te se čisti i priprema sustav, prije sâmog pokretanja servisa. Ovo je dodatno važno u slučajevima kada je neka od skripti ostavila neko smeće zbog bilo kojeg razloga: zaglavila se tijekom prošlog gašenja računala, nije se uredno zaustavila i slično.

**3.** Nakon toga se pokreću one skripte sa prvim slovom **`S`** u imenu. Dakle tek sada se zapravo pokreću servisi, ponovno s najmanjim brojem, primjerice: **S1…**, **S2…** pa **S3..** i tako do najvećeg broja i to tako što se svaka od njih poziva sa parametrom `start`. Dakle **`S1`**... `start`. Primjerice datoteka/skripta imena: `S10network` se pokreće sa: `S10network start`, a to vrijedi za sve njih s prvim slovom **`S*`** u imenu, koliko god ih ima.

## Zbog čega brojevi kod imena datoteka to jest skripti, koji počinju sa slovima `S` i `K`?

Zbog toga kako bismo točno mogli kontrolirati koja će se datoteka odnosno skripta prije ili poslije koje skripte pokrenuti.
To znači da se brojevi koriste kao indikatori redoslijeda pokretanja (ili zaustavljanja).
Jedina mala razlika su ***runleveli 0*** (to je *halt/shutdown*) te ***runlevel 6*** (to je *reboot*).

Zbog toga, za ***runlevel 0*** i ***runleveli 6*** odnosno njihovi pripadajući direktoriji, u kojima se nalaze njihove datoteke/skripte: `/etc/rc.d/rc0.d/` i `/etc/rc.d/rc6.d/`, se sastoje većinom od datoteka imena `K*` pošto oni kod gašenja ili restarta računala moraju "uredno" zaustaviti odnosno poubijati sve servise, a odatle i slovo ***K*** (engl. *Kill*).

U njima su obično samo dvije **`S*`** datoteke:

- Prva je ona koja nakon što je sve uredno zaustavljeno preko `K*` skripti, poubija sve što je možda zaostalo.
- Druga je ona koja u slučaju `init 0` gasi računalo, a u slučaju `init 6` restarta računalo.



U svakom trenutku, moguće je promijeniti *runlevel*, naredbom `init` **`X`**, pri čemu je **`X`** broj *runlevela* u koji želimo preći odnosno prebaciti sustav.

**Slijede primjeri:**
**1.** Želimo li ugasiti računalo to možemo napraviti i sa:
```
init 0
```

**2.** Ako ga ipak želimo *restartati*, onda pokrećemo drugi *init* broj:
```
init 6
```

**3.** Želimo li nešto raditi/popraviti na sustavu i ne želimo dozvoliti pristup nikome preko mreže već sâmo nama, kao isključivo ***root*** korisniku, jer se nalazimo fizički ispred računala/poslužitelja. Tada možemo koristiti odnosno pokrenuti sljedeću naredbu:
```
init 1
```

**Izvori informacija: (848),(849),(K-12), `man init`,`man runlevel`.**

#### 7.3.1.1. *Init* sistemski servisi (*daemoni*) detaljnije

Sistemski servisi koji se u Unix/Linux svijetu nazivaju "*daemoni*" su programi koji se pokreću u pozadini, a koji su zamišljeni da rade bez stalne kontrole i uplitanja korisnika. Dakle oni su zaduženi za pojedine funkcionalnosti sustava.
Naime funkcionalnost cijelog Linux sustava je raspodijeljena u cijeli nîz servisa, od kojih svaki odrađuje svoj zadatak.

Tako primjerice jedna skripta pokreće servise koji inicijaliziraju mrežne kartice i konfiguriraju ih, druga koja pokreće primjerice SSH mrežni servis, treća pokreće neki drugi mrežni servis poput *NFS* mrežnog datotečnog sustava i tako dalje, svaka za svoju namjenu. U praksi, servise pozivaju *shell* skripte koje se inicijalno nalaze u direktoriju `/etc/init.d/` te se njihove poveznice nalaze u direktoriju `/etc/rc.d/init.d/`, a koje sustav pokreće/zaustavlja ili restarta prema pravilima koja smo već objasnili u prijašnjem poglavlju. Kako ih možemo ručno pokretati ili zaustaviti i restartati, te što moraju sadržavati, objasnit ćemo ubrzo.



Jedna od najvažnijih skripti sustava u ovom direktoriju, po pitanju mreže je: `/etc/rc.d/init.d/network` skripta koja je zadužena za pokretanje mreže: od mrežnih sučelja, do konfiguracije istih, dodavanja statičkih ruta (ako ih ima), pa sve do gašenja ili restartanja mrežnih sučelja. Ova skripta u radu čita i mnoge mrežne konfiguracijske datoteke, locirane uglavnom u direktoriju: `/etc/sysconfig/network-scripts/`

Zbog jednostavnosti, navest ćemo primjer izlistanja sâmog početka skripte koja pokreće *SSH Server* (skripta: `/etc/rc.d/init.d/sshd`) na kojoj ćemo opisati procese inicijalizacije. Stoga pogledajmo ovu skriptu (skraćenu):

```
#!/bin/bash
#
# sshd Start up the OpenSSH server daemon
#
# chkconfig: 2345 55 25
# description: SSH is a protocol for secure remote shell access. \
# This service starts up the OpenSSH server daemon.
```

Linija odnosno redak: `# chkconfig: 2345 55 25` znači sljedeće (vidjet ćemo i u sljedećem poglavlju):

- Prvi nîz brojeva: `2345` u ovom slučaju, definiraj u kojim sve *runlevelima* će se pokretati ova skripta.
- Drugi broj: `55` u ovom primjeru je prioritet tijekom pokretanja i označava `S*` broj (to bi bilo `S55`...).
- treći broj: `25` u ovom primjeru je prioritet tijekom zaustavljanja i označava `K*` broj (to bi bilo `K25`...).

U ovom slučaju sustav će kreirati simboličke linkove naziva `S55sshd` u *runlevel* direktorijima `2`,`3`,`4` i `5`.
Dakle: `/etc/rc.d/rc*.d` te `K25sshd` u preostalim *runlevelima* u kojima sustav mora gasiti ovaj servis, dakle u *runlevelima**: `0` (*shutdown*), `1` (*singleuser*, *no network*) i `6` (*reboot*). Osim navedenog, svaka skripta koja poziva određene *servise* mora imati minimalno implementirane tri metode: `stop`, `start` i `restart`.
Kako to pojednostavljeno izgleda na primjeru SSH skripte, koja je nastavak na početak `/etc/rc.d/init.d/sshd` skripte izlistane gore(ponovno smo skratili ispis):

```
start()
{
        [ -x $SSHD ] || exit 5
        [ -f /etc/ssh/sshd_config ] || exit 6
        # Create keys if necessary
        if [ "x${AUTOCREATE_SERVER_KEYS}" != xNO ]; then
                do_rsa1_keygen
                do_rsa_keygen
                do_dsa_keygen
        fi
}
stop()
{
        echo -n $"Stopping $prog: "
        killproc -p $PID_FILE $SSHD
        RETVAL=$?
        # if we are in halt or reboot runlevel kill all running sessions
        # so the TCP connections are closed cleanly
        if [ "x$runlevel" = x0 -o "x$runlevel" = x6 ] ; then
            trap '' TERM
            killall $prog 2>/dev/null
            trap TERM
        fi
}
restart() {
        stop
        start
}
```

U ispisu vidimo kako unutar skripte `/etc/init.d/sshd` odnosno `/etc/rc.d/init.d/sshd` koja je poveznica na prvu skriptu, ovdje imamo tri funkcije: `start`, `stop` i `restart` koje u slučaju pozivanja skripte s parametrom `start` pokreću *SSH* servis, koji je binarna izvršna datoteka: `/usr/sbin/sshd` i koja je zadužena za servis odnosno *SSH* mrežni poslužitelj.

Naime u trenutku pokretanja *init* (inicijalizacijske) skripte za neki servis, pokreće se njegova skripta s parametrom `start` odnosno kod zaustavljanja pokreće se ista skripta s parametrom `stop` koja, kako je vidljivo iz programskog kôda, zaustavlja isti servis, odnosno parametar (konkretno metoda) `restart`, prvo poziva metodu `stop`, a potom metodu `start`.



Na novijim inačicama ***RedHat/CentOS*** 7+ `init` proces i servisi vezani uz njega, poput naredbe `chkconfig`, zamijenjeni su sa `systemd` servisom i alatima koji dolaze s njim. Ove nove alate, trenutno ćemo spominjati u jednom od sljedećih poglavlja.

Za sada sâmo pogledajmo usporednu tablicu funkcionalnosti `init` + `chkconfig` (Sysvinit) u odnosu na novi `systemd` i pripadajuću naredbu `systemctl`.

**Usporedna tablica: init vs. systemd.**

| **SysVinit** naredbe (u jednm retku) | **Systemd** naredba (u jednm retku) | **Opis** |
|---|---|---|
| **service** IME_SERVISA **start** | **systemctl start** IME_SERVISA | Pokretanje servisa. |
| **service** IME_SERVISA **stop** | **systemctl stop** IME_SERVISA | Zaustavljanje servisa. |
| **service** IME_SERVISA **restart** | **systemctl restart** IME_SERVISA | Zaustavljanje pa pokretanje servisa. |
| **service** IME_SERVISA **reload** | **systemctl reload** IME_SERVISA | Ako je podržano, omogućuje samo ponovno učitavanje konfiguracijske datoteke servisa, bez restartanja cijelog servisa. |
| **service** IME_SERVISA **condrestart** | **systemctl condrestart** IME_SERVISA | Restart servisa, ako je servis već pokrenut. |
| **service** IME_SERVISA **status** | **systemctl status** IME_SERVISA | Daje nam status servisa: pokrenut/zaustavljen/greške. |
| **ls /etc/rc.d/init.d/** | **systemctl list-unit-files --type=service** | Daje nam listu servisa koji uopće mogu biti pokrenuti ili stopirani. |
| **chkconfig** IME_SERVISA **on** | **systemctl enable** IME_SERVISA | Uključuje automatsko pokretanje servisa, i nakon restarta računala. |
| **chkconfig** IME_SERVISA **off** | **systemctl disable** IME_SERVISA | Isključuje automatsko pokretanje servisa, i nakon restarta računala. |
| **chkconfig** IME_SERVISA | **systemctl is-enabled** IME_SERVISA | Za provjeru je li servis konfiguriran za automatsko pokretanje nakon restarta računala. |
| **chkconfig --list** | **systemctl list-unit-files --type=service**<br>ili<br>**ls /etc/systemd/system/*.wants/** | Ispisuje tablicu svih servisa i njihovih konfiguraciju za automatsko pokretanje nakon restarta računala. |
| **chkconfig** IME_SERVISA **--list** | **ls /etc/systemd/system/*.wants/**IME_SERVISA**.service** | Za pojedini servis ispisuje njegovu konfiguraciju za automatsko pokretanje nakon restarta računala. |
| | **systemctl show** IME_SERVISA | Ispisuje svojstva servisa. |
| **chkconfig** IME_SERVISA **--add** | **systemctl daemon-reload** | Za dodavanje novog servisa. |
| | **systemctl --failed** | Prikaže servise koji se nisu uspješno pokrenuli. |
| | **systemctl reboot** | Restart računala *[reboot.target]*. |
| | **systemctl poweroff** | Gašenje računala [*poweroff.target*]. |
| | **systemctl emergency** | Postavljane sustava u način održavanja (ekvivalent **init 1**) *[emergency.target]*. |

**Izvori informacija: (431),(432),(K-12)**,**`man chkconfig`**,**`man systemctl`**.

### 7.3.1.2. Rad s naredbom *chkconfig*

**Vratimo se na *RedHat*/*CentOS* 6.x** koji koristi `init` sustav i pripadajuću `chkconfig` naredbu.

Za Linuxe bazirane na *Redhatu* i ***init**-u* postoji i naredba s kojom možemo vidjeti u kojim *runlevelima* se pojedini servis (*daemon)* odnosno skripta koja ga pokreće ili zaustavlja: aktivira ili ne aktivira. Odnosno s ovom naredbom možemo vidjeti koji servisi se pokreću automatski s pokretanjem sustava (Linuxa).

Ova naredba je `chkconfig`.
Možemo ju i sâmo pozvati bez parametara, opcija i prekidača, ili s njima.

Pogledajmo njenu listu opcija:
- `--list` - prikazuje popis svih servisa i njihov status u svim *runlevelima*.
- `--level` - određuje *runlevel* za koji želite provjeriti status servisa.
- `--add` - dodaje servis u *runlevel*.
- `--del` - uklanja servis iz *runlevela*.
- `--on` - aktivira servis u *runlevelu*.
- `--off` - deaktivira servis u runlevelu.

Te pogledajmo i njene prekidače:
- `-a` – aktivira sve servise.
- `-x` – deaktivira sve servise.

Probat ćemo ispisati sve *runlevele* za skriptu (servis) koju smo maloprije gledali; dakle za servis *sshd*:
```
chkconfig --list sshd

sshd 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```
Ovdje vidimo ono o čemu smo pričali gore; pogledajte vrijednosti za *runlevele*: `2`, `3`, `4` i `5`, za koje vidimo kako je servis uključen (`on`).

To znači da će se *SSH* servis pokretati sa sustavom, ako sustav ulazi u *runlevele*: `2`, `3`, `4` i `5`.
S naredbom `chkconfig` moguće je i naknadno mijenjati *runlevele* pri pokretanju skripti odnosno servisa, koji su već na sustavu ili dodavati nove, ali i brisati postojeće.

**Slijede primjeri**
**1.** Želimo li da se *HTTP* poslužitelj (*httpd*) pokreće sa sustavom automatski, to znači kako on mora biti u *runlevelima* `2`, `3`, `4` i `5`
```
chkconfig httpd on
```

Provjera:
```
chkconfig --list httpd

httpd 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

**2.** Ako ipak ne želimo da se *HTTP* poslužitelj uopće pokreće automatski (niti za jedan *runlevel*), tada trebamo napraviti:
```
chkconfig httpd off
```

Sada provjerimo. što smo napravili u prethodnom koraku:
```
chkconfig --list httpd

httpd 0:off 1:off 2:off 3:off 4:off 5:off 6:off
```
Vidimo kako je za sve *runlevele*, naš servis: `httpd` isključen, odnosno vidljivo je da je u stanju: `off`

**3.** U ovom primjeru, kreirali smo svoju novu skriptu, naziva `/etc/rc.d/init.d/moj-servis` koja sadrži `start`, `stop` i `restart` metode te ju želimo dodati pomoći `chkconfig` naredbe, kako bi se pokretala automatski sa sustavom.

Stoga ju prvo dodajmo sa `chkconfig` na sljedeći način:
```
chkconfig --add moj-servis
```

Sada joj dodijelimo prava kako bi se pokretala automatski u *runlevelima* `2`,`3`,`4`,`5`
```
chkconfig moj-servis on
```

Vezano za postojeće skripte odnosno u konačnici servise*;* možemo vidjeti za sve koji su dostupni na sustavu, u kojim *runlevelima* se pokreću, pozivanjem naredbe `chkconfig` na sljedeći način(skratili smo ispis):

```
chkconfig

auditd              0:off   1:off   2:on    3:on    4:on    5:on    6:off
blk-availability    0:off   1:on    2:on    3:on    4:on    5:on    6:off
crond               0:off   1:off   2:on    3:on    4:on    5:on    6:off
ip6tables           0:off   1:off   2:off   3:off   4:off   5:off   6:off
iptables            0:off   1:off   2:off   3:off   4:off   5:off   6:off
iscsi               0:off   1:off   2:off   3:on    4:on    5:on    6:off
iscsid              0:off   1:off   2:off   3:off   4:off   5:off   6:off
lvm2-monitor        0:off   1:on    2:on    3:on    4:on    5:on    6:off
mdmonitor           0:off   1:off   2:on    3:on    4:on    5:on    6:off
multipathd          0:off   1:off   2:off   3:off   4:off   5:off   6:off
netconsole          0:off   1:off   2:off   3:off   4:off   5:off   6:off
netfs               0:off   1:off   2:off   3:on    4:on    5:on    6:off
network             0:off   1:off   2:on    3:on    4:on    5:on    6:off
postfix             0:off   1:off   2:off   3:off   4:off   5:off   6:off
rdisc               0:off   1:off   2:off   3:off   4:off   5:off   6:off
restorecond         0:off   1:off   2:off   3:off   4:off   5:off   6:off
rsyslog             0:off   1:off   2:on    3:on    4:on    5:on    6:off
saslauthd           0:off   1:off   2:off   3:off   4:off   5:off   6:off
sshd                0:off   1:off   2:on    3:on    4:on    5:on    6:off
stinit              0:off   1:off   2:off   3:off   4:off   5:off   6:off
udev-post           0:off   1:on    2:on    3:on    4:on    5:on    6:off
```

Dakle za svaki *runlelevel*, koji predstavljaju stupci s brojevima (`0:`, `1:`, `2:`, `3:`, `4:`, `5:` i `6:`) vidimo, za svaki servis (*daemon*)*,* treba li se pokretati u tom *runlevelu* (`on`) ili ne treba (`off`).

Na ***RedHat/CentOS 6.x*** ili distribucijama Linuxa koje i dalje koriste **init**, ručno pokretanje i zaustavljanje servisa se radi s naredbom `service`, koja ima metode: ***stop***, ***start*** i ***restart***. Primjerice, ako ručno želimo zaustaviti servis ***httpd***, napravimo:

```
service httpd stop
```

Pogledajmo sažetak rada naredbe `chkconfig`:

- Prikazuje trenutni status servisa u svim *RunLevel*-ima.
- Omogućuje aktiviranje ili deaktiviranje servisa u određenom *RunLevel*-u.
- Omogućuje provjeru koji servisi su aktivirani u svim *RunLevel*-ima.



Novije inačice distribucije ***Red Hat*** (**v.7+**) i kompatibilnih distribucija Linuxa (kao i neke druge) više ne koriste (**System V**) **init** sustav, već noviji **systemd** sustav.



Od ***Red Hat*** **v.7+** više se ne koriste niti sljedeće naredbe:

- `chkconfig`
- `service`

**Izvori informacija: (K-12),** `man chkconfig`**.**

### 7.3.2. *Systemd* i sistemski servisi (*daemoni*)

Način na koji su se svi Unix i Linux operativni sustavi pokretali i radili sa servisima je bio pomoću ***`init`*** servisa odnosno *daemona* te pripadajućih *init* skripti i *runlevela*. Međutim s vremenom su otkrivene i neke mane pokretanja servisa, kao i drugih komponenti sustava, pomoću ***`init`*** procesa i sistemskih skripti. Zbog navedenih razloga, tijekom 2010 godine počelo se raditi na zamjeni *init* procesa, ali i metode pokretanja i kontrole raznih komponenti sustava, kao i na razvoju novih servisa.
Posao je većim dijelom završen 2015 godine, kada je uveden takozvani ***Systemd*** sustav.

Naime *Systemd* (`systemd`) osim što je zamjena za `init` proces, nije sve željene promjene mogao odraditi sâm pa su u paketu s njim razvijeni i mnogi pomoćni programi i servisi, njih gotovo sedamdeset (70). Određeni dio Linux distribucija danas, prešlo je sa *init* servisa i sistemskih skripti unutar direktorija `/etc/init.d/` odnosno njihovih poveznica (linkova) u direktoriju `/etc/rc.d/init.d/` i pripadajućim *init runlevel* poddirektorijima `/etc/rc.d/rcX.d/`, na ***systemd*** sustav u kojemu je sâm servis `systemd` sada direktna zamjena za `init` proces. Dakle kod *systemd* više nemamo niti navedenih *init* skripti niti pripadajućih direktorija prema *runlevelu*, kao niti definicije u koji *runlevel* se sustav treba pokrenuti, definirane u datoteci `/etc/inittab`.

Dakle *systemd* kao softverski paket, zamjenjuje inicijalizacijske (*init*) skripte i pripadajuće *runlevel* skripte koje kontrolira tradicionalni ***init*** servis, zajedno sa svim ostalim skriptama koje se izvršavaju pod njegovom kontrolom. *Systemd* također integrira mnoge druge usluge koje su uobičajene na Linux sustavima, poput: upravljanja sustavom za prijavljivanja (*logiranje*) korisnika na sustav, konzolama sustava, prepoznavanje uređaja (***udev*** odnosno `udevd` servis), izvršavanja zadataka u određeno vrijeme (zamjena za `cron` servis), logiranja poruka sustava, postavljanje naziva računala (*hosta*) i druge lokalne postavke.
Kao i ***init*** servis, ***systemd*** je po logici rada servis koji upravlja s drugim servisima.



Primarno je **systemd** prvi servis koji se pokreće tijekom pokretanja sustava, ali i posljednji servis koji ostaje raditi tijekom isključivanja odnosno gašenja sustava. *Systemd* servis je kao korijen procesnog stabla korisničkog prostora procesa (programa). On stoga kao i bivši **init**, ima prvi broj procesa (**PID 1**) i ima posebnu ulogu u Unix/Linux sustavima, jer zamjenjuje roditeljski proces kada izvorni roditelj prestane s radom. Stoga je on kao prvi proces osobito prikladan za praćenje drugih servisa.

U tom području *systemd* više od tradicionalnog pristupa (*init*) pokušava poboljšati: metodu pokretanja, praćenja i rada procesa (programa) koje pokreće, koje je bivši (*init*) mogao pokrenuti i zaustaviti bez praćenja njihovog rada, što je bila jedna od većih slabosti *init* procesa. Naime *init* nije imao metode za stvarnu kontrolu i provjeru rada servisa koji je pokrenuo, već je sâmo na osnovu ***PID*** broja pokrenutog servisa ***pretpostavljao*** da je taj pokrenuti servis i dalje živ te da radi kako treba.

Dodatno *systemd* može izvršavati pokretanje servisa odnosno njegove *startup* sekvence paralelno, što je brže od redoslijednog pristupa tradicionalnog načina pokretanja kod *init* metode rada. S paralelnim pokretanjem dijelova sustava, sustav se vrlo brzo pokreće.

Moguće je i provjeriti koliko je vremena potrebno za pokretanje svake komponente sustava pomoću naredbe `systemd-analyze` i to na sljedeći način:

```
systemd-analyze

Startup finished in 1.138s (kernel) + 1.967s (initrd) + 1min 11.262s (userspace) = 
1min 14.368s
```

Ovdje vidimo koliko je općenito trebalo sustavu da se učita (1 minuta i 14 sekundi), vidljivo prema važnijim cjelinama: konkretno je za učitavanje kernela trebalo 1,138s, te je za inicijalni RAM disk (`initrd`) trebalo još 1.9s, dok je za sve ostale servise i programe bilo potrebno još 1 minuta i 11 sekundi da se učitaju. Na kernelima novijim od **5.12**, s podrškom za tzv. *ACPI Firmware Performance Data Table* (**FPDT**) još je moguće vidjeti i statistike vezane za ***firmware*** uređaja, te vrijeme koje je trebalo za njihovu inicijalizaciju.

Međutim možemo vidjeti i detaljniji ispis, za svaki pojedini servis i komponentu, sa sljedećom naredbom:

```
systemd-analyze blame

         1.027s dracut-initqueue.service
          869ms initrd-switch-root.service
          775ms NetworkManager-wait-online.service
          613ms tuned.service
          520ms firewalld.service
          434ms vdo.service
          374ms sssd.service
          258ms sysroot.mount
          199ms polkit.service
           94ms chronyd.service
```

Slijedi nastavak ispisa (izbacili smo odnosno skratili smo dobar dio ispisa):
```
             86ms smartd.service
             77ms initrd-parse-etc.service
             75ms systemd-journald.service
             72ms systemd-udevd.service
             64ms sys-kernel-debug.mount
             60ms user@0.service
             59ms systemd-logind.service
             56ms systemd-tmpfiles-setup.service
             53ms NetworkManager.service
             38ms systemd-sysctl.service
             29ms sshd.service
```
Ovdje vidimo, koliko vremena (u *ms*) je bilo potrebno svakom pojedinom servisu sustava, da se učita, prilikom pokretanja računala to jest Linuxa. Ove informacije nam mogu biti vrlo koriste u slučaju kada se sustav sporo inicijalizira te trebao pronači uzrok tome. Moguće je i ovu statistiku izvesti odnosno snimiti kao ***SVG*** sliku, s naredbom:
```
systemd-analyze plot > plot.svg
```
Za komunikaciju s procesima i servisima koje je ***systemd*** pokrenuo, koristi se klasična UNIX/Linux ***IPC*** (*Inter-process communication*) metoda komunikacije, preko *Unix socketa* kao i *D-Bus* sustava.

Naime *systemd* prati procese koji koriste kernelov podsustav *cgroups* umjesto korištenja identifikatora procesa odnosno ***PID*** brojeva procesa, kako ih je jedino i mogao pratiti *init* proces. Dakle servisi više nikako ne mogu pobjeći odnosno izgubiti se `systemd` servisu, čak niti pokretanjem pôd procesa niti nekom drugom metodom. Nadalje *systemd* ne samo da koristi *cgroups* mehanizam (kontrolne grupe), nego ih može pratiti i kontrolirati pomoću dvije nove metode odnosno programa: `systemd-nspawn` i `machinectl` koji su zaduženi za upravljanje Linux kontejnera. Od novijih inačica sustav također nudi *ControlGroupInterface*, koji je **API** za Linux *kernel cgroups* mehanizam. Sada pogledajmo strukturu (stablo) procesa, s pogledom na ***systemd*** proces, koju smo dobili s naredbom: `pstree`, a ispis smo skratili zbog lakšeg razumijevanja:
```
pstree -a

systemd --switched-root --system --deserialize 21
  ├─crond -n
  ├─dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
  ├─irqbalance --foreground
  ├─agetty -o -p -- \\u --noclear tty1 linux
  ├─login
  │   └─bash
  ├─rsyslogd -n
  │   └─2*[{rsyslogd}]
  ├─sshd -D
  │   ├─sshd
  │   │   └─bash
  │   └─sshd
  │       └─bash
  │           └─pstree -a
  ├─systemd-journal
  ├─systemd-udevd
  ├─systemd-logind
```
Naredba `pstree`, dolazi u softverskom paketu `psmisc`, pa ako ju nemamo, možemo ju instalirati sa:
```
yum -y install psmisc
```
U gornjem ispisu je jasno vidljivo kako `systemd` proces pokreće sve ostale servise, kao i sve programe (procese) na sustavu. Naime ***systemd*** proces se sâm pokreće kao servis isto kao što je bio i slučaj sa `init` procesom.
Međutim ***systemd*** nema strukturu direktorija poput ***init*** procesa (Unix *System V* inicijalizacije) u kojem su se na osnovi *runlevel* broja s kojim se trebao pokrenuti sustav, i pokretale skripte koje se nalaze u direktorijima unutar glavnog vršnog direktorija za sve *runlevel-e*: `/etc/rc.d/`. Kod *systemd* je ovaj sustav direktorija (mapa) i način rada malo drugačiji.
Naime osnovni direktorij za inicijalizacijske odnosno takozvane ***unit*** *skripte* se nalazi u strukturi datoteka i poddirektorija s pripadajućim datotekama unutar vršnog direktorija: `/etc/systemd/system/`.



Kod **systemd** sutava za rad sa servisima koristi se naredba **`systemctl`**, a primjere njene upotrebe pogledajte u: **7.3.1.1. *Init* sistemski servisi (*daemoni*) detaljnije** i to u tablici na kraju poglavlja (**usporedna tablica: init vs. systemd.**).

Druge primjere upotrebe naredbe `systemctl` vidjet ćete u poglavljima koja slijede.

**Izvori informacija: (421),(430), `man systemd`,`man systemctl`,`man pstree`,`man systemd-analyze`.**

### 7.3.2.1. *Unit* datoteke

***Slijedi napredna cjelina!***

***Systemd*** kao inicijalizacijske datoteke više ne koristi *shell* skripte, kao što je koristio ***init*** proces, već koristi posebne takozvane „*Unit*“ datoteke, koje koriste posebnu sintaksu i naredbe s kojima se željenom servisu daju određene instrukcije kako se treba pokrenuti, zaustaviti, restartati i slično, poput *init* skripti, ali na potpuno drugačiji način. Dakle korištenjem posebne sintakse u novom deklarativnom jeziku *systemd-a*.
*Unit* datoteke mijenjaju ono što su bile *init* skripte u direktoriju `/etc/init.d/` odnosno *init runlevel* skripte unutar nekog od pôd direktorija unutar `/etc/init.d/` direktorija.

Sada možemo zaboraviti na stari *init* i način inicijalizacije sustava i *init* skripti kako smo ih do sada poznavali. *Unit* skripte odnosno *systemd* inicijalizacijske skripte za svaki pojedini servis se nalaze u posebnim konfiguracijskim datotekama, slično kao što je bilo s *init* skriptama. *Unit* skripte *systemd* promatra kao objekte koji predstavljaju sistemske resurse za koje je zadužen određeni *servis*. Ovdje postoji još jedna podjela, a to je podjela prema namjeni *unit* skripti; pa su ovdje moguće razne kategorije namjena: za servise, mrežne resurse, točke montiranja datotečnog sustava (*mount points*) i slično.

Ove (*unit*) konfiguracijske datoteke se izvorno nalaze u direktoriju `/lib/systemd/system/` i njegovim pôd direktorijima, a prema njima postoje simbolički linkovi u vršnom direktoriju `/etc/systemd/system/`.

Pogledajmo sve kategorije namjena *unit* datoteka, u kojima praktično ekstenzija datoteke predstavlja kategoriju kojoj pripada *unit* datoteka:

- **`service`** – namijenjena je za standardne servise (*daemone*) odnosno za rad s procesima koje kontrolira *systemd*; ekstenzija im je `.service`
- **`socket`** – namijenjena je za *unit* datoteke s kojima s kojima se može baratati sa: mrežnim *socketima*, *unix socketima ili* datotečnim *FIFO socketima*. Ekstenzija im je `.socket`
- **`device`** – namijenjena je za uređaje (hardver). Ekstenzija im je `.device`
- **`mount`** – namijenjena je za točke montiranja kojima upravlja *systemd*. Ekstenzija im je `.mount`
- **`automount`** – namijenjena je za automatsko *montiranje* datotečnih sustava kojima upravlja *systemd*. Ekstenzija im je `.automount`
- **`swap`** – namijenjena je za *swap* uređaje, kojima upravlja *systemd*. Ekstenzija im je `.swap`
- **`path`** – namijenjena je za *unit* datoteke s kojima možemo pratiti određenu putanju direktorija (*path*) za odrađivanje nekih radnji koje će se pokrenuti od strane *systemd*. Ekstenzija im je `.path`
- **`timer`** - namijenjena je za *unit* datoteke s kojima možemo raditi s vremenski kontroliranim uvjetima, kojima konrolira *systemd*. Ekstenzija im je `.timer`
- **`snapshot`** - namijenjena je za snimanje stanja rada *systemd* servisa (*snapshot*). Ekstenzija im je `.snapshot`
- **`slice`** – namijenjena za *unit* datoteke s kojima možemo raditi s grupom procesa ili procesima. Ekstenzija je `.slice`
- **`scope`** – namijenjena je za *unit* datoteke s kojima možemo raditi sa procesima na sustavu. Ekstenzija im je `.scope`

Listu svih *unit* datoteka na sustavu možemo dobiti s naredbom: `systemctl`, na sljedeći način:

```
systemctl list-unit-files
```

Odnosno, ako želimo izlistati samo određenu vrstu, recimo samo one koji su u kategoriji: `service`, napravimo to ovako:

```
systemctl list-unit-files --type service
```



Unit datoteke za **systemd** sustav na Linuxu su konfiguracijske datoteke koje definiraju i upravljaju resursima koje **systemd** pokreće ili kontrolira. Ove datoteke specificiraju kako bi određeni resurs (servis) trebao biti pokrenut, zaustavljen ili ponovno pokrenut, kao i ostale parametre vezane uz njegovo ponašanje.

**Izvori informacija:** **`man systemctl`**, **`man systemd`**.

### 7.3.2.2. *Systemd* - rad sa servisima (*daemonima*)

***Slijedi napredna cjelina!.***
Rad sa servisima svodi se na rad sa *unit* datotekama koje su zadužene za njih, dakle one koje su u kategoriji *service*.
I za ovu namjenu, koristimo naredbu `systemctl` koja mijenja funkcionalnosti starih naredbi: `service` i `chkconfig`.

Pogledajmo koje sve operacije možemo raditi sa servisima, odnosno koje metode oni podržavaju:

- `start` – za trenutno pokretanje servisa.
- `stop` – za zaustavljanje već pokrenutog servisa.
- `status` – za ispis statusa (stanja rada i drugih poruka) trenutno pokrenutog servisa. On standardno prikazuje informacije o statusu svih pokrenutih servisa na cijelom sustavu ili o jednom ili više servisa, ako navodimo ime servisa.
- `restart` – za restart trenutno pokrenutog servisa.
- `reload` – ova naredba nalaže servisu da ponovno pročita i učita svoju konfiguraciju s postavkama, bez restarta samog servisa (ako servis to uopće podržava). Dakle servis će prilikom pozivanja ove neradbe i dalje zadržati isti **PID** broj. To je korisno, ako smo mijenjali konfiguraciju servisa, a ne želimo restartati servis zbog toga, već samo želimo da već pokrenuti servis (učitan u RAM memoriju) ponovno pročita novu konfiguraciju i aktivira promijene definirane u njoj.

Postoje i posebne dvije opcije (zamjena za staru `chkconfig` naredbu):

  - `enable` – trajno **UKLJUČI** ovaj servis; za automatsko pokretanje tijekom svakog pokretanja sustava.
  - `disable` – trajno **ISKLJUČI** ovaj servis; za automatsko pokretanje tijekom svakog pokretanja sustava.

Za primjer provjerimo status *sshd* servisa (sintaksa je: ***systemctl OPERACIJA IME-SERVISA***)

```
systemctl status sshd

● sshd.service - OpenSSH server daemon
Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2018-10-11 09:58:14 UTC; 1 weeks 6 days ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 47067 (sshd)
   CGroup: /system.slice/sshd.service
           └─47067 /usr/sbin/sshd -D
```

Iz ispisa osim primjerice PID broja procesa (`47067`) vidimo i nama dvije zanimljive stvari:

- ***Sshd*** servis je pokrenut (`Loaded: loaded`) i aktivan je (`Active: active`).
- ***Sshd*** servis je trajno uključen (`enabled`), kod svakog pokretanja ili restarta (Linux) sustava.

Ako ipak želimo privremeno zaustaviti ovaj servis, to možemo napraviti sa (*za ovakve radnje je potrebno biti root korisnik*):

```
systemctl stop sshd
```

Odnosno, ako ga ponovno želimo pokrenuti, to možemo učiniti sa sljedećom naredbom:

```
systemctl start sshd
```

Moguće je i vidjeti o kojim servisima ovisi naš željeni servis (konkretno nas zanima *sshd* servis):

```
systemctl list-dependencies --before sshd.service
```

I potom ćemo dobiti listu servisa/*daemona* koji se moraju pokrenuti prije njega (*sshd*).

Isto tako možemo vidjeti i koji se sve servisi pokreću nakon njega, sa sljedećom naredbom:

```
systemctl list-dependencies --after sshd.service
```

Isto tako možemo dobiti i više detalja o našem pokrenutom *sshd* servisu:

```
systemctl show sshd.service
```



Za druge primjere upotrebe naredbe **`systemctl`** pogledajte tablicu na kraju poglavlja:
**7.3.1.1. *Init* sistemski servisi (*daemoni*) detaljnije**.



U slučajevima kada smo:

- Ručno mijenjali neku **Unit** datoteku.
- Dodavali nove **Unit** datoteke.
- Brisali **Unit** datoteke.
- Mijenjali simboličke poveznice na **Unit** datoteke.

**Obavezno je potrebno pokrenuti osvježavanje systemctl servisa sa sljedećom naredbom:**

```
systemctl daemon-reload
```

**Sada pogledajmo i *unit („jediničnu“)* datoteku za naš *sshd* servis, koja se nalazi u datoteci**: `/lib/systemd/system/sshd.service`:

```
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s
RestartPreventExitStatus=255

[Install]
WantedBy=multi-user.target
```

Objasnit ćemo odjeljak po odjeljak odnosno sekciju po sekciju ove datoteke:

**`[Unit]:`** ovaj odjeljak sadrži generičke opcije koje ne ovise o vrsti *unit-a*. Te opcije pružaju opis *unit-a* (jedinice), navode njegovo ponašanje i postavljaju ovisnosti o drugim jedinicama (*unit-ima*). Pogledajmo što mi sve imamo u našoj konfiguraciji:

- `Description=` ovo je opis servisa.
- `Documentation=` ovdje je opisano u kojim *man* stranicama se nalazi dokumentacija ovog servisa.
- `After=` ovdje se navode drugi servisi koji se moraju pokrenuti prije nego se pokrene ovaj servis*;* u ovom slučaju su to: `network.target` (mrežna komponenta) i `sshd-keygen.service` (servis za generiranje *SSH* ključeva).
- `Wants=` ovdje je opisano koji od *unita* se treba učitati prije ovog našeg *unita* odnosno servisa.
- Osim opcije `Wants=` moguća je i opcija `Requires=` koja je restriktivnija jer traži **obavezno** učitavanje *unita* odnosno servisa prije inicijalizacije našeg servisa.

**`[Service]:`** ovaj odjeljak naziva se prema kategoriji *unit-a*, pa kao u našem slučaju ako se radi o kategoriji „`Service`“, ovaj odjeljak ima naziv `[Service]`, dok za druge kategorije ima naziv prema kategoriji (`Service`, `Socket`, `Device`, …). Naime, ako *unit* ima specifične postavke ili direktive, one su grupirane u ovom odjeljku (sekciji).
U našem slučaju imamo:

- `Type=` ovo je kategorija u kojoj se definira metoda koja se koristi tijekom pokretanja servisa. U našem slučaju: `notify` znači kako se od servisa očekuje da će slati sistemsku *notification* poruku, kada se pokrene. Tek nakon što *systemd* primi ovu poruku, nastavit će s drugim *unit* datotekama koje se eventualno trebaju pokrenuti.
- `EnvironmentFile=` ovdje je definirana koja je inicijalna konfiguracijska datoteka našeg *sshd* servisa. U našem slučaju je to datoteka imena: `/etc/sysconfig/sshd`. O ovim datotekama ćemo govoriti u poglavlju: **7.4 Upravljanje konfiguracijom servisa**
- `ExecStart=` ovdje je navedena izvršna datoteka s opcijama i parametrima s kojima se pokreće naš servis *ssh*. *Moguće je imati i* `ExecStop=` *(izvršna datoteka) koji se pokreće kod metode STOP.*
- `ExecReload=` u ovom dijelu se navodi naredba s kojom se pokreće „*reload*“ servisa, a koja može imati i više argumenata i opcija. *Reload* se odnosi na metodu brzog ponovnog pokretanja servisa koji je već pokrenut.
- `KillMode=` u ovom dijelu se navodi kako se proces ovog servisa ubija. U našem slučaju, ako je postavljeno na `process`, to znači da se ubija samo glavni, roditeljski proces. Postoje i druge mogućnosti ubijanja, poput `mixed` koji ubija glavni (roditeljski) proces, ali šalje i signale za *terminaciju* svih njegovih pôd procesa (pod programa).
- `Restart=` ovo je postavka koja govori *systemd* servisu treba li restartati servis (*sshd u našem slučaju*), ako naš servis (*sshd*) prestane s radom, bude ubijen (od bilo čije strane) ili ako se zaustavi zbog nekog isteka vremenskog okvira. Ako je naš servis (*ssh*) zaustavljen od strane *systemd*, onda se ova radnja neće odraditi. Ovdje mogu biti postavljene mnoge vrijednosti od kojih svaka ima svoj način ponašanja. Primjerice opcija koju imamo ovdje (`on-failure`) označava kako će se proces automatski restartati u bilo kojem od slučajeva: ako se proces sruši zbog bilo kojeg od razloga ili ako mu istekne neki vremenski okvir za izvršavanje (ako postoji) ili ako ga *watchdog* servis zaustavi.
- `RestartSec=` označava vrijeme u sekundama, koje će se čekati prije nego se aktivira radnja i opcije prije (`Restart=`) odnosno prije nego se uopće krene u proceduru restarta.
- `RestartPreventExitStatus=` označava sistemski statusni „*exit status*“ kôd, našeg procesa odnosno servisa (*u našem slučaju sshd)* koji ako se dobije, na njega se neće reagirati sa restartom, ovdje može biti i lista *statusnih* kôdova.
- Ako imamo potrebu da se servis pokreće s drugim korisničkim pravima (ne ***root***); koristimo: `USER=XX` i `GROUP=YY`.

**`[Install]:`** ovaj odjeljak sadrži informacije o aktivaciji ovog *unit*-a, koja se koristi pomoću programa `systemctl` kada se određeni *unit* (servis u konačnici) želi aktivirati ili deaktivirati odnosno automatski pokretati s podizanjem sustava ili ne. Primjerice: `WantedBy=multi-user.target` označava stanje sustava u kojem su pokrenuti svi mrežni servisi.

**Izvori informacija:(421),(422),(423),(430),(431),(432),(K-14), man: `systemd.snapshot`,`systemd.slice`, `systemd.scope`,`systemd.service`,`systemd.socket`,`systemd.device`,`systemd.mount`,`systemd.automount`,`systemd.swap`,`systemd.path`,`systemd.timer`,`systemd.unit`,`systemd.target`.**

### 7.3.2.3. Osnovni servisi (*daemoni*) unutar *systemd* softverskog paketa

***Slijedi napredna cjelina!***
***Systemd*** dolazi u paketu s gotovo 70 programa i servisa, koji mijenjanju ***init*** i neke od servisa koji su dolazili uz njega.
Ovdje ćemo se pozabaviti osnovnim servisima koji dolaze u ***systemd*** paketu programa, a mijenjaju servise koji su dolazili kao standardni uz (stariji) *init*, kao i one nove.

Ono što smo standardno dobili od osnovnih sistemskih servisa su sljedeći servisi:

`systemd-journald` - ovo je takozvani *journaling servis*, koji je zadužen za prikupljanje, spremanje i prikaz sistemskih poruka. On nadopunjuje, a nekada i mijenja servise poput: ***syslog-ng*** ili ***rsyslog***. Naime sve poruke sustava sada se spremaju u *journal* log datoteke u posebnom binarnom formatu. Njegova konfiguracija se nalazi pohranjena u datoteku*: `/etc/systemd/journald.conf`. Ove log datoteke se standardno konstantno zapisuju na disk, a one starije i arhiviraju. Sve *log* poruke možemo vidjeti s naredbom: `journalctl`. Pogledajmo samo neke opcije odnosno mogućnosti ove naredbe:

Ispiši sve log poruke, prikazane u ***UTC*** (*GMT*) vremenu; nultoj vremenskoj zoni *(Hrvatska je u* ***UTC+1*** *vremenskoj zoni)*:
```
journalctl --utc
```

Ispiši sve *log* poruke tijekom pokretanja sustava (tijekom *boot* procesa):
```
journalctl -b
```

Ispiši sve *log* poruke u vremenu od 10:00.h. do prije sat vremena:
```
journalctl --since 10:00 --until "1 hour ago"
```

Ispiši apsolutno sve *log* poruke od jučer (engl. *Yesterday*):
```
journalctl --since yesterday
```

Moguće je pregledavati poruke točno određenog servisa; primjerice ispišimo sve *log* poruke ***sshd*** servisa (*sshd unit-a*):
```
journalctl -u sshd.service
```

Ako u navedenoj konfiguracijskoj* datoteci imamo postavku: `Storage=persistent`, to znači da se sve log poruke snimaju na disk, obično u direktorij: `/var/run/log/journal/`. Zadana granica veličine postavljena je na vrijednost od 10% veličine osnovnog datotečnog sustava na kojem se nalazi ovaj direktorij, ali dodatno ograničena na 4 GB maksimalno. U slučaju kada nam je prostor na disku prepunjen s log porukama (do maksimuma od 4GB), možemo ove poruke jednokratno očistiti.

Primjerice obrisati sve starije od 10 dana, te eventualno očistiti `journald` na primjerice maksimalno 2GB prostora:
```
journalctl --vacuum-time=10d ;  journalctl --vacuum-size=2G
```
Za trajnu promjenu ograničenja na primjerice 2GB prostora, u konfiguracijsku datoteku* dodajte: `SystemMaxUse=2G`.



Za druge detalje oko ***journald*** servisa i pripadajuće naredbe ***journalctl***, pogledajte poglavlje:
**17.5. Journald, journalctl i dnevničke (log) datoteke.**

`systemd-logind` - ovo je servis **(424)** je koji zadužen za rad sa prijavljivanjem (*logiranjem*) korisnika na sustav odnosno on se brine o svim radnjama vezanim za korisničke račune: od trenutka kada se korisnik logira (spoji) na sustav i unese svoje korisničko ime i lozinku, do praćenja svih njegovih procesa i njegovih sesija kao i pristupa korisnika uređajima. Statuse i dodatne informacije od ovog servisa možemo dobiti s naredbom `loginctl`.

Pogledajmo i nekoliko primjera.
Ispišimo sve sesije trenutno logiranog korisnika; ovdje ćemo vidjeti sve procese koje je trenutni korisnik pokrenuo, uz dodatne detalje:
```
loginctl session-status

89570 - root (0)
           Since: Thu 2018-10-25 09:15:29 UTC; 6min ago
           Leader: 32028 (sshd)
           Remote: 192.168.1.240
           Service: sshd; type tty; class user
           State: active
           Unit: session-89570.scope
                 ├─32028 sshd: root@pts/0
                 ├─32030 -bash
```

`systemd-networkd` - ovo je mrežni servis koji upravlja mrežnim konfiguracijama. Ovaj servis može biti osobito koristan za postavljanje složenih mrežnih konfiguracija za linux kontejnere ili za virtualna računala, na distribucijama Linuxa koje ga koriste.



Ovaj servis standardno se **ne pokreće** s **Red Hat** 6.x - 9.x, već se za mrežu koriste klasične metode konfiguracije i rekonfiguracije pomoću servisa: **network** (Red Hat 6-8) ili **NetworkManager** (Red Hat 8+).

`systemd-tmpfiles` - ovo je servis koji vodi brigu o čišćenju privremenih datoteka i direktorija. Obično se pokreće prilikom pokretanja sustava, a zatim u određenim intervalima.

Ako želimo vidjeti njegov status, možemo pokrenuti:

```
systemctl status systemd-tmpfiles-clean.timer
● systemd-tmpfiles-clean.timer - Daily Cleanup of Temporary Directories
Loaded: loaded (/usr/lib/systemd/system/systemd-tmpfiles-clean.timer; static; vendor
preset: disabled)
Active: active (waiting) since Fri 2018-10-05 11:30:31 UTC; 2 weeks 6 days ago
Docs: man:tmpfiles.d(5)
      man:systemd-tmpfiles(8)
```

Ovaj *unit* je vrste *timer* jer je vremenski ovisan odnosno mora se pokretati u određeno vrijeme. Vidimo da se ovaj servis, osim tijekom pokretanja sustava pokreće i svaki dan, kako bi čistio privremeno kreirane datoteke, vidljivo iz njegove *unit* datoteke:

```
cat /usr/lib/systemd/system/systemd-tmpfiles-clean.timer

[Unit]
Description=Daily Cleanup of Temporary Directories
Documentation=man:tmpfiles.d(5) man:systemd-tmpfiles(8)

[Timer]
OnBootSec=15min
OnUnitActiveSec=1d
```

U odjeljku `[Timer]` vidimo kako se ovaj servis pokreće 15 minuta nakon pokretanja sustava (`OnBootSec=15min`) te da se aktivira svaki dan nakon toga (`OnUnitActiveSec=1d`).

- `systemd-timedated` - ovo je servis koji se može koristiti za kontrolu vremenskih postavki, kao što su vrijeme sustava, vremenska zona sustava ili odabir između *UTC* i lokalnog sata unutar vremenske zone.
- `systemd-udevd` - ovo je zamjena za klasični linuxov servis `udevd`, a koji je potpuno integriran u ovaj novi servis. On je osnovni menadžment servis za sav hardver (uređaje) u linuxu, koji je zadužen za prepoznavanje i inicijalizaciju svôg hardvera i u konačnici kreiranja strukture posebnog direktorija s pripadajućim datotekama koje predstavlja sâv hardver odnosno uređaje u linuxu. Dakle direktorija: `/dev/` te pripadajućih *device* datoteka.
- `systemd-boot` - ovo je takozvani ***boot*** menadžer zadužen za pokretanje sustava, koji koristi *systemd.*

## *Targets* (odredišta)

Do sada nismo pričali o *runlevelima*, koji su kako smo spomenuli stanja rada svakog Unixa odnosno Linuxa koji koristi *init* metodu odnosno ***System V*** način inicijalizacije i rada sustava. Kod ovakvog rada, sustav je mogao biti sâmo u jednom stanju rada odnosno samo u jednom *runlevelu*. Prelaskom na *systemd*, više nema klasičnih *runlevela,* odnosno oni više ne postoje na način na koji su prije postojali. *Systemd* je uveo pojam ***Targets*** odnosno ciljeve, koji predstavljaju sličan način rada u kojem se sustav postavlja u željeno stanje rada. To radi tako da se servisi preko svojih *unit* inicijalizacijskih datoteka mogu grupirati u jedan vršni „*cilj*“ odnosno *Target*.
Pri tome *Target* predstavlja logičku grupu servisa koji će se pokrenuti, kada je sustav postavljen da se inicijalizira prema tom *cilju* odnosno *Targetu.*

Pogledajmo usporednu tablicu koja donekle pokazuje ekvivalente između *init runlevela* i *systemd Targeta*:

| Systemd targeti | Ekvivalent init runlevelima |
|---|---|
| `poweroff.target` | 0 |
| `rescue.target` | 1 |
| `multiuser.target` | 2 |
| `multiuser.target` | 3 |
| `multiuser.target` | 4 |
| `graphical.target` | 5 |
| `reboot.target` | 6 |

Prema potrebi je dodatno moguće i da više *Targeta* može biti aktivno u isto vrijeme, što nema analogiju u starom *init* sustavu.

Kako biste vidjeli sve ciljeve (*Targete*) dostupne na vašem sustavu, pokrenite naredbu `systemctl` na sljedeći način:

```
systemctl list-unit-files --type=target
```

Vidimo kako imamo definiran cijeli niz *Targeta* odnosno grupa *unit-a*, koje možemo koristiti.
Logično je i vidljivo da su i *Targeti* zapravo *unit-i* odnosno inicijalizacijske datoteke servisa/*daemona*, što je točno.

Pogledajmo koji *Target* poziva naš sustav tijekom inicijalizacije:

```
systemctl get-default

multi-user.target
```

Vidimo kako se poziva `multi-user.target` koji je ekvivalent *init 3 runlevelu*, a koji pokreće sve više-manje standardne servise odnosno *daemone*.

Možemo vidjeti i ekvivalent *init runlevelu*, za konkretan sustav očekujemo da je to *init runlevel* 3, ali to ipak provjerimo s naredbom `runlevel` koja će nam dati ekvivalent *runlevel* broju u kojem se sustav trenutno nalazi:

```
runlevel
N 3
```

Sada pogledajmo kako izgleda *systemd* inicijalizacijska datoteka `multi-user.target` (izbacili smo komentare, da nam ne zauzimaju mjesto):

```
[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
```

Odjeljak **`[Unit]:`** standardno sadrži generičke opcije koje ne ovise o vrsti *unit-a*. Te opcije pružaju opis *unit-a* (jedinice), navode njegovo ponašanje i postavljaju ovisnosti o drugim jedinicama (*unit-ima*). Pogledajmo što mi sve imamo u našoj konfiguraciji, a pri tome nećemo ponovno opisivati one opcije koje smo spomenuli prije nekoliko stranica (`Description`, `Documentation`, `Requires` i `After`).

Dakle ovdje vidimo nove opcije:

- `Conflicts=` ovdje može biti lista *unita*, koji se ne smiju pokretati u isto vrijeme kada i ovaj naš *unit.*
- `AllowIsolate=` ako je postavljeno u `yes`, a što nije normalna postavka normalnih *unita* koji nisu *Target uniti*, to označava kako se za ovaj cijeli *unit* (inicijalizacijska *systemd* datoteka) trebaju pokrenuti svi *uniti* (*servisi/daemoni*) koji su označeni da se moraju ili trebaju pokrenuti (engl. *dependencies*). Svi ostali koji nisu navedeni se trebaju zaustaviti. Ovo je praktično znak da se radi o *Target unitu* koji je ekvivalent *runlevelu* prema *init* terminologiji.

Sada kada smo saznali kako je naš standardni *Target unit*: `multi-user.target` pogledajmo kako saznati, koje on sve ostale *unite* odnosno servise (*daemone*) pokreće:

```
systemctl list-dependencies multi-user.target
```

Dobit ćemo ispisano stablo u kojem će zeleno (●) biti označeni oni servisi/*daemoni* koji se pokreću, a crveno (●) oni koji se ne pokreću.

Ako zbog nekog razloga želimo promijeniti naš standardni *Target unit*, to možemo napraviti sa slijedećom sintaksom: `systemctl set-default <ime.target>`.

Odnosno, ako želimo promijeniti stvarni ekvivalent *init runlevelu*, moramo koristiti *isolate* opciju poput:
`systemctl isolate set-default <ime.target>`

**Izvori informacija: (425),(426),(427),(428),(429), `man journalctl`,`man systemctl`,`man systemd-journald`.**

### 7.3.2.4. Kreiranje vlastite Systemd servisne skripte (sistemskog servisa)

***Slijedi napredno poglavlje!***
U određenim slučajevima moguće je imati potrebu i za kreiranjem vlastite ***Systemd*** servisne skripte odnosno sistemskog servisa. To je relativno jednostavno. Pogledajmo i kako.



Prvo se podsjetite ***unit*** datoteka u poglavlju:
**7.3.2.1. *Unit* datoteke.**

Kreirat ćemo jednu *Systemd* skriptu odnosno servisnu datoteku imena: `/etc/systemd/system/status.service`.
Važno je da se ova datoteka nalazi u direktoriju (mapi): `/etc/systemd/system/` te da je izvršna, pa to i napravimo:

```
touch /etc/systemd/system/status.service
chmod +x /etc/systemd/system/status.service
```

Naša navedena *systemd* skripta će nakon svakog urednog pokretanja sustava (Linuxa) zapisati ispis nekoliko naredbi, od kojih ćemo dobiti neke osnovne statistike o sustavu (slobodno mjesto na disku i ispis mrežnih sučelja), pa će ona sadržavati sljedeće:

```
[Unit]
Description=Osnove statistike sustava
After=network.target

[Service]
Type=oneshot
ExecStart=/usr/bin/date
ExecStart=/usr/sbin/ip addr show
ExecStart=/usr/bin/df -h
RemainAfterExit=no
StandardOutput=journal+console

[Install]
WantedBy=multi-user.target
```

Na početku odjeljka: `[Unit]` smo definirali opis našeg servisa/skripte, unutar polja: `Description=`.
Potom smo definirali da će se naša skripta pokrenuti tek nakon što su se inicijalizirali svi mrežni servisi, što je definirano u: (`After=network.target`).
Dakle mrežni servisi su definirani u odredištu (*target-u*): `network.target`.

U odjeljku `[Service]` smo definirali da se ova skripta/servis izvršava samo jednom: `Type=oneshot`.
Potom smo konačno definirali naredbe koje se trebaju pokretati, svaka od njih u svom polju: `ExecStart=`.



Važno je znati kako kod naredbi koje pozivamo u **`ExecStart=`** ne možemo koristiti preusmjeravanje:
odnosno znakove: `<`, `>`, `<<` ili `>>`.

Zatim imamo definirano: `RemainAfterExit=no` što znači kako ovaj servis (skripta) neće nastaviti s radom nakon što se izvrše sve naredbe u njoj, odnosno kako ona nije klasičan servis na koji se oslanja neki drugi servis ili skripta.

I na kraju ove sekcije imamo definirano gdje će se zapisivati ispis svih ovih naredbi: `StandardOutput=journal+console` što konkretno znači da će se ispis naredbi koje pokrećemo vidjeti u terminalu, kao i u *journal* log datoteci.

***Journal*** log datoteku za naš servis (`status.service`) možemo gledati sa sljedećom naredbom:

```
journalctl -u status.service
```

Osim ispisa standardnog izlaza naredbe u terminal (`console`) ili *journal*/log datoteku (`journal`), možemo preusmjeravanje raditi i drugdje; primjerice: *nigdje* (`null`), u *syslog* (`syslog`), u *kernel log buffer* (`kmsg`), ili u specifično definiranu datoteku na disku (`file:`**`XY`**) pri čemu je **`XY`** puna putanja do te datoteke (uključujući i nju).

Nadalje, osim definiranja gdje će se zapisivati standardni izlaz (*STDOUT*), možemo definirati i gdje će se snimati standardna greška (*STDERR*), korištenjem ključne riječi: `StandardError=`.

Na kraju imamo odjeljak: `[Install]` u kojem definiramo kako nam je potrebno da se i cijeli sustav inicijalizira prije nego se naše naredbe navedene u skripti počnu izvršavati; što smo postavili sa: `WantedBy=multi-user.target`.

Sada možemo pokrenuti naš *systemd* servis odnosno skriptu:
```
systemctl start status.service
```

Odnosno trajno ju možemo aktivirati sa sljedećom naredbom:
```
systemctl enable status.service
```

***U slučaju kada mijenjate Systemd skriptu, potrebno je naložiti sustavu da osvježi njene unose (za našu skriptu), sa:***
```
systemctl reenable status.service
```



Vidjeli smo da se za rad sa servisima koristi naredba **`systemctl`** čije dodante primjere upotrebe pogledajte u: **7.3.1.1. *Init* sistemski servisi (*daemoni*) detaljnije** i to u tablici na kraju poglavlja (**usporedna tablica: init vs. systemd**).

**Izvori informacija: (427),(428),(429),(933),(934),** **`man systemctl`**, **`man systemd.exec`**, **`man systemd.service`**.

## 7.3.2.5. Drugi korisni programi i komponente unutar *systemd* paketa

***Slijedi napredno poglavlje (7.3.2.5.x)!***
Pogledajmo i druge korisne programe i komponente unutar *Systemd* paketa programa, u cjelinama koje slijede.

### 7.3.2.5.1. Aktiviranje `/etc/rc.local` datoteke

Standardno, na sustavima koji koriste *init* način rada, a to su: ***RedHat/CentOS*** 6.x, ***UNIX***-i kao i druge distribucije Linuxa koje ga koriste, odnosno prema Unix System V definiciji, nakon što se pokrenu svi potrebni servisi i komponente sustava, sustav pokreće skriptnu datoteku imena: `/etc/rc.local`.

Međutim ***Systemd*** više ne koristi ovu datoteku koja je vrlo korisna u slučajevima kada želite odraditi jednostavne promjene na sustavu: pokrenuti neku jednostavnu skriptu ili odraditi bilo koju drugu radnju tek nakon što su se pokrenule sve komponente i servisi sustava; dakle nakon što je cijeli sustav (Linux) uredno pokrenut.

Ova funkcionalnost odnosno datoteka se intenzivno koristila, ali ju više ne možemo koristiti (za ***RedHat/CentOS 7.x***). Ipak, moguće ju je aktivirati i na sustavima koji koriste ***Systemd*** sustav.

Prvo ju kreirajmo i promijenimo joj ovlasti da postane izvršna; sve radimo kao ***root*** korisnik jer je i namjena upotrebe ove datoteke za administratore sustava:
```
touch /etc/rc.local
chmod +x /etc/rc.local
```

Sada moramo pokrenuti poseban ***Systemd*** servis koji je zadužen za nju:
```
systemctl start rc-local.service
```

I u slijedećem koraku trajno aktivirajmo ovaj servis (da se pokreće tijekom svakog pokretanja računala):
```
systemctl enable rc-local.service
```

I to je sve što je potrebno da možemo koristiti datoteku: `/etc/rc.local` na koju smo već navikli, a koja je stvarno korisna!.



Na ***RedHat/CentOS*** 8.x+ više nije potrebno pokretati servis: `rc-local.service`, već je dovoljno samo imati ovu datoteku s izvršnim ovlastima (***x***). Dakle od ***RedHat/CentOS*** 8.+ je vraćena stara funkcionalnost upotrebe ove skripte.
**Ipak, ako koristite *Systemd*, preporuka je umjesto ove datoteke kreirati svoju Systemd servisnu skriptu!**

**Izvori informacija: (932),(933)**,**`man systemctl`**.

#### 7.3.2.5.2. Podešavanje sistemskog vremena i vremenske zone

Podešavanje sistemskog vremena i vremenske zone, moguće je pomoću naredbi koje nam dolaze u *systemd* softverskom paketu. Ovdje se konkretno radi o naredbi: `timedatectl`.



Za klasično konfiguriranje Linux sustava, po pitanju postavki vremena i vremenske zone pogledajte poglavlje: **10.4 Postavke vremenske zone i sata te regionalne postavke.**

U slučaju kada želimo vidjeti koje sve postavke vremena, datuma i vremenske zone imamo trenutno postavljene na sustavu, možemo pokrenuti sljedeću naredbu:

```
timedatectl

      Local time: Fri 2018-10-26 10:43:42 UTC
  Universal time: Fri 2018-10-26 10:43:42 UTC
        RTC time: Fri 2018-10-26 10:43:42
       Time zone: Etc/UTC (UTC, +0000)
     NTP enabled: yes
NTP synchronized: no
 RTC in local TZ: no
      DST active: n/a
```

U ispisu vidimo sljedeće:

- Trenutno vrijeme je vidljivo pod: `Local time:`
- Univerzalno (svjetsko) vrijeme (*UTC*) je vidljivo pod: `Universal time:`
- Vrijeme koje je trenutno na *RTC* satu (na matičnoj ploči) je vidljivo pod: `RTC time:`
- Vremenska zona je vidljiva pod: `Time zone:`
- Ako koristimo *NTP* poslužitelj za dohvaćanje točnog vremena, to ćemo vidjeti kao: `NTP enabled: yes` mada to možemo postići sa konfiguracijom *NTP* servisa (o tome kasnije). Ako ovdje ipak želimo uključiti *NTP* klijenta odnosno mogućnost da se naše računalo sinkronizira s referentnim *NTP* poslužiteljem. Uključivanje *NTP* opcije ovdje znači aktiviranje servisa: `chronyd` ili `ntpd`, ovisno koji od njih je instaliran.
  U svakom slučaju aktivaciju *NTP*-a, ovdje možemo postići s naredbom:

```
timedatectl set-ntp yes
```

Odnosno, ako ga ipak želimo isključiti:

```
timedatectl set-ntp no
```

**Kako promijeniti trenutni datum i vrijeme?**
Promijenimo ga u slijedeći datum i vrijeme:

```
timedatectl set-time 2018-10-27 20:26:00
```

## Vremenska zona

Kako bismo vidjeli sve vremenske zone koje možemo odabrati na sustavu, možemo pokrenuti naredbu:

```
timedatectl list-timezones
```

Ako želimo promijeniti vremensku zonu u recimo postavku: ***Europa – Zagreb***; to možemo napraviti s naredbom:

```
timedatectl set-timezone Europe/Zagreb
```

**Izvori informacija: `man timedatectl`.**

---

#### 7.3.2.5.3. Konfiguracija lokalnih postavki sustava

Podešavanje lokalnih postavki sustav, osim na klasičan način () moguće je pomoću *systemd* naredbe `localectl`. Lokalne postavke možemo vidjeti sa:

```
localectl

   System Locale: LANG=en_US.UTF-8
       VC Keymap: us
      X11 Layout: us
```

Dakle vidimo kako je lokalna postavka za jezik postavljena na engleski (`LANG=en_US.UTF-8`) u UTF-8 formatu.
Isto tako vidimo da su postavke tipkovnice postavljene na: `US`.



Za više detalja o postavkama vremenske zone pogledajte: **10.4 Postavke vremenske zone i sata te regionalne postavke.**

Ako želimo promijeniti lokalne postave to možemo sa:
```
localectl set-locale LANG=en_US.UTF-8
```
Mijenjanjem postavi s ovom naredbom mijenjaju se sljedeće sistemske datoteke:
- `/etc/locale.conf`.
- `/etc/vconsole.conf`.



Za više detalja o lokalnim postavkama pogledajte poglavlje:
**10.4.4 Regionalne postavke odnosno *Locale*.**



Konfiguracija osnovnih ***systemd*** opcija rada, tijekom pokretanja i gašenja servisa, kao i drugih opcija za rad sa servisima (pr. ograničenja na maksimalni broj procesa i sl.), nalaze se u datoteci: **`/etc/systemd/system.conf`**.

**Izvori informacija: `man localectl`.**

### 7.3.2.5.4. Postavke imena računala

Naredba `hostnamectl` omogućuje nam da trajno promijenimo naziv računala (*hosta*) odnosno sustava bez potrebe za ručnim uređivanjem datoteka. Ova naredba nam daje i neke dodatne informacije o sustavu, kao primjerice rad s imenima računala (*hostnames*).

Ova naredba oslanja se na čitanje (i zapisivanje) u datoteke: `/etc/hosts`, `/etc/hostname`, ali i mnoge druge konfiguracijske datoteke te na servis/*daemon* `systemd-hostnamectl.service` koji ona poziva po potrebi.

Prvo pogledajmo kako dobiti informacije o našem računalu:
```
hostnamectl status

   Static hostname: server-01
         Icon name: computer-server
           Chassis: server
        Machine ID: 7022d20f64cd443698389a4761eabfc2
           Boot ID: eb914e0be59e45078bebd2820154fc71
  Operating System: Red Hat Enterprise Linux Server 7.3 (Maipo)
       CPE OS Name: cpe:/o:redhat:enterprise_linux:7.3:GA:server
            Kernel: Linux 3.10.0-514.21.2.el7.x86_64
      Architecture: x86-64
```
Ovdje vidimo kako se osim postavljenog imena računala (**`Static hostname:`** `server-01`) dobivaju i druge dodatne informacije o sustavu, poput:
- Inačice operativnog sustava (**`Operating System:`** `Red Hat Enterprise Linux Server 7.3 (Maipo)`)
- Kernela (**`Kernel:`** `Linux 3.10.0-514.21.2.el7.x86_64`)
- Arhitekture računala (**`Architecture:`** `x86-64`), ali i drugih podataka o sustavu

Istom naredbom možemo i promijeniti ime našeg računala, sa sljedećom sintaksom:
```
hostnamectl set-hostname racunalo-1
```
Moguće je koristiti i dodatne parametre, koje sada nećemo spominjati.
Mi ćemo ipak koristiti klasičnu metodu konfiguracije sustava, a koja se ne kosi s ovom metodom.



Klasičnu metodu konfiguracije ovog dijela sustava smo objasnili u poglavlju:
**25.6. Osnovna konfiguracija mreže i mrežnog podsustava.**

**Izvori informacija: (433),(434),(435),(436),(437),(K-14),`man timedatectl`,`man localectl`,`man hostnamectl`**

### 7.3.2.6. Ograničenja resursa Systemd servisa

Vezano za ograničenje resursa sustava, tradicionalno na Linuxu postoji nekoliko mehanizama za ograničenja: od takozvanih ***nice*** postavki za postavljanje prioriteta pri izvršavanju programa (procesa), ili postavljanje diskovnog prioriteta (disk I/O), ali i postavke sustava vezane za ograničenja za izvršavanje programa, preko ***ulimit*** sustava, te još neki drugi mehanizmi.



Za više detalja o ograničavanju resursa sustava, pogledajte sljedeća poglavlja:
**4.3. Ovlasti (*permissions* & *modes*).**
**4.5.6. Ograničenja sustava i naredba *ulimit*.**
**4.5.7. Datoteka limits.conf i druga ograničenja sustava.**
**9.1. Prioriteti i procesi.**
**14.1.5.1. Ograničenje diskovnih (I/O) performansi programa/procesa.**

Pogledajte i kako ograničiti ***systemd*** servis po pitanju korištenja sistemskih poziva
**9.4.6. Secure Computing Mode.**

Međutim ista potreba za ograničavanje resursa sustava pokazala se potrebnom i pri pokretanju *systemd* servisa.
Ona je sada standardna mogućnost *systemd* servisa, a sve njene opcije se definiraju u konfiguracijskoj datoteci: `/etc/systemd/system.conf`. Standardno imamo definirano gotovo 500 opcija vezanih za ograničavanje resursa *systemd* servisa odnosno servisa koje on pokreće. Što se tiče postavki za ograničavanje resursa, imamo tri mogućnosti kako ih promijeniti:

- Unosima u konfiguracijskoj datoteci: `/etc/systemd/system.conf`. Potom je potrebno osvježiti *systemd* servis:

```
systemctl daemon-reload
```

- Ručno, postavljanjem željene opcije i njene vrijednosti za konkretna servis. Primjerice za ***httpd*** servis postavimo opciju „`CPUShares`" na vrijednost `1000`:

```
systemctl set-property httpd.service CPUShares=1000
```

- Uređivanjem *unit* (servisne datoteke) željenog servisa, dodavanjem željene opcije i njene vrijednosti unutar sekcije `[service]`. Potom je potrebno osvježiti *systemd* servis i restartati željeni servis (pr. ***httpd***), što postižemo sa:

```
systemctl daemon-reload
systemctl restart httpd
```

Pokazat ćemo vam samo nekolicinu opcija vezanih za ograničavanje resursa *systemd* servisa odnosno servisa koje pomoću *systemd* servisa pokrećemo.

**1.** Postavimo standardno proširenje za sve servise na sustavu na mogućnost da svaki *systemd* servis i pripadajući servisi koje on pokreće imaju pravo otvoriti do 65535 programskih niti. To ćemo uraditi dodavanjem sljedećeg retka u datoteku: `/etc/systemd/system.conf` :

```
DefaultTasksMax=65535
```

Zatim moramo osvježiti *systemd* servis:

```
systemctl daemon-reload
```

**2.** Ograničimo sve servise na sustavu to jest svaki *systemd* servis i pripadajuće servise koje on pokreće, da imaju pravo koristiti maksimalno do 2GB RAM memorije. To ćemo uraditi dodavanjem sljedećeg retka u datoteku: `/etc/systemd/system.conf` :

```
MemorySoftLimit=2G
```

Zatim moramo osvježiti *systemd* servis:

```
systemctl daemon-reload
```

Pogledajmo kako ispisati željenu vrijednost (ako smo ju mijenjali); primjerice vrijednost koju smo upravo postavili:

```
systemctl show --property MemorySoftLimit
```

Iako ovdje imamo gotovo 500 opcija vezanih za ograničavanje resursa *systemd* servisa, u nekim slučajevima one možda neće biti dovoljne za vaš slučaj upotrebe. Međutim postavke ***cgroup*** mehanizama kernela niske razine mogu nam ovdje pomoći. Dakle moguće je definirati i postavke koje možemo dobiti samo preko sustava kontrolnih grupa to jest ***cgroup*** sustava. Primjerice pogledajmo kako preko *cgroup* sustava postaviti ***memory swapiness*** opciju (objašnjenu u poglavlju: **13.8.2.2.),** na ***20***, za naš ***httpd*** servis:

```
systemctl set-property httpd.service ControlGroup=memory.swappiness 20
```

Isto možemo postići dodavanjem opcije **`ControlGroup=memory.swappiness 20`** u `[service]` sekciju servisa ***httpd,*** tj. unutar datoteke: `/usr/lib/systemd/system/httpd.service`.

***Navedene opcije i parametri te način njihove inicijalizacije se može s vremenom mijenjati (ovisno o inačici Systemd servisa).***

**Izvori informacija: (1189),(1190),** **`man systemd.resource-control`**,**`man systemd-system.conf`**, **`man systemd.exec`**.

### 7.3.3. Sistemski servisi (i procesi)

Na većini UNIX operativnih sustava, kao i na svim klasičnim (ili starijim) distribucijama Linuxa, tijekom pokretanja sustava, nakon inicijalizacije kernela pokreće se takozvani ***init*** proces. Procesima se prema Unix/Linux terminologiji nazivaju pokrenuti programi ili servisi. *Init* proces se prvi pokreće prije svih drugih programa odnosno procesa. Važno je znati kako je **PID** broj (*Proces ID*) od ***init*** procesa, broj jedan (**1**), pošto je on prvi servis (proces) koji pokreće sve ostale procese i servise te se brine o njihovom cijelom životnom vijeku. Nadalje on je i zadnji proces tijekom gašenja računala.
Međutim kod novijih sustava više se ne koristi ***init*** već ***systemd*** servis kao njegova zamjena te i on tijekom pokretanja sustava dobiva **PID** broj jedan (**1**) i ima istu namjenu.



Za više detalja o ***init*** i ***systemd*** sevisu pogledajte poglavlja:
**7.3.1. *Init* i sistemski servisi (*daemoni*).**
**7.3.2. *Systemd* i sistemski servisi (*daemoni*)**.

Ipak, **PID** broj jedan (**1**) nije zapravo prvi servis (proces) koji se pokreće tijekom pokretanja računala. Naime, nakon što se računalo pokrene i inicijalizira se kernel, prvi servis koji se pokreće dobiva **PID** broj nula (0). Ovaj servis je poseban kernel servis, koji je skriven od normalnih korisnika (čak i administratora sustava). Pogledajmo osnovno stablo procesa (skraćeno):

```
ps -eaf
UID        PID  PPID  C STIME TTY      TIME        CMD
root         1     0     0 09:31 ?       00:00:02   /usr/lib/systemd/systemd …
root         2     0     0 09:31 ?       00:00:00   [kthreadd]
```

← Vidimo da proces s **PID** brojem 1 (`systemd`) ima roditeljski proces s **PID** brojem 0 (**PPID 0**), a isto je i s procesom koji ima **PID** 2 (`kthreadd`). Ovaj proces s PID brojem nula se naziva procesom mirovanja (engl. *Idle*). Ipak, proces s **PID** brojem 0 nije nepostojeći proces, kako bi mogli pomisliti. Proces s **PID** brojem nula odgovoran je za takozvani *memory paging*, a ovaj se proces naziva ***swapper*** ili ***sched proces*** (na nekim UNIX-ima). Njegov zadatak je eventualno premještanje procesa odnosno pokrenutih programa, između glavne memorije (RAM) i sekundarne memorije odnosno sustava za pohranu (***swap*** particije) kao dijela sustava virtualne memorije operativnog sustava. Pošto je ovaj proces dio kernela, on nije niti vidljiv u ispisu svih procesa na sustavu. Nadalje, on u slučaju upotrebe ***swap*** diska (particije) kao privremenog proširenja RAM memorije aktivira ***kswapd*** servis, koji i odrađuje taj zadatak.



Za više detalja pogledajte:
**9. Proces menadžment.**
**9.6. Procesne grupe.**
**13.8.2. Swap particija.**

Sljedeći poseban proces je ***kthreadd*** koji ima **PID** broj dva (2). On je poseban proces koji radi na vrlo niskoj razini unutar prostora kernela, odnosno unutar direktnog adresnog prostora kernela. Ovaj proces je zadužen za pokretanje novih kernel nîti odnosno kernel pod procesa (engl. *kernel threads*). To znači da je **PPID** broj svih kernel pod procesa je dva (2) jer ih sve pokreće ***kthreadd*** (što nije slučaj s normalnim procesima i servisima). Namjena ovih posebnih kernel procesa je njihovo izvršavanje na vrlo niskoj razini odnosno ispod razine svih drugih programa, a njihovu zadaću čine funkcionalnosti koje se logički nalaze u komponentama kernela, prema principu jedan pod proces - jedna zadaća odnosno skup specijaliziranih zadataka koje treba obaviti. Pogledajmo skraćenu tablicu linux kernel procesa s opisima:

| Proces | Opis | Proces | Opis |
|---|---|---|---|
| **`aio`** | Zadužen za asinkroni pristup diskovima. | **`ksoftirqd`** | Zadužen za softverske signale prekida. |
| **`cpuhp`** | Zadužen za *CPU hot plug* značajku sustava odnosno dodavanje/micanje CPU jezgri tijekom radu računala. | **`kswapd`** | Zadužen za ***Swap*** memoriju (sustav). |
| **`crypto`** | Daje API za kripto funkcije. | **`kworker`** | Zadužen za rad sa signalima prekida, brojačima, I/O operacijama i slično. Ova komponenta pokreće zasebne pod procese, svaki sa svojom specifičnom zadaćom; primjerice: **`kworker:....-flush`**. Zadužen za zapisivanje podataka iz RAM međumemorije za diskovne operacije prema disku (kernel 3.10+). Pogledajte poglavlje: **14.1.1. Sinkroni i asinkroni *I/O*.** |
| **`flush`** | Zadužen za zapisivanje podataka iz RAM međumemorije za diskovne operacije prema disku | **`md`** | Zadužen za sučelja koja imaju više uređaja (pr. RAID polja/kontroleri). |

| Proces | Opis | Proces | Opis |
|---|---|---|---|
| `idle_inject` | Zadužen za praćenje opterećenja jezgri procesora te eventualnog dodavanja vremena mirovanja, kako bi se izbjeglo pregrijavanje procesora. | `migration` | Zadužen za migraciju procesa između procesorskih jezgri (od strane ***task schedulera***). |
| `kauditd` | Zadužen za praćenje sigurnosnih postavki sustava | `netns` | Zadužen za održavanje izoliranih prostora Linuxa (*Linux Namespaces*). |
| `kdevtmpfs` | Zadužen za popunjavanje/izgradnju stabla posebnih datoteka (uređaja [**/dev/**]) | `oom_reaper` | Zadužen za čišćenje sustava u slučaju **OOM** stanja. |
| `khugepaged` | Zadužen za ***Huge pages***. | `writeback` | Zadužen za sve zadatke asinkronog pisanja prema diskovnom podsustavu. Koristi se za zapisivanje onih stranica memorije označenih kao "dirty", prema disku. |
| `ksmd` | Zadužen za ***Same Page Kernel Merging***. | | |

**Izvori informacija: (1512),(1513),(1514),(1515),(1516),(K-16), `man 2 idle`.**

# 7.4. Upravljanje konfiguracijom servisa

***Slijedi napredno poglavlje (7.4.x)!***
Konfiguracija i rad sa servisima ovisi koristimo li ***Init*** ili ***Systemd*** komponentu. Međutim ovdje ćemo govoriti o inicijalnim i globalnim postavkam servisa, neovisno o vršnom centralnom servisu (***Init*** ili ***Systemd***). Dakle upoznat ćemo se s datotekama s kojima se konfiguriraju servisi. Datoteke u direktoriju `/etc/sysconfig/` koriste se za kontrolu i upravljaju konfiguracijom sustava kao i inicijalnom konfiguracijom *servisa* odnosno takozvanih „*daemona*", svaki puta kada se *servis* restarta ili ponovno pokrene. Sadržaj ovog direktorija ovisi i o *softverskima paketima* koje ste instalirali na vaš sustav. Međutim postoji i osnovni skup konfiguracijskih datoteka koje su vidljive na svakom sustavu. Ovdje postoje i određeni pôd direktoriji s pripadajućim datotekama, od kojih su neki standardni, a neki ovise o softverskim paketima koji su instalirani.
Ako se radi o konfiguracijskoj datoteci nekog *servisa*, a nismo sigurni kojega, to možemo provjeriti na sljedeći način:

```
yum provides DATOTEKA
```

S time da je `DATOTEKA` (uz punu putanju) ime datoteke koju tražimo. Ovdje nećemo spomenuti apsolutno sve konfiguracijske datoteke i njihove postavke, već samo neke od njih. Samo neke od standardnih datoteka koje možete pronaći u direktoriju `/etc/sysconfig/` uključuju:

- `/etc/sysconfig/authconfig` – ova datoteka (***RedHat/Centos 7.x***) sadrži opcije i parametre vezane za *autentikaciju*/logiranje na sustav, pa tako imamo slijedeće. ***Međutim za RedHat/Centos 8.x se koriste zasebne datoteke u*** `/etc/authselect/` ***direktoriju koje se isključivo kreiraju s novom naredbom*** `authselect`.
  - `FAILLOCKARGS="deny=4  unlock_time=1200"` – ova opcija označava kako će sustav tolerirati 4 pogrešna pokušaja logiranja, a nakon toga čekati 1.200 sekundi do dopuštanja novog pokušaja logiranja i to direktno na sustav, jer su postavke za spajanje preko mrežnih servisa (za udaljeni pristup) konfigurirane drugdje.
  - `PASSWDALGORITHM=sha512` - ovo je opcija u kojoj se definira algoritam za *hash* funkciju koja će se koristiti za spremanje korisničkih lozinki. U ovom slučaju odabran je algoritam `sha512`.
  - `USEKERBEROS=no` – definira kako se *KERBEROS* protokol neće koristiti za autorizaciju korisnika na sustav.
  - `USELDAP=no` - definira kako se LDAP protokol neće kontaktirati za autorizaciju korisnika na sustav.
  - `USELDAPAUTH=no` - definira kako se LDAP protokol neće koristiti za autorizaciju korisnika na sustav.
  - `USELOCAUTHORIZE=yes` - ova opcija označava kako će se za autorizaciju korisnika na sustav koristiti lokalna autorizacija (na ovom računalu).
  - `USEPWQUALITY=yes` - ova opcija označava kako će se za autorizaciju korisnika na sustav, kod kreiranje lozinke koristiti provjera kvalitete lozinke, prije nego bude prihvaćena.
  - `USESHADOW=yes` - ova opcija označava kako će se za lokalnu autorizaciju korisnika na sustav provjeravati lokalna *shadow* datoteka (`/etc/shadow`) koja će sadržavati *hash* vrijednosti lozinki za korisnike.
  - `USESMARTCARD=no` - ova opcija označava kako se za autorizaciju korisnika na sustav neće koristiti „pametne" kartice (engl. *Smart card*).
  - `USESSSD=yes` - ova opcija označava kako će se za lokalnu autorizaciju koristiti SSSD modul (ovo je standardno). *System Security Services Daemon* (***SSSD***) pruža skup servisa (*demona*) za upravljanje pristupom udaljenim imeničnim servisima i mehanizmima za provjeru autentičnosti. On omogućuje sučelja prema *Name Service Switch* (***NSS***) i *Pluggable Authentication Module* (***PAM***) servisima.


Pogledajte poglavlja:
**7.1.6.6. Linux PAM (Pluggable Authentication Modules)** i
**7.1.6.3. Datoteka /etc/nsswitch.conf**.


*Za provjeru i promjenu ovih opcija i parametara moguće je pozvati naredbu* **`authconfig`**, *a za* ***RedHat/CentOS 8+*** *postoji nova naredba* **`authselect`**.

- `/etc/sysconfig/autofs` – ova datoteka sadrži dodatne opcije i parametre vezane za *automatsko montiranje/demontiranje* odnosno montažu i demontažu datotečnih sustava. Ovaj servis ne mora uvije biti instaliran, pa ga je moguće instalirati sa: `yum install autofs`. Ako imamo ovaj servis, za njega imamo sljedeće opcije:
  - `LOCALOPTIONS=""` – ova opcija unutar navodnika definira posebna pravila kod pokretanja, standardno ovdje nije ništa postavljeno.
  - `DAEMONOPTIONS=""` – u ovoj opciji unutar navodnika definiraju se opcije servisa/*daemona*, poput vremena koje će se čekati prije nego se pokrene ***automount*** procedura, od trenutka pokretanja servisa. Ovo vrijeme se definira kao „`--timeout=60`", što je i standardna vrijednost. Dakle čeka se 60 sekundi prije nego se pokrene „*automount*".

*Standardna postavka osnovne konfiguracijska datoteka autofs servisa je datoteka:* `/etc/autofs.conf`
*Dok se postavke za montiranje (mount) datotečnih sustava nalaze u datoteci:* `/etc/auto.master`
**Izvor informacija:** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/storage_administration_guide/s2-nfs-config-autofs

- `/etc/sysconfig/clock` – ova datoteka sadrži konfiguraciju vremenske zone i postavki vremena, ali je na *RedHat*/*CentOS* Linuxu v.7.x izbačena iz upotrebe jer se na njima koristi alat imena: `timedatectl` koji više ne treba ovu datoteku.
- `/etc/sysconfig/crond` – ova datoteka sadrži konfiguraciju sâmog servisa *cron*, u trenutku njegovog pokretanja. Ovdje imamo varijablu kojoj dodjeljujemo željene argumente.
  - `CRONDARGS=""` – ovdje dodajemo argumente koje želimo proslijediti *cron* servisu u trenutku pokretanja
- `/etc/sysconfig/dhcpd` – ova datoteka sadrži konfiguraciju DHCP servisa u trenutku njegovog pokretanja. Konfiguracija ***DHCP*** servisa se definira u datoteci: `/etc/dhcp/dhcpd.conf`
- `/etc/sysconfig/firewalld` – ova datoteka, zadužena je za dodavanje argumenata pri pokretanju servisa *firewalld*. Ovdje se obično može uključiti argument kojim želimo kod pokretanja ovog servisa uključiti najveću razinu logiranja svih poruka, kojih ***firewwald*** prima i/ili obrađuje; dakle *DEBUG* razina (DEBUG *level*). Opcija je:
  - `FIREWALLD_ARGS=` ovdje možemo postaviti jedan jedini argument: `--debug`

- `/etc/sysconfig/firstboot` – ova datoteka, ***SAMO AKO POSTOJI***, sadrži samo jednu opciju:
  - `RUN_FIRSTBOOT=YES|NO` – ovdje postavljamo vrijednost ***YES*** ili ***NO***. Prvi puta kada se linux sustav pokrene, `init` program poziva skriptu imena: `/etc/rc.d/init.d/firstboot` koja prvo traži postoji li datoteka `/etc/sysconfig/firstboot`. Ako ova datoteka ne postoji, ništa se ne događa. A ako postoji i ne sadrži opciju `RUN_FIRSTBOOT=NO`, pokreće se `firstboot` program, koji pokreće inicijalnu konfiguraciju Linuxa (kao da je tek instaliran) odnosno usmjerava korisnika na početnu konfiguraciju sustava; od mreže, mrežnih servisa i slično.

- `/etc/sysconfig/init` – ova datoteka sadrži konfiguraciju `init` servisa u trenutku inicijalizacije sâmog operacijskog sustava:
  - `BOOTUP=` – ovdje možemo postaviti vrijednost koja označava, hoće li se inicijalni menî, kod pokretanja sustava prikazivati u boji (vrijednost `color`), što je standardno ili ćemo imati detaljniji prikaz (`=verbose`).
  - `RES_COL=` – ova vrijednost (standardno je postavljena na 60) označava broj stupaca (engl. *Column*) u kojima se može prikazati ispis.
  - `MOVE_TO_COL=""` – ova vrijednost, a standardno je postavljena na `"echo -en \\033[${RES_COL}G"` označava sekvencu terminala s kojom se kursor pomiče u drugi stupac.
  - `SETCOLOR_SUCCESS=""` – ova vrijednost, a standardno je postavljena na `"echo -en \\033[0;32m"` označava boju kojom će biti označen tekst koji je „uspješno" odrađen (engl. *Success*). Standardno je to zelena boja.
  - `SETCOLOR_FAILURE=""` – ova vrijednost, a standardno je postavljena na `"echo -en \\033[0;31m"` označava boju kojom će biti označen tekst koji je „neuspješno" odrađen (engl. Failure). Standardno je to crvena boja.
  - `SETCOLOR_WARNING=""` – ova vrijednost, a standardno je postavljena na `"echo -en \\033[0;33m"` označava boju kojom će biti označen tekst koji ima „upozorenje" (engl. *Warning*).
  - `SETCOLOR_NORMAL=""` – ova vrijednost, a standardno je postavljena na `"echo -en \\033[0;39m"` označava boju s kojom će se označiti tekst kod *resetiranja*.
  - `LOGLEVEL=` ova opcija može imati brojčanu vrijednost od `1` do `8` (**1**=najmanje, **8**=najviše), a predstavlja razinu logiranja sistemskih poruka `init` procesa.

- `/etc/sysconfig/irqbalance` - ova datoteka sadrži konfiguraciju `irqbalance` servisa u trenutku njegovog svakog pokretanja. Za više detalja o ovom *servisu.*



Pogledajte poglavlje*:*
***10.5.1.2* Servis irqbalance**.

Pogledajmo i što sve ovdje (/etc/sysconfig/irqbalance) možemo mijenjati:

- IRQBALANCE_BANNED_CPUS= ovdje definiramo sve CPU jezgre za koje **NE** želimo da se koriste od strane irqbalance *servisa/daemona*. Vrijednost je 64 bitna bît maska koja je objašnjena u pogl.: **10.5.1.1 IRQ i Linux**.
- IRQBALANCE_ARGS= ovdje možemo dodavati dodatne argumente kod pokretanja irqbalance *servisa/daemona*

- /etc/sysconfig/iptables-config – ova datoteka sadrži konfiguraciju iptables (vatrozida) za IPv4 protokol koji koristi kernel za postavljanje IPv4 filtriranja mrežnih paketa pri pokretanju sustava, kao i u trenutku kada god se pokrene iptables servis:
  - IPTABLES_MODULES="" - unutar navodnika se može navesti lista kernel modula koji se trebaju učitati zbog proširenja funkcionalnosti *iptables* vatrozida primjerice: "nf_nat_ftp nf_conntrack  nf_tables".
    Ovi kernel moduli se mogu pronaći sa sljedećom naredbom:

```
ls -al  /lib/modules/`uname -r`/kernel/net/netfilter/
```

  - IPTABLES_MODULES_UNLOAD="yes" - ovo je standardno postavljeno na yes, jer se nakon što se vatrozid zaustavi i ponovno pokrene ili restarta, automatski se moraju restartati i kernel moduli navedeni u opciji gore.
  - IPTABLES_SAVE_ON_STOP="no" – u ovom slučaju (standardno), nakon što se vatrozid (iptables) zaustavi, NEĆE se snimiti trenutna aktivna konfiguracija vatrozida (osim ako ju nismo sami snimili).
  - IPTABLES_SAVE_ON_RESTART="no" – u slučaju restarta vatrozida, neće se snimiti aktivna konfiguracija.
  - IPTABLES_SAVE_COUNTER="no" – brojači paketa (statistike) se neće snimati kod zaustavljanja vatrozida.
  - IPTABLES_STATUS_NUMERIC="yes" – standardno će se kod ispisa pravila vatrozida ispisivati IP adrese (numerički) bez da se pretvaraju u imena računala (*no DNS lookup*).
  - IPTABLES_STATUS_VERBOSE="no" – ovakva (standardna) postavka ne daje nam vrlo detaljni statusni ispis .
  - IPTABLES_STATUS_LINENUMBERS="yes" – ovakvom postavkom (standardno), kod ispisa statusa/statistike prikazivati će se i redni brojevi pravila (*rules-a*) vatrozida.
  - IPTABLES_SYSCTL_LOAD_LIST="" – unutar navodnika, navodi se lista sysctl postavki, koje želimo da se ponovno pročitaju i učitaju kod restarta ili stop/start metode pokretanja vatrozida. Lista može biti recimo: ".nf_conntrack .bridge-nf"


Za više detalja o iptables vatrozidu, pogledajte poglavlje: **26.7 Vatrozid (*Firewall*).**

- /etc/sysconfig/ip6tables-config – ova datoteka sadrži konfiguraciju iptables (vatrozida) za IPv6 protokol (isto kao i za IPv4).

- /etc/sysconfig/ipvsadm-config – ova datoteka sadrži konfiguraciju ipvsadm servisa koji je zadužen za mogućnost balansiranja prometa ugrađenu u kernel (engl. *Load Balancing*). Ona postoji samo, ako je ovaj servis instaliran.


Pogledajte poglavlje: **23.4.5.3. Balansiranje opterećenja (*Load Balancing*) – ručni rad.**

- /etc/sysconfig/kernel – ova datoteka definira neke postavke vezane za sâm linuxov kernel:
  - UPDATEDEFAULT=yes - označava hoće li nakon instaliranja novog kernela, taj novi kernel postati „*default*“ odnosno standardni pri pokretanju sustava. Oznaka yes označava, da hoće.
  - DEFAULTKERNEL=kernel - označava standardni naziv softverskog paketa za kernel. Standardni naziv je kernel. To znači kako će se kod nadogradnje sustava, tražiti novi kernel, čije ime (paketa) mora biti kernel. Konkretno bi to bio paket generičke oznake: kernel.x86_64 odnosno recimo imena: kernel-3.10.0-862.9.1.el7.x86_64.rpm. **Izvori informacija: (402),(403),(404).**

  - /etc/sysconfig/network – ova datoteka definira odabir podrazumijevanog usmjerivača (*Default Gateway-a*), postavke imena računala (*hostname*) i drugo, što je zajedničko za sva mrežna sučelja i sve mrežne kartice na sustavu. Naime ona se upotrebljava za postavljanje direktiva koje imaju globalni učinak, odnosno svih onih mrežnih postavki koje nisu specifične za točno određeno mrežno sučelje već za cijelo računalo. Za više informacija o ovoj datoteci i uputama koje ona prihvaća:


Pogledajte poglavlje: **25.6. Osnovna konfiguracija mreže i mrežnog podsustava.**
i to cjelinu: „**Datoteka /etc/sysconfig/network**“

  - /etc/sysconfig/ntpd – ova datoteka sadrži konfiguraciju ***NTP*** poslužitelja u trenutku njegovog pokretanja, ako je instaliran (primjerice sa: yum install ntp).
    - OPTIONS="-g" - ovo je standardno postavljena opcija, kojom se ***NTP*** servisu naređuje da odmah postavi vrijeme prema udaljenom referentnom ***NTP*** poslužitelju. Konfiguracija ***NTP*** poslužitelja se obično nalazi u datoteci: /etc/ntp.conf.


Pogledajte i poglavlje: **10.4.2. NTP.**

- `/etc/sysconfig/rsyslog` – ova datoteka sadrži konfiguraciju ***Rsyslog*** servisa u trenutku njegovog pokretanja.
- `/etc/sysconfig/sshd` – ova datoteka sadrži konfiguraciju ***SSH*** servisa u trenutku njegovog pokretanja. Stoga ovdje imamo samo osnovne opcije i parametre:
  - `SSH_USE_STRONG_RNG=0` – ovo je standardna postavka koja označava kako nemamo poseban hardver za generiranje slučajnih brojeva (*Random Number Generator* - ***RNG***).
  - `AUTOCREATE_SERVER_KEYS="RSA ECDSA ED25519"` - označava da će se kod prvog pokretanja SSH servisa, u trenutku odmah nakon instalacije, kreirati ključevi, i to navedenim redoslijedom: prvo *RSA* pa potom *ECDSA* i *ED25519.*

  Sve ostale opcije i parametri nalaze se u mnogim konfiguracijskim datotekama unutar direktorija: `/etc/ssh/`.
- `/etc/sysconfig/static-routes` – ova datoteka sadrži statičke rute; ako ih baš želimo dodati, a ako nisu vezane za neko mrežno sučelje (*interface*). Ovo je (starija) metoda dodavanja *rute* jer u pozadini koristi stariju naredbu za dodavanje *rute*: `route add`. Važno je znati kako u slučajevima kada se mrežno sučelje isključuje (gasi) pa ponovno uključuje, izbrisat će se sve rute koje su bile vezane za to mrežno sučelje. Ako prvo isključimo pa uključimo određeno mrežno sučelje, `/etc/sysconfig/static-routes` neće se ponovno pokrenuti, tako da se rute navedene u toj datoteci za ovo mrežno sučelje gube.
  Međutim, ako stavite rute u datoteku specifične za sučelje: `route-`**`INTF`** – pri čemu je **`INTF`** ime mrežnog sučelja; pa bi za `eth0` to bila datoteka imena: `/etc/sysconfig/network-scripts/route-eth0`, sustav će ih vraćati kada ponovo podignete (ovo) mrežno sučelje. U ovu datoteku statičke *route* dodajemo prema sljedećem principu (nova *routa* - novi redak), kao i za naredbu `route add`, pa to ovdje nećemo objašnjavati.


Pogledajte i poglavlje:
**25.6.7.2. Trajna konfiguracija statičkih ruta.**

- `/etc/sysconfig/sysstat` – ova datoteka sadrži inicijalnu konfiguraciju ***sysstat*** servisa:
  - `HISTORY=28` – definira koliko dugo će se čuvati log datoteke; standardno je to 28 dana, a nakon toga će se čuvati po direktorijima, po jedan za svaki mjesec.
  - `COMPRESSAFTER=31` – sve **`sa`** i **`sar`** log datoteke starije od 31 dana (ovdje definirano) će se komprimirati.
  - `SADC_OPTIONS="-S DISK"` – ovo su opcije s kojima će se `sadc` servis pokretati.
  - `ZIP="bzip2"` – ovdje se definira s kojim programom će se komprimirati starije log datoteke, Konkretno je postavljeno na program `bzip2`.

# Pôddirektoriji (pôdmape)

Unutar vršnog direktorija `/etc/sysconfig/` nalazi se i nekoliko važnih pôddirektorija. Pogledajmo neke od njih:

- `/etc/sysconfig/cbq/` – ovaj direktorij sadrži konfiguracijske datoteke potrebne za takozvani *Class Based Queuing* (***CBQ***) odnosno menadžment za kontrolu protoka (*bandwidth management*) mrežnih sučelja. Ovaj mehanizam može dijeliti mrežni promet prema bilo kojoj kombinaciji: IP adrese, protokola ili vrste aplikacije.
- `/etc/sysconfig/networking/` – ovaj direktorij (mapa) sadrži dodatne konfiguracijske datoteke, ali se u novijim inačicama ***RedHat/CentOS 7.x+*** više ne koristi.
- `/etc/sysconfig/modules/` – ovaj direktorij može sadržavati skriptne datoteke pomoću kojih će se kernel moduli automatski učitavati pri pokretanju sustava. Novi kernel modul možemo dodati kreiranjem nove datoteke imena: **`ime`**`.modules` gdje je „**`ime`**“ bilo koji opisni naziv po vašem izboru. Vaše datoteke „*ime.module*“ tretiraju skripte za pokretanje sustava kao skriptne datoteke, a kao takve trebaju započeti s direktivom „shabang“.
  (Pr.. `#!/bin/bash`) kao prvom linijom.

  ***Ovo nije mjesto za klasično učitavanja kernel modula***, već je to direktorij: **`/etc/modprobe.d/`.**


Za primjere ispravnog učitavanja kernel modula u direktoriju: `/etc/modprobe.d/` pogledajte poglavlje:
**11.1.1.2 Napredno: Rad s kernel modulima** i to cjelinu: ***Automatsko učitavanje željenih kernel modula.***

- `/etc/sysconfig/network-scripts/` - ovaj direktorij sadrži konfiguracijske datoteke za sva pojedina mrežna sučelja (do **Red Hat 8.x**): mrežne kartice, kao i skripte za uključivanje i isključivanje (podizanje i spuštanje) mrežnih sučelja i za rute.


Pogledajte i poglavlje:
**25.6.5.1.2 Trajna (permanentna) ručna konfiguracija mreže**.

Ako primjerice imamo mrežnu karticu imena `eth0` tada ćemo imati slijedeće datoteke na sustavu:

- `ifcfg-eth0` – ovo je datoteka za konfiguriranje mrežnog sučelja odnosno mrežne kartice `eth0`. Ovdje su moguće razne varijable, koje su navedene u poglavlju:



Pogledajte poglavlje:
**25.6.5.1.2 Trajna (permanentna) ručna konfiguracija mreže.**

- Kontrolne skripte za mrežna sučelja poput `ifup` i `ifdown`, koje se zapravo pozivaju s imenom mrežnog sučelja (pr. `ifup eth0`), u trenutku podizanja ili spuštanja mrežnog sučelja.

- `route-`***`INTF`*** – pri čemu je ***`INTF`*** ime mrežnog sučelja; pa bi za `eth0` to bila datoteka imena: `route-eth0`. Dakle ove skripte mogu postojati za svako pojedino mrežno sučelje za koje se žele dodati specifične STATIČKE *route* (za to konkretno mrežno sučelje). Važno je razumjeti da se ove rute dodaju i nakon što se mrežno sučelje gasi pa ponovno uključuje, jer nakon svakog gašenja mrežnog sučelja inače se brišu i rute koje su vezane za njega.
  Stoga su rute dodane ovdje otporne na restart mrežnog sučelja. Unutar ove datoteke mogu biti sljedeće opcije/rute:
  - *IP ADRESA MREŽE/CIDRmaska* **dev** *UREĐAJ*: poput: `192.168.100.0/24 dev eth0` ili
  - *ADRESA RAČUNALA* **dev** *UREĐAJ*: poput: `192.168.100.145 dev eth0` ili
  - Korištenjem argumenata naredbe `ip route`: *IP ADRESA MREŽE/CIDRmaska* **via** *IP adresa routera* **dev** *UREĐAJ*, poput: `192.168.10.0/24 via 192.168.10.1 dev eth0` ili sa skraćenim zapisom, poput: `192.168.10.0/24 via 192.168.10.1`

- Ovdje se mogu nalaziti i mnoge druge funkcijske skripte zadužene za mrežna sučelja i njihovu konfiguraciju, poput: `network-functions` i drugih.

## Direktorij `/etc/default/`

Unutar direktorija `/etc/default/` nalaze se datoteke specifične za svaki pojedini servis: jedan servis – jedna datoteka, slično kao kod vršnog direktorija `/etc/sysconfig/`. Unutar ovih datoteka se nalaze konfiguracijske varijable i opcije za servise čije opcije nisu konfigurirane unutar direktorija `/etc/sysconfig/`.

Stoga se ovdje radi o samo nekoliko datoteka:

- `/etc/default/grub` – ovo je datoteka s inicijalizacijskim varijablama za ***GRUB*** *boot loader*.



Za detalje pogledajte poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

- `/etc/default/nss` – ovo je datoteka s inicijalizacijskim varijablama za ***NSS*** (*Name Service Switch*).



Za detalje pogledajte poglavlje:
**7.1.6.3. Datoteka /etc/nsswitch.conf.**

- `/etc/default/useradd` je datoteka s inicijalizacijskim varijablama koje se koriste tijekom kreiranja korisničkih računa na sustavu.



Za detalje pogledajte poglavlje:
**7.1.1. Osnovne naredbe za rad s korisničkim računima i grupama.**
**7.1.6.2. Datoteka /etc/login.defs.**
**7.1.6.4. Datoteka /etc/libuser.conf.**

**Izvori informacija:(405),(406),(407),(408)`,man yum` te datoteka: `/usr/share/doc/initscripts/sysconfig.txt`.**

# 7.5. Izvršavanje naredbi u zadano vrijeme

U svakom UNIX odnosno Linux sustavu, moguće je automatizirati pokretanje naredbi ili *shell* skripti u određeno, željeno vrijeme. Naime nekada je potrebno pokretati određene naredbe ili skripte periodički, a nekada postoji potreba pokrenuti ih samo jednom odnosno jednokratno.
Prvi scenarij je pokriven sa dvije naredbe odnosno servisa (*daemona)*, a zadnji s jednim dodatnim. Oni su redom:

- `crontab` - koristi se za periodičko, repetitivno pokretanje programa ili skripti. Ako primjerice želimo pokretati neke skripte ili programe, primjerice svaki dan u ponoć (*antivirus scan*, provjeru diska, *ntp* dohvaćanje točnog vremena i slično). Za njen rad je zadužen `cron` servis odnosno *daemon.*
    - `anacron` - koristi se za periodičko repetitivno pokretanja poput *cron*(tab)-a, ali za slučajeve kada se nešto treba pokretati na dnevnoj, tjednoj ili mjesečnoj bazi bez obzira na točno vrijeme. Za njen rad je također zadužen `cron` servis/*daemon.*
- `at` - ona se koristi za jednokratno pokretanje, kada primjerice želimo samo danas u 20:00.h. jednokratno pokrenuti neku skriptu ili naredbu.



*Crontab* i *anacron* koriste mnoge sistemske skripte i servisi, kako bi osigurale da se važni sistemski servisi pokreću u točno određeno vrijeme dana ili nakon svakog pokretanja računala. Poput provjere dostupnih nadogradnji sustava, postavljanja točnog vremena (preko ***NTP*** servisa), izvršavanja provjere stanja hardvera, izrade sigurnosnih kopija podataka (*backupa*) i slično.

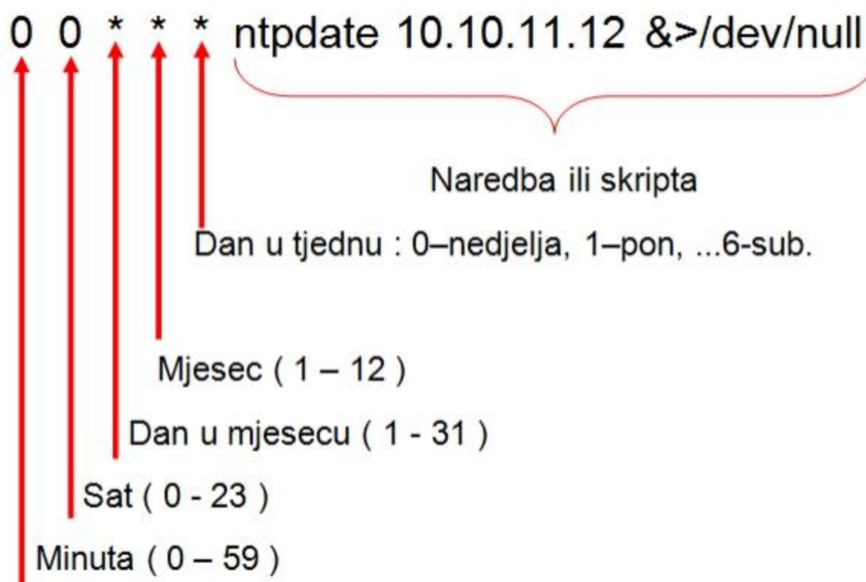
## 7.5.1. *Crontab* servis

*Crontab* servis se koristi za automatsko pokretanje željenih programa u točno određeno vrijeme, pri tome:

- Postoji *crontab* konfiguracija za cijeli operacijski sustav, neovisno o korisnicima.
- Svaki pojedini korisnik može imati svoje *crontab* unose.

Format `crontab` unosa je vidljiv na slici 24. (dolje).

**Slika 24. Servis *crontab* i njegova konfiguracijska datoteka**




Svaki novi redak u *crontab*-u je i novi unos, koji mora biti strukturiran kako je opisano odnosno vidljivo na slici. Dakle za svaki novi unos moramo definirati u koliko minuta i sati te koji dan u mjesecu, kao i u kojem mjesecu se mora pokrenuti željena naredba ili skripta. Na slici je konkretan primjer pokretanja naredbe `ntpdate`, koja se spaja na *ntp* poslužitelj na IP adresi: 10.10.11.12 te preko njega povlači točno vrijeme i postavlja ga na sustav (računalo). Kako je ovdje definirano, to se događa svaki dan, mjesec i dan u tjednu, u ponoć (00:00.h.). Dakle praktično svaki dan u ponoć.

Opcije naredbe `crontab` su:

- `-e` – otvori (uređuj) crontab datoteku (za trenutnog korisnika) (engl. *Edit*).
- `-l` – (malo slovo L) ispiši sadržaj *crontab* datoteke (za trenutnog korisnika).
- `-r` – briše korisnikovu *crontab* datoteku.

**Slijede primjeri**:
Kreirajmo novi *crontab* unos, koji će se pokretati svaki dan u 00:00.h. a koji će pokretati naredbu: `df -h` koja će ispisati zauzeće diskovnih particija. Stoga kreirajmo novi unos, pokretanjem sljedeće naredbe:

```
crontab -e
```

Sada će nam se otvoriti uređivač te preko njega dodajmo naš novi unos (novi redak), koji mora sadržavati samo sljedeće:

```
0 0 * * * df -h
```

Prema opisu iz slike 24. jasno je da prve dvije nule označavaju sat i minutu odnosno 00:00, a tri zvjezdice (*****) znače redom:

- * odnosno prva zvjezdica koja slijedi nakon nula (`0 0`) označava svaki dan u mjesecu.
- * odnosno druga zvjezdica koja potom slijedi označava svaki mjesec.
- * i konačno treća zvjezdica koja zatim slijedi označava svaki dan u tjednu.

I potom navodimo skriptu ili naredbu koje se trebaju izvršiti, u ovom slučaju je to naredba: `df -h`



Važno je razumjeti kako se ***crontab*** unosi rade za svakog pojedinog korisnika. Dakle ovaj unos koji smo napravili, napravljen je za trenutnog korisnika s kojim smo i napravili unos. Korisnici nemaju uvid u unose drugih korisnika, osim što *root* korisnik (*administrator*) može vidjeti sve unose od svih korisnika, ako to želi.

**Slijedi napredna cjelina!**
Da bi **cron** kao usluga uopće radio potrebno je da je ***crond*** servis aktivan, što možemo provjeriti sa sljedećom naredbom:

```
systemctl status crond
● crond.service - Command Scheduler
Loaded: loaded (/usr/lib/systemd/system/crond.service;enabled;vendor preset: enabled)
Active: active (running) since Thu 2022-02-10 08:58:13 EST; 2 weeks 4 days ago
```

U gornjem slučaju je sve u redu (`active`), ali u slučaju kada ***crond*** servis nije pokrenut, možemo ga aktivirati i ručno pokrenuti:

```
systemctl start crond
systemctl enable crond
```

Vezano za konfiguraciju ***crond*** servisa na razini cijelog operativnog sustava, nevezano za korisnika, konfiguracija se nalazi u datoteci: `/etc/crontab`. Ova konfiguracijska datoteka obično izgleda ovako:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# For details see man 4 crontabs
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
```

Osnovno što je vidljivo u ovoj konfiguracijskoj datoteci je kako se osim standardnih polja kao kod običnog *crontab*-a za korisnike, ovdje nalaze i varijable:

- `SHELL` – koja označava s kojim *shellom (ljuskom)* će se svi *crontab* zadaci biti pokrenuti.
- `PATH` – koja definira koje su putanje do datoteka (*PATH*) standardno dostupne za sve *crontab* zadatke.
- `MAILTO` – ovdje se definira korisnik kojem će biti standardno biti poslana elektronička pošta (e-mail) u slučaju neke greške tijekom izvršavanja *crontab* zadatka.
- `user-name` – ako je postavljeno, ovdje se definira korisnički račun s kojim će zadatak (naredba) biti pokrenut.

Osim navedenih metoda, na razini cijelog operativnog sustava, definirano je i nekoliko grupa za vremensko izvršavanje. Točnije kreirani su direktoriji (mape), unutar kojih je moguće stavljati *crontab* zadatke, a odnose se na:

- `/etc/cron.hourly/` – za sve *crontab* zadatke koji se moraju izvršiti svaki sat, a moraju biti kao skripte (datoteke) kreirane u ovom direktoriju.
- `/etc/cron.daily/` – za sve *crontab* zadatke koji se moraju izvršiti svaki dan, a moraju biti kao skripte (datoteke) kreirane u ovom direktoriju.
- `/etc/cron.weekly/` – za sve *crontab* zadatke koji se moraju izvršiti svaki tjedan, a moraju biti kao skripte (datoteke) kreirane u ovom direktoriju.
- `/etc/cron.monthly/` – za sve *crontab* zadatke koji se moraju izvršiti svaki mjesec, a moraju biti kao skripte (datoteke) kreirane u ovom direktoriju.



Datoteke u navedenim vršnim direktorijima obično ne morate mijenjati, iako se one kreiraju za pojedine servise!

**Kako izgleda jedna od ovih datoteka?**
Uzet ćemo za primjer datoteku koja predstavlja *crontab* zadatak, koji se mora izvršavati svaki sat, dakle nalazi se u direktoriju: `/etc/cron.hourly/`, a jedan od takvih je datoteka imena `0anacron`. Ona sadrži nešto poput sljedećeg:

```
#!/bin/bash
# Skip excecution unless the date has changed from the previous run
if test -r /var/spool/anacron/cron.daily; then
    day=`cat /var/spool/anacron/cron.daily`
fi
if [ `date +%Y%m%d` = "$day" ]; then
    exit 0;
fi

# Skip excecution unless AC powered
if test -x /usr/bin/on_ac_power; then
    /usr/bin/on_ac_power &> /dev/null
    if test $? -eq 1; then
    exit 0
    fi
fi
/usr/sbin/anacron -s
```

Dakle ovo je klasična *shell* skripta, koja se pokreće svakih sat vremena, od strane globalnog *cron* servisa.
Konkretna skripta, pokreće `anacron` servis, o kojem ćemo govoriti u sljedećem poglavlju.
Drugi dio globalne konfiguracije *cron* servisa su datoteke koje se nalaze unutar direktorija: `/etc/cron.d/`. Ove datoteke imaju klasičan *crontab* format, jer se u njima mora definirati vrijeme u koje se određena naredba ili skripta moraju pokrenuti, na razini cijelog sustava, nevezano za bilo kojeg korisnika odnosno korisnički račun.

Za primjer pogledajmo datoteku: `/etc/cron.d/sysstat`

```
# Run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib/sa/sa1 1 1
# 0 * * * * root /usr/lib/sa/sa1 600 6 &
# Generate a daily summary of process accounting at 23:53
53 23 * * * root /usr/lib/sa/sa2 -A
```

U ovoj datoteci vidimo dva *crontab* unosa:

- Prvi aktivan je unos: `*/10 * * * * root /usr/lib/sa/sa1 1 1`. Ovaj unos odnosno *crontab* zadatak se pod *root* korisnikom, pokreće svakih deset minuta. Dakle on pokreće skriptu: `/usr/lib/sa/sa1` s parametrima `1 1` (to je `sar` alat za prikupljanje sistemskih statistika).
- Drugi aktivan unos je: `53 23 * * * root /usr/lib/sa/sa2 -A`. I ovaj unos odnosno *crontab* zadatak se pod *root* korisnikom, pokreće svaku noć u 23:53 h. Dakle on pokreće skriptu: `/usr/lib/sa/sa2` s opcijom `-A`. To je također `sar` alat za prikupljanje sistemskih statistika, ali koji prethodno prikupljene statistike sprema u datoteku: `/var/log/sa/sar`**`XX`**, pri čemu je **`XX`** broj dana u mjesecu (pr. `21`).



Za detalje oko naredbe **`sar`**, pogledajte poglavlje:
**10.7.2.3.4 Naredba sar (NUMA ili SMP)**.

**Izvori informacija: (K-12),(K-14), `man cron`, `man crontab`, `man 5 crontab`.**

## 7.5.2. *Anacron* servis

Servis `anacron` je za razliku od `crontab`-*a* zamišljen za pokretanje na stolnim ili prijenosnim računalima (laptopima, tabletima i sl.), a u posebnim slučajevima i na poslužiteljima. Stoga često ne dolazi standardno instaliran sa poslužiteljskim inačicama Linuxa. Naime njegova je namjena pokretanje željenih zadataka: naredbi ili skripti, unutar određenog dana, a ne točnog sata i minute u određenom danu. To je stoga jer stolna ili prijenosna računala nisu aktivna odnosno uključena stalno poput poslužitelja, a često imamo potrebu pokrenuti neki zadatak u danu kada se računalo uključi. *Crontab* nam ovdje ne bi bio od koristi jer, ako bi propustili točno vrijeme kada se pokreće *crontab* zadatak, on ne bi bio pokrenut. S *anacron*om je točno vrijeme pokretanja nevažno; on će pokrenuti željeni zadatak unutar određenog dana, kada god je računalo pokrenuto.

Možemo reći i kako je preciznost pokretanja željene naredbe ili skripte *anacron*a jedan dan, dok je kod *crontab*a to jedna minuta. Globalna konfiguracijska datoteka ***anacrona*** je: `/etc/anacrontab`.

Pogledajmo koje sve osnovne ***anacron*** poslove imamo definirane na sustavu, sa sljedećom naredbom:

```
cat /etc/anacrontab
```

Dakle to je tekstualno konfiguracijska datoteka, čiji sadržaj je otprilike ovakav:

```
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22

#period in days   delay in minutes   job-identifier   command
1                       5            cron.daily       nice run-parts /etc/cron.daily
7                      25            cron.weekly      nice run-parts /etc/cron.weekly
@monthly               45            cron.monthly     nice run-parts /etc/cron.monthly
```

Naime *anacron* provjerava je li potrebno izvršiti neki od *anacron* poslova, unutar radnih sati svakoga dana, što je definirano u varijabli: `START_HOURS_RANGE`. Pri tome, standardno je vrijeme od 03:00.h. ujutro, do 22:00.h. navečer. Dodatno svaki *anacron* posao će biti standardno pokrenuti s namjerno unesenim kašnjenjem, koje je definirano u varijabli: `RANDOM_DELAY`, a koja je standardno postavljena na 45 minuta, unutar kojih će se pokrenuti svaki *anacron* posao. Postoji i minimalno standardno kašnjenje, koje iznosi šest (6) minuta. Naime ovo nasumično vrijeme koje je definirano između 6 i 45 minuta za izvršavanje, osigurava da se svi *anacron* poslovi ne bi pokrenuli u isto vrijeme i tako preopteretili sustav.
Ovo slučajno vrijeme se dodaje na onaj *DELAY* odnosno *KAŠNJENJE* koje upisujemo za svaki pojedini *anacron* posao.

Sintaksa za `anacron` je sljedeća: `DAN        KAŠNJENJE     IDENTIFIKATOR  SKRIPTA`, pri tome:

- `DAN` označava dan u mjesecu kada se skripta ili naredba moraju izvršiti (može biti bilo koji broj : 1-31). Vrijednost 1 znači svaki dan, dok vrijednost 7 znači svakih sedam dana i tako dalje. Moguće je koristi i definirane nazive:
  - `@daily` – označava "dnevno", unutar radnih sati definiranih u varijabli: `START_HOURS_RANGE`
  - `@monthly` – označava "mjesečno" izvršavanje.
- `KAŠNJENJE` je vremenski odmak od trenutka pokretanja operacijskog sustava, u minutama, kada želimo da se skripta ili program pokrenu. Primjerice 15 minuta nakon pokretanja našeg operativnog sustava Linux.
- `IDENTIFIKATOR` je jedinstveni identifikator *anacron* zadatka.
- `SKRIPTA` je naredba ili skripta koju želimo pokrenuti.

U gore navedenom primjeru konfiguracije; datoteke `/etc/anacrontab` pogledajmo jedan unos, koji izgleda ovako:

```
1                 5          cron.daily            nice run-parts /etc/cron.daily
```

To znači kako će ovaj posao identifikatora/imena: `cron.daily` biti:

- Izvršen svaki dan (što definira prvi broj `1`).
- S kašnjenjem od `5` minuta (drugi broj) unutar vremena: 03:00.h. - 22:00.h. i to sa slučajnim dodatnim kašnjenjem između šest (6) i 45 minuta, dakle negdje između:
  - 03:11 (6 minuta standardno + 5 minuta našeg kašnjenja, koje smo definirali za ovaj posao) i
  - 03:50 (45 minuta standardno + 5 minuta našeg kašnjenja, koje smo definirali za ovaj posao)
    pa sve negdje do 22:00.h., ovisno kada se računalo uključilo odnosno pokrenulo.

Globalnu konfiguraciju *anacron*a radimo dodavanjem unosa u datoteku: `/etc/anacrontab`. Pri tome se svaki novi unos mora upisani u novi redak. Ako bi dodali novi redak odnosno novi *ancron* posao, na globalnoj razini, to bi izgledalo ovako:

```
@daily    10    backup.daily   /bin/bash /root/backup.sh
```

U ovom primjeru, vidimo slijedeće:

- `@daily` – ovaj posao se mora izvršavati svaki dan.
- `10` – ovaj posao se mora izvršiti 10 minuta nakon što se uključi računalo.
- `backup.daily` – ovo je identifikator (naziv) posla.
- `/bin/bash  /root/backup.sh` – ovo označava kako se skripta koju želimo pokretati, nalazi na lokaciji: `/root/backup.sh`, a pokreće ju *BASH* ljuska odnosno *bash shell*: `/bin/bash`.

Nakon što smo ovaj novi red/unos dodali u konfiguracijsku datoteku: `/etc/anacrontab` dobro je napraviti provjeru sintakse, kako bismo bili sigurni da nismo nešto pogrešno upisali.
Stoga pokrenimo `anacron` naredbu na sljedeći način:

```
anacron -T
```

Svaki put, kada se pokrene pojedini *anacron* posao, njegova vremenska oznaka se zapisuje u datoteke koje se standardno nalaze u direktoriju: `/var/spool/anacron/`.

Pogledajmo vremensku oznaku *anacron* posla, čiji je identifikator odnosno ime: *cron.daily*. To ćemo vidjeti na sljedeći način:

```
cat  /var/spool/anacron/cron.daily

20180311
```

Vidimo kako je ovaj *anacron* posao zadnji puta izvršen 2018. godine u 3. mjesecu, 11. dan. (`20180311`)

**Izvori informacija: (254),(255),(256),(257),(258),(K-12),(K-14), `man anacron`,`man 5 anacrontab`.**

### 7.5.3. Naredba *at*

Naredba at koristi se za jednokratno pokretanje programa ili skripti u točno određeno vrijeme.
Ako izlaz naredbe at nije preusmjeren u datoteku generirati će se i elektronička pošta (e-mail) koja sadrži:

- Standardnu poruku programa/skripte koja se pokreće odnosno ono što se šalje na standardni izlaz (*Stdout)*.
- Standardnu poruku o grešci samo, ako je ima, odnosno ono što se šalje kao standardna greška *(Stderr)*.

Ovisno o distribuciji Linuxa, ova naredba može ili ne mora biti prisutna na vašem Linux sustavu. Ako ***at*** nije instalirana, možete ju jednostavno instalirati pomoću upravitelja paketa vaše distribucije. Za ***Red Hat*** bazirane distribucije to možemo napraviti sa:

```
yum -y install at
```

Nakon što je program instaliran, provjerite je li **atd** servis pokrenut i postavljen za pokretanje pri podizanju sustava:

```
systemctl status atd

● atd.service - Job spooling tools
   Loaded: loaded (/usr/lib/systemd/system/atd.service;enabled;vendor preset: enabled)
   Active: active (running) since Thu 2022-02-10 08:58:13 EST; 2 weeks 4 days ago
 Main PID: 2867 (atd)
    Tasks: 1 (limit: 1648066)
   Memory: 424.0K
   CGroup: /system.slice/atd.service
           └─2867 /usr/sbin/atd -f
```

- ✓ Ovdje vidimo da je ***atd*** servis aktivan (active (running)) odnosno uredno pokrenut i aktiviran pri svakom pokretanju sustava (enabled).

U slučaju kada nije pokrenut, možemo ga pokrenuti i postaviti da se pokreće sa svakim pokretanjem sustava:

```
systemctl start atd
systemctl enable atd
```

**Slijede primjeri**:
Kreirajmo novi at zadatak koji će kreirati datoteku at.datoteka u 10:18.h. (Tipke: *CTRL* i *d* su za izlaz):

```
at 10:18
touch at.datoteka

CTRL d
```

Neka at pokrene samo skriptu (datoteku) at.skripta.sh (prekidač: -f) u 10:19.h. i još pošalje e-mail (prekidač: -m):

```
at 10:19 -m -f at.skripta.sh
```

Osim navođenja točnog vremena, moguće je navoditi i relativno vrijeme, primjerice kada se određeni at zadatak pokreće za nekoliko, minuta, sati ili dana od sada. Pogledajmo primjere upotrebe naredbe at upotrebom relativnog vremena:

**1.** Za 2 minute od sada:

```
at now + 2 min
```

2. Za 2 sata od sada:

```
at now + 2 hour
```

3. Za 2 dana od sada:

```
at now + 2 day
```

Provjerimo koje sve at zadatke imamo aktivne, pomoću naredbe atq pozivajući ju na sljedeći način:

```
atq

3       Tue Jan  7 10:519:00 2014 a root
```

U prvom stupcu je vidljiv broj posla [***job number***] (on ima oznaku 3 u našem primjeru), a potom slijedi datum kada se treba izvršiti, te pod kojim korisničkim imenom (root u našem slučaju). Ako zbog nekog razloga želimo maknuti odnosno obrisati određeni at zadatak (engl. *job*) za to se koristi naredba atrm nakon koje je potrebno navesti broj ***at*** posla (engl. ***at*** *job number*).

U primjeru ćemo obrisati naš prethodni at posao broj 3, na sljedeći način:

```
atrm 3
```

**Izvori informacija:** **man atd**,**man at**,**man atrm**,**man atq**.

# 7.6. Drugi poznatiji programi i servisi u Linuxu

U ovoj cjelini upoznat ćemo se s važnijim programima i servisima koje nismo opisali u drugim poglavljima ili cjelinama, a koji se često koriste i važni su za rad mnogih drugih servisa ili su važni korisnicima i programerima.



Servisi (i njihovi pripadajući protokoli) s kojima smo se do sada upoznali su:



---

## 7.6.1. Sustavi za verzioniranje kôda

Sustavi za verzioniranje koda (VCS - *Version Control System*) znani su i kao sustavi za kontrolu inačica, također poznati kao kontrola revizije (engl. *revision control* ), kontrola izvornog kôda (engl. *source control*) i upravljanje izvornim kodom (engl. source code management). Ovakvi sustav i alati omogućava praćenje promjena u programskom kôdu tijekom razvoja softvera. Njihova glavna svrha je omogućiti timovima i pojedincima da efektivno rade na istom projektu bez međusobnog prepisivanja rada (kôda ili teksta), dok ujedno pružaju mogućnost praćenja povijesti svih promjena.
Ovakvi sustavi u softverskom inženjeringu zaduženi su za kontrolu, organiziranje i praćenje različitih inačica kôda (teksta ili programa odnosno datoteka).

U načelu svi oni imaju sljedeće značajke:

- Praćenje promjena praćenjem svake promjene napravljenu na kodu. Svaka promjena se sprema kao tzv. ***commit***, koji uključuje informacije poput autora, vremena i opisa izmjene.
- Grananje (*Branching*) - ono omogućava rad na različitim inačicama (verzijama) kôda paralelno. Na primjer, jedan tim može raditi na novim značajkama, dok drugi tim popravlja greške na drugoj grani.
- Spajanje (*Merging*) – kada su promjene na grani završene, mogu se spojiti natrag u glavnu inačicu (obično se naziva *master* ili *main*), čime se osigurava da svi članovi tima imaju ažuriran kôd.
- Kolaboracija – omogućava programerima da rade na istim datotekama bez straha od prepisivanja tuđih izmjena, jer sustav upravlja konfliktima prilikom spajanja promjena.
- Vraćanje unatrag (*Rollback*) – ako dođe do greške, lako je vratiti projekt na prethodnu, stabilnu inačicu koda.

Najpoznatiji primjeri sustava za verzioniranje kôda su ***Git***, ***Mercurial*** i ***Subversion*** (SVN), dok se **Git** najčešće koristi u kombinaciji s platformama poput *GitHub* ili *GitLab*.

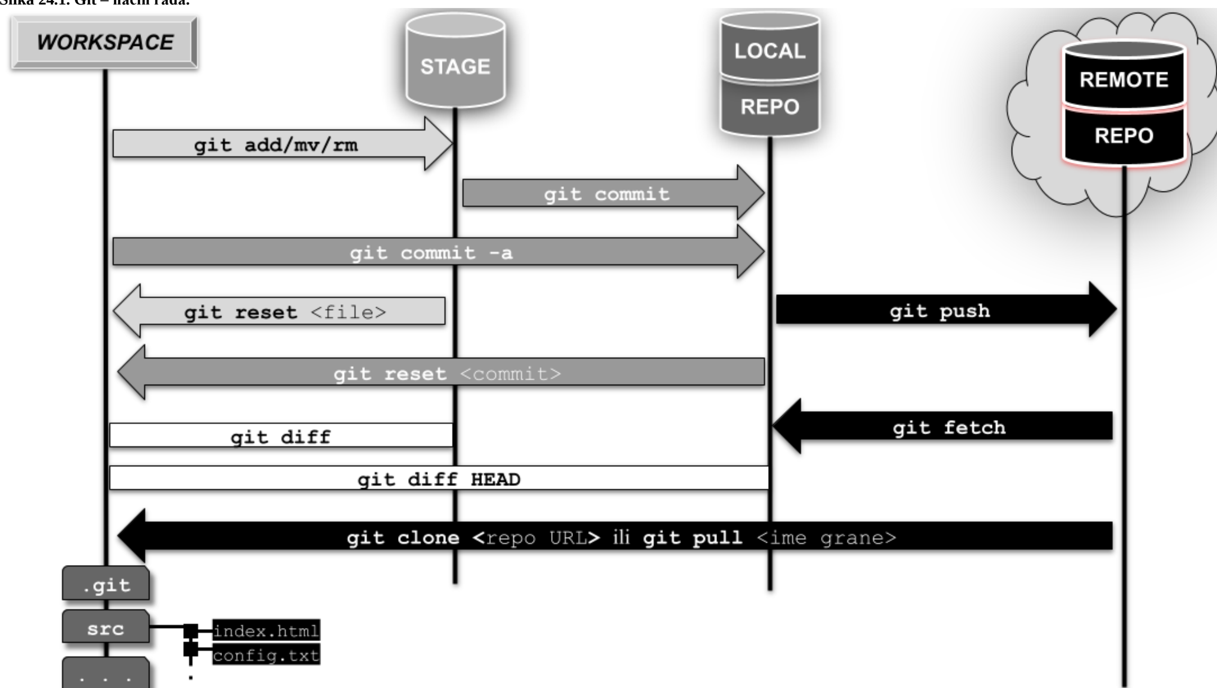
### Git

*Git* je specifičan i jedinstven među sustavima za verzioniranje kôda  zbog nekoliko ključnih karakteristika koje ga čine iznimno popularnim i široko korištenim. Nekoliko glavnih specifičnosti *Git*-a su sljedeće.

- ***Distribuirani sustav verzioniranja*** - za razliku od tradicionalnih sustava verzioniranja poput *Subversiona* (***SVN***), *Git* je distribuirani sustav. To znači da svaki korisnik koji klonira *Git* repozitorij dobiva potpunu kopiju povijesti projekta na svoje računalo, uključujući sve promjene i inačice. Ovaj pristup omogućava:
    - Rad bez potrebe za stalnom vezom s centralnim poslužiteljem.
    - Brži rad jer većina operacija (npr. stvaranje grana, usporedba inačica) ne zahtijeva mrežni pristup.
- **Grananje i spajanje** - *Git* je poznat po svojoj fleksibilnoj i jednostavnoj podršci za grananje (engl. *branching*). Svaka grana može predstavljati novi razvojni pravac, eksperiment ili ispravku greške. Grananje je vrlo brzo i efikasno jer *Git* koristi referenciranje na *commit-ove*, a ne duplicira datoteke. Također, spajanje (engl. *merging*) grana je jednostavno, a *Git* omogućuje i tzv. *rebase* kao alternativu tradicionalnom spajanju.
- ***Snapshot umjesto delta pristupa*** - *Git* ne koristi tradicionalni delta pristup koji pohranjuje samo razlike između inačica datoteka. Umjesto toga, *Git* pohranjuje snimku (engl. *snapshot*) cijelog projekta za svaku promjenu. Ako se datoteka nije promijenila, *Git* jednostavno pokazuje na prethodnu inačicu, što omogućava brži i efikasniji rad.
- ***Efikasno upravljanje granama i spajanjima*** – za razliku od nekih drugih sustava, gdje su grananje i spajanje složeni i ponekad problematični procesi, u *Git*-u je ovaj proces vrlo lagan i brz. To omogućava programerima da slobodno stvaraju grane za nove značajke, eksperimente ili ispravke grešaka, a zatim ih jednostavno spajaju u glavnu inačicu kôda kad završe.
    - ***Jednostavno stvaranje forkova –*** *Git* omogućava lako stvaranje *forkova* projekta. *Forkanje* je proces stvaranja vlastite kopije projekta, gdje korisnik može raditi nezavisno od glavnog repozitorija, a potom, ako želi, predati svoje promjene (*pull request*) natrag u originalni projekt. Ovo je vrlo korisno za projekte otvorenog kôda, gdje više ljudi surađuje izvan centralnog tima.
- ***Brzina i performanse*** – *Git* je dizajniran da bude brz. Većina operacija, kao što su grananje, usporedba inačica, stvaranje *commit-ova* i pregledavanje povijesti, izvode se lokalno na računalu, što ih čini vrlo brzim. Čak i za velike projekte s tisućama datoteka i dugom poviješću, *Git-ova* arhitektura omogućava odlične performanse.
- ***Integritet podataka*** – *Git* koristi kriptografski algoritam SHA-1 za identifikaciju svakog *commit-a*, inačice datoteka i svih ostalih objekata u repozitoriju. To osigurava da je svaki *commit* jedinstveno identificiran, a svaka promjena u kôdu ili povijesti lako prepoznatljiva. Ovo također sprječava manipulaciju i korupciju podataka.
- ***Rad s raznim repozitorijima*** – *Git* podržava rad s više udaljenih repozitorija (*remote*), omogućavajući programerima da surađuju s različitim timovima ili sudjeluju u projektima na različitim platformama. Na primjer, programer može imati pristup na više udaljenih repozitorija i slobodno povlačiti ili gurati promjene između njih.
- ***Blokiranje promjena (staging area)*** – *Git* koristi koncept tzv. *staging area* (područje za pripremu), što omogućava korisnicima da precizno kontroliraju koje će promjene biti dodane u *commit*. To omogućuje razdvajanje radnih promjena od onih koje su spremne za *commit*, pružajući veću fleksibilnost.

Budući da *Git* omogućuje lokalno stvaranje *commit-ova* i grananje (na tzv. *"Local repo"*), centralni poslužitelj je manje opterećen jer korisnici ne moraju neprestano slati i primati podatke. Samo kada su promjene spremne za dijeljenje, korisnik ih gura (*push*) na udaljeni odnosno centralni repozitorij (tzv. *"Remote repo"*), kako je vidljivo na slici dolje.

**Slika 24.1. Git – način rada.**




## Instalacija i rad s GIT-om

Instalacija *Git-a* radi se na sljedeći način:

```
yum -y install git
```

Nakon što je instalacija završena, možete provjeriti inačicu *Git-a* na sljedeći način s naredbom `git`:

```
git --version
```

Nakon instalacije *Git-a*, poželjno je postaviti globalne postavke, poput vašeg imena i e-mail adrese, koje će se pojavljivati u *commit*-ovima. Postavljanje korisničkog imena i e-mail adrese se radi na sljedeći način:

```
git config --global user.name "Vaše Ime"
git config --global user.email "vaš-mail@domena.com"
```

Tek potom možemo provjeriti konfiguraciju odnosno možete provjeriti trenutne postavke pomoću:

```
git config --list
```

**Osnovni primjeri upotrebe**

1. Inicijalizacija novog *Git* repozitorija
Ako imate projekt koji želite staviti pod *Git* verzioniranje, prvo morate inicijalizirati repozitorij:

```
cd /putanja/do/projekta
git init
```

Ova naredba stvara skriveni `.git` direktorij u projektu, gdje *Git* pohranjuje sve svoje podatke.

2. Prije nego što napravite *commit*, morate dodati datoteke koje želite *verzionirati (*ovo je dodavanje u tzv. "*Stage*"*)*:

```
git add ime_datoteke.txt
```

Ili, za dodavanje svih datoteka u trenutnom direktoriju:

```
git add .
```

3. Stvaranje *commit-a*.
Nakon što su datoteke dodane u *staging area*, možete stvoriti *commit*:

```
git commit -m "Opis promjene u ovom commitu"
```

*Commit* je ključni koncept u *Git*-u koji predstavlja spremanje trenutnog stanja kôda u repozitorij. Možemo ga shvatiti kao "fotografiju" ili snimku trenutnog stanja projekta u određenom trenutku. Svaki put kad napravite *commit*, *Git* zabilježi koje su datoteke promijenjene i njihove sadržaje te pohranjuje te informacije zajedno s *metapodacima* poput autora, vremena i poruke (komentara) *commit*-a. C*ommit* je važan zbog praćenja povijesti – *Commi*t omogućuju praćenje svakog koraka razvoja projekta. Svaki *commit* bilježi što je promijenjeno i zašto. *Commit* je važan i zbog izrade sigurnosnih kopija svih promjena – ako se dogodi neka greška, lako je vratiti projekt na stariji *commit* i poništiti problematične promjene. *Commit* je, dakle, osnovni način na koji *Git* prati razvoj projekta, bilježeći svaku važnu promjenu i omogućavajući jednostavno upravljanje s povijesti promjena. Ako primjerice želite *commit*-ati sve promjene u praćenim datotekama (tj. one datoteke koje su već dodane za verzioniranje), možete koristiti opciju `-a`; primjerice ovako:

```
git commit -a -m "Ažurirane sve konfiguracijske datoteke"
```

Ako želite *commit*-ati samo određene datoteke, prvo ih dodajete u *staging area*, a zatim stvorite *commit;* primjerice:

```
git add index.html style.css
git commit -m "Ažuriran sadržaj i stil glavne stranice"
```

4. Pregled statusa repozitorija.
Ako želite vidjeti koje su datoteke promijenjene ili nisu još *commit*-ane, koristite:

```
git status
```

5. Pregled povijesti *commit-ova*.
Za pregled povijesti *commit-ova* u projektu možemo koristiti:

```
git log
```

6. Stvaranje nove grane. Za stvaranje nove grane koristimo primjerice:

```
git branch ime_grane
```

Dok za prebacivanje na tu granu trebamo koristiti:

```
git checkout ime_grane
```

7. Spajanje grana (*merge*). Nakon što ste završili rad na grani i želite te promjene spojiti natrag u glavnu (obično *main* ili *master*), prebacite se na glavnu granu:

```
git checkout main
```

A zatim je potrebno povezati i spojiti promjene:

```
git merge ime_grane
```

8. Povezivanje s udaljenim repozitorijem odnosno s takozvanim *"Remote repo"*.
Ako želite povezati lokalni repozitorij s udaljenim (npr. na *GitHub*-u), koristite:

```
git remote add origin https://github.com/korisnik/ime-repozitorija.git
```

9. Guranje (*push*) promjena na udaljeni repozitorij (*"Remote repo"*).
Nakon što ste *commit-ali* promjene, možete ih poslati na udaljeni repozitorij:

```
git push origin main
```

10. Povlačenje odnosno dohvaćanje (*pull*) promjena s udaljenog repozitorija (*"Remote repo"*).
Ako radite u timu i želite povući najnovije promjene s udaljenog repozitorija:

```
git pull origin main
```

**Izvori informacija: (1746),(1747),(1748),(1749), `man git`.**

### 7.6.2. SQL baze podataka

SQL (*Structured Query Language*) baze podataka su relacijske baze podataka koje koriste SQL jezik za upravljanje podacima. SQL je standardni jezik za definiranje, upravljanje i manipuliranje podacima u bazama podataka.

**Relacijski model**
SQL baze podataka temelje se na relacijskom modelu, što znači da su podaci u bazi podataka organizirani u tablicama (relacijama). Svaka tablica u bazi se pri tome sastoji od redaka (zapisa) i stupaca (polja). Ključni pojmovi uključuju:

- *Tablica* – skupina povezanih podataka u obliku redaka i stupaca. Tablica je osnovna struktura u relacijskom modelu. Sastoji se od redaka (zapisa) i stupaca (atributa). Tablica je zbirka svih podataka o jednom entitetu.
  - *Stupci* – definiraju vrste (tipove) podataka u tablici (npr. ime, dob, email). Atributi (stupci) su svojstva ili karakteristike koje opisuju entitet. Oni definiraju koje vrste podataka će biti pohranjene u tablici. Svaki stupac ima ime i tip podataka (npr. cijeli broj, tekst, datum). Primjer: U tablici "*Zaposlenici*", stupci bi mogli biti **ID**, **ime**, **prezime**, **pozicija**, **plaća**.
  - *Redovi* – pojedinačni zapisi podataka u tablici. Redak predstavlja jedan zapis u tablici. Svaki redak sadrži vrijednosti za svaki atribut tablice. Primjer: Jedan redak u tablici "*Zaposlenici*" može izgledati ovako: **101**, **Ivan**, **Horvat**, **Inženjer**, **5000**.
- *Primarni ključ* – jedinstveni identifikator za redak u tablici. Primarni ključ je jedan ili više atributa (stupaca) koji jedinstveno identificiraju svaki redak u tablici. Vrijednost primarnog ključa mora biti jedinstvena i ne smije biti prazna. Primjer: U tablici "*Zaposlenici*", stupac **ID** može biti primarni ključ jer svaki zaposlenik ima jedinstveni identifikator.
- *Strani ključ* – povezuje tablice pomoću *primarnih ključeva* iz drugih tablica. *Strani ključ* je stupac ili skupina stupaca koji povezuju dvije tablice, pri čemu se koristi *primarni ključ* iz jedne tablice kao referenca u drugoj. *Strani ključevi* omogućuju stvaranje odnosa između tablica. Primjer: U tablici "*Odjeli*", stupac **voditelj_ID** može biti *strani ključ* koji se odnosi na *primarni ključ* **ID** u tablici "*Zaposlenici*", što znači da je određeni zaposlenik voditelj odjela.

**Odnosi između tablica**
U relacijskom modelu tablice mogu biti povezane putem odnosa. Postoje tri osnovne vrste odnosa:

- **1:1 - *Jedan-prema-jedan*** – jedan zapis iz prve tablice odgovara točno jednom zapisu u drugoj tablici.
- **1 - *Jedan-prema-više* –** jednom zapisu iz prve tablice može odgovarati više zapisa u drugoj tablici. Ovaj tip odnosa je vrlo uobičajen.
- **M - *Više-prema-više*** – više zapisa iz prve tablice može odgovarati više zapisa u drugoj tablici. Ovakav odnos se obično implementira putem *poveznih* ili *asocijativnih* tablica koje služe za povezivanje mnogih zapisa iz obje tablice.

**Primjer relacijskog modela**
Zamislimo da imamo dvije tablice u bazi podataka, a to su tablica **Zaposlenici** i tablica **Odjeli**.

Tablica **Zaposlenici**:

| ID | Ime | Prezime | Pozicija | Plaća |
|---|---|---|---|---|
| 101 | Ivan | Horvat | Inženjer | 5000 |
| 102 | Ana | Marić | Menadžer | 6000 |
| 103 | Marko | Kovačević | Programer | 4500 |

Tablica **Odjeli**:

| Odjel_ID | Naziv odjela | Voditelj_ID |
|---|---|---|
| 1 | IT | 101 |
| 2 | Financije | 102 |
| 3 | Razvoj | 103 |

**Voditelj_ID** u tablici "**Odjeli**" je *strani ključ* koji se odnosi na **ID** u tablici "*Zaposlenici*". To stvara odnos **1 (*jedan-prema-više*)** između "**Zaposlenici**" i "**Odjeli**", gdje jedan zaposlenik može biti voditelj samo jednog odjela, ali svaki odjel mora imati jednog voditelja.

Relacijski model omogućuje strukturiranje podataka na logičan način, što olakšava njihovo upravljanje, pretraživanje i ažuriranje. Odnosi između podataka su jasno definirani, čime se omogućuje bolja organizacija i integritet podataka. Normalizacija, odnosno proces minimiziranja redundantnih podataka i osiguravanja konzistentnosti, može se lako primijeniti u relacijskim bazama.

Najpoznatije relacijske SQL baze podataka otvorenog kôda su:

- **MySQL** – ona je jedna od najpopularnijih baza podataka otvorenog koda, često korištena za web aplikacije. Razvila ju je kompanija *MySQL AB*, kupila tvrtka *Sun Microsystems,* a sada je u vlasništvu Oraclea.
  - **MariaDB** – ona je nastala od izvornog kôda *MySQL*-a, koju su kreirali bivši *MySQL* programeri nakon što je Oracle preuzeo *MySQL*. Kompatibilna je s *MySQL*-om, ali nastoji zadržati otvoreni razvoj i pruža dodatne značajke.
- **PostgreSQL** – napredna, objektnu-orijentirana relacijska baza podataka koja podržava širok raspon tipova podataka i složenih upita. Poznata je po svojoj stabilnosti i skalabilnosti.
- **CockroachDB** – distribuirana relacijska baza podataka otvorenog koda koja je dizajnirana za rad na više poslužitelja istovremeno te omogućava visoku dostupnost i toleranciju na greške.

Od SQL baza podataka koje nisu otvorenog kôda najpopularnije su: **Microsoft SQL server** i **Oracle Database**.

SQL upiti su logički kategorizirani u nekoliko skupova naredbi koje se mogu neformalno klasificirati kao podjezici. Navest ćemo najčešće kategorije i primjere naredbi kompatibilnih s ***MySQL***/***MariaDB*** bazama podataka:

- *Data Definition Language* (**DDL**) – koristi se za definiranje strukture baze podataka, neke od naredbi ovdje su:
    - `CREATE` – stvara nove tablice, indekse, baze podataka.
    - `ALTER` – mijenja strukturu postojeće tablice.
    - `DROP` – briše tablice ili baze podataka.
- *Data Control Language* (**DCL**) – koristi se za upravljanje pristupom podacima:
    - `GRANT` – dodjeljuje prava pristupa korisnicima.
    - `REVOKE` – oduzima prava pristupa korisnicima.
- *Data Manipulation Language* (**DML**) – koristi se za manipuliranje podacima:
    - `SELECT` – dohvaća podatke iz tablica.
    - `INSERT` – umeće nove redove u tablice.
    - `UPDATE` – ažurira postojeće zapise.
    - `DELETE` – briše podatke iz tablica.

Naredbe mogu biti pisane velikim ili malim slovima, iako je uobičajeno naredbe pisati velikim slovima, a pri tome svaka cjelina (ili uobičajeno redak) mora završiti sa znakom točka-zarez (`;`).

Pogledajmo primjer kako iz **MySQL/MariaDB** naredbenog retka kreirati bazu imena `Podaci` i unutar nje tablicu imena `Zaposlenici`:

```
CREATE DATABASE Podaci;
```

← Važno je razumjeti da je moguće imati više različitih baza na sustavu, od kojih svaka sadrži svoje tablice i relacije među njima. Svaka baza je logički zaseban entitet bez interakcije s drugom bazom.

Nakon toga moramo ući u tu bazu, kako bi mogli kreirati tablice unutar nje:

```
USE Podaci;
```

A potom nastaviti s kreiranjem prve tablice u toj bazi:

```
CREATE TABLE Zaposlenici (
    ID INT PRIMARY KEY,
    Ime VARCHAR(50),
    Prezime VARCHAR(50),
    Pozicija VARCHAR(50),
    Plaća DECIMAL(10, 2)
);
```

Potom trebamo kreirati i drugu tablicu i relacije među njima (što nećemo objašnjavati) i te tablice popuniti kao u prijašnjem primjeru (s prethodne stranice). Sada možemo vidjeti primjer kako iz tablice `Zaposlenici` pronaći ime i prezime zaposlenika koji su *Inženjer(i)*. Dakle ispisujemo stupce `Ime` i `Prezime` za zaposlenike koji su `Inženjeri`.

```
SELECT ime, prezime FROM Zaposlenici WHERE pozicija = 'Inženjer';
```

I dobit ćemo:

| Ime | Prezime |
|---|---|
| Ivan | Horvat |

## Normalizacije, indeksi i transakcije

***Normalizacija*** je proces strukturiranja podataka u bazama podataka kako bi se smanjila redundancija i poboljšao integritet podataka. ***Indeksi*** ubrzavaju dohvat podataka iz tablica, ali zauzimaju dodatni prostor. Ključne vrste indeksa su:

- *Primarni indeks* – automatski se kreira za primarne ključeve.
- *Sekundarni indeks* – kreira se za stupce koji nisu ključni kako bi se ubrzalo specifične upite.

***Transakcije*** omogućuju grupiranje SQL naredbi kako bi se osigurao integritet podataka.
Važne karakteristike transakcija odnosno tzv. **ACID** svojstva SQL baza podataka su:

- ***Atomicity*** – transakcija se ili u potpunosti izvršava ili uopće ne izvršava. Ako bilo koji dio transakcije ne uspije, sve promjene se vraćaju na početno staje prije problematične transakcije (*rollback*).
- ***Consistency*** – transakcija mora ostaviti bazu podataka u konzistentnom stanju, tj. sva pravila baze podataka (integritet podataka, ograničenja itd.) moraju biti zadovoljena.
- ***Isolation*** – paralelne transakcije ne smiju utjecati jedna na drugu. Njihovi rezultati moraju biti isti kao da su izvršene sekvencijalno (jedna za drugom).
- ***Durability*** – nakon što je transakcija potvrđena (*commit*), njeni rezultati moraju biti trajno zapisani i dostupni, čak i u slučaju kvara sustava.

**ACID** svojstva (*Atomicity*, *Consistency*, *Isolation*, *Durability*) nisu specifična samo za SQL baze podataka. Ona su karakteristika transakcijskih sustava, što znači da se mogu primijeniti i na druge vrste baza podataka, a ne samo na relacijske (SQL) baze. Ipak, relacijske baze su poznate po rigoroznoj primjeni *ACID* svojstava, dok ostale vrste baza podataka mogu imati različit stupanj podrške ovisno o specifičnoj tehnologiji.

*Takozvane **NoSQL** baze poput **MongoDB**, **Cassandra**, **Couchbase**, i **Redis** nisu inicijalno dizajnirane da striktno podržavaju ACID svojstva jer su često optimizirane za skalabilnost i visoke performanse na distribuiranim sustavima. Međutim, mnoge **NoSQL** baze pružaju parcijalnu podršku za ACID svojstva ili nude opcije koje omogućuju ACID transakcije u određenim scenarijima. Umjesto ACID svojstava, **NoSQL** baze često implementiraju **BASE** principe (Basically Available, Soft-state, Eventual consistency), koja su manje striktna u pogledu konzistentnosti podataka, ali bolje prilagođena distribuiranim sustavima.*

**Izvori informacija: (1750),(1751).**

# 8. Arhiviranje i komprimiranje podataka

Za arhiviranje i čitanje arhiviranih datoteka inicijalno se koristila naredba `tar`.
Nastala je zbog potrebe za arhiviranjem podataka na tračne medije, za dugotrajnu pohranu podataka (engl. *backup*).
Njen naziv je došao od ***T**ape **AR**chive* odnosno tračne arhive ili arhive na traku/vrpcu.
Kasnije su nastali i novi programi za istu namjenu. U poglavljima koja slijede upoznat ćemo se s nekima od njih.

## 8.1. Arhiviranje, komprimiranje i dekomprimiranje

Za potrebe arhiviranja podataka ili komprimiranje i dekomprimiranje, najčešće se koristi naredba `tar`. Pri tome razlikujmo arhiviranje od komprimiranja (sažimanja) ili dekomprimiranja, jer arhiviranje uključuje i više od toga. Tako naredba `tar` osim sâmih datoteka i/ili direktorija (mapa) može spremati i sve ovlasti datoteka i direktorija. Ovo je vrlo važno jer u slučaju kada smo arhivirali podatke odnosno napravili njihovu sigurnosnu kopiju (engl. *Backup*), a kasnije ih želimo dearhivirati odnosno vratiti nazad na izvornu lokaciju i u izvornom stanju, želimo da nam budu sačuvane i sve ovlasti datoteka i direktorija: od vlasnika, grupe i ostalih, sve do prava korištenja odnosno ovlasti `rwx` ili posebnih ovlasti poput: `s` ili `t`.

Neki od prekidača naredbe `tar` su:

- `-c` – kreiraj arhivu (engl. *Create*).
- `-v` – detaljniji ispis statusa u radu naredbe (engl. *Verbose*).
- `-t` – samo izlistaj sadržaj arhive.
- `-f` *ime datoteke* – sljedeći argument nakon ovog prekidača je naziv datoteke; pr. `-f arhiva.tar`
- `-z` – dodatno komprimiraj arhivu sa ***gzip*** algoritmom.
- `-j` – dodatno komprimiraj arhivu sa ***bzip2*** algoritmom.
- `-J` – dodatno komprimiraj arhivu sa ***xz*** algoritmom.
- `-I zstd` – dodatno komprimiraj arhivu sa ***zstd*** algoritmom (korištenjem vanjskog programa `zstd`).
- `-p` – sačuvaj sve ovlasti (ovo je standardno, ako ste *root* korisnik) (engl. *Preserve permissions*).
- `--selinux` – sačuvaj i ***SELinux*** atribute (pogledajte poglavlje: **28.2. SELinux sustav**).
- `-x` – dekomprimiraj iz postojeće *TAR* arhive (engl. *Extract*).

**Primjeri**
Kreirajmo `tar` arhivu od trenutnog direktorija (`.`) pri čemu će se arhiva zvati `arhiva.tar`
```
tar -cvf arhiva.tar .
```

Dodajmo datoteku imena `podaci.txt` u postojeću `tar` arhivu (`arhiva.tar`) koju smo upravo napravili:
```
tar -rvf arhiva.tar podaci.txt
```

Kreirajmo `tar` arhivu od trenutnog direktorija (`.`) direktno na traku/vrpcu na `/dev/st0` u tračnom uređaju.
```
tar -cvf /dev/st0 .
```

Kreiraj komprimiranu (*gzip*) *tar* arhivu od trenutnog direktorija (`.`) naziva arhivske datoteke `arhiva.tgz`
```
tar -cvzf arhiva.tgz .
```

Kreirajmo komprimiranu (*gzip*) *tar* arhivu koja će sadržavati samo pâr datoteka: `datoteka1.txt` i `datoteka2.txt`, pri čemu će se arhivska datoteka zvati `arhiva.tgz`
```
tar -cvzf arhiva.tgz datoteka1.txt datoteka2.txt
```

Ispišimo sâmo sadržaj arhivske komprimirane datoteke imena `arhiva.tgz`
```
tar -tvf arhiva.tgz
```

Dekomprimirajmo (*gzip*) i potom de arhivirajmo arhivu (datoteku) imena `arhiva.tgz`
```
tar xvzf arhiva.tgz
```

Kreirajmo ***tar*** arhivu cijelog `/home` direktorija sa svim poddirektorijima i datotekama, te ju komprimirajmo sa *xz* algoritmom:
```
tar cJvf home-dir.tar.xz /home
```

Sada napravimo istu arhivu, ali koja će biti komprimirana upotrebom *gzip* algoritma:
```
tar czvf home-dir.tar.gz /home
```

**Izvor informacija: (K-12), `man tar`.**

### 8.1.1. Komprimiranje i dekomprimiranje

U slučajevima kada nam je potrebno sâmo komprimiranje (sažimanje) ili dekomprimiranje datoteka i/ili direktorija (mape), zbog potreba arhiviranja, kopiranja s medija na medij: poput kopiranja s tvrdog diska na USB disk ili preko mreže, možemo koristiti neke od često korištenih programa za tu namjenu:

- `gzip` i `gunzip` – za komprimiranje (`gzip`) i dekomprimiranje (`gunzip`) sa *GZIP* algoritmom.
- `zip` i `unzip` – za komprimiranje (`zip`) i dekomprimiranje (`unzip`) sa *ZIP* algoritmom.
- `bzip2` i `bunzip2` – za komprimiranje (`bzip2`) i dekomprimiranje (`bunzip2`) sa *BZIP2* algoritmom.
- `xz` i `unxz` – za komprimiranje (`xz`) i dekomprimiranje (`unxz`) sa *XZ* algoritmom.
- `zstd` i `unzstd` – za komprimiranje (`zstd`) i dekomprimiranje (`unzstd`) sa *ZSTD* algoritmom.

Dalje u tekstu upoznat ćemo se sa svakim od navedenih programa.

#### 8.1.1.1. Naredba *gzip*

`Gzip` je program odnosno format datoteka ekstenzija: `.gz` ili `.gzip`, koji je nastao kao zamjena za komercijalni UNIX alat odnosno naredbu `compress`, u sklopu ***GNU*** projekta, početkom 1992.g. On je baziran na *DEFLATE* algoritmu koji je kombinacija *LZ77* i *Huffman coding* algoritama. Nastao je zbog činjenice jer su u to vrijeme *LZW* i drugi algoritmi za komprimiranje bili zaštićeni patentima. Jednu varijantu *LZ77* algoritma za komprimiranje/dekomprimiranje koristi i **ALDC** (*Adaptive Lossless Data Compression*) algoritam koji koriste i tračni uređaji za snimanje na trake (vrpce). **LZ4** format komprimiranja i dekomprimiranja podataka tijekom snimanja odnosno u „letu“ (engl. *on the fly*), koji je baziran na *LZ77*, koristi i **ZFS** datotečni sustav (uz: *gzip*, *DEFLATE*, *LZJB* i druge), što govori o njegovoj brzini rada.
Svaka `gzip` datoteka osim sâmih komprimiranih podataka sadrži i *CRC-32 provjerni zbroj (engl. Checksum)*, tako da se integritet podataka može provjeriti u bilo kojem trenutku. Česti prekidači naredbe `gzip` su:

- `-d` – dekomprimiraj (isto kao i pozivanje naredbe `gunzip`)
- `-c` – izlaz je na standardni izlaz (`stdout`) tako da će izvorne datoteke biti sačuvane.
- `-l` – pogledajmo koliki je omjer komprimiranja postignut na *gzip* datoteci.
- `-r` – rekurzivno komprimiranje svih datoteke unutar svih poddirektorija.
- `-t` – samo testirajmo komprimiranu arhivu, `-1` … `-9` : faktor/brzina komprimiranja: -1 najbrže i “najlošije”, … -9.

**Primjeri**
Prvo izlistajmo datoteke i direktorije, kako bismo vidjeli što sve imamo:

```
ls -alh

total 164K
drwxr-xr-x 3 root root 4.0K Aug  7 11:27 .
drwxr-xr-x 3 root root 4.0K Aug  7 11:18 ..
drwxr-xr-x 2 root root 4.0K Aug  7 11:27 podaci
-rw-r--r-- 1 root root  33K Aug  7 11:22 text1.txt
-rw-r--r-- 1 root root 112K Aug  7 11:22 text2.txt
-rw-r--r-- 1 root root 2.7K Aug  7 11:20 text.txt
```

**1.** Komprimirajmo datoteke: `text.txt`, `text1.txt` i `text2.txt` pomoću programa `gzip` na sljedeći način:

```
gzip text.txt text1.txt text2.txt
```

Pri tome će svaka datoteka biti komprimirana, a izvorne nekomprimirane datoteke neće biti zadržane.

Pogledajmo što smo sada dobili odnosno kakvo je stanje s datotekama u trenutnom direktoriju (mapi):

```
ls -alh

total 24K
drwxr-xr-x 3 root root 4.0K Aug  7 11:30 .
drwxr-xr-x 3 root root 4.0K Aug  7 11:18 ..
drwxr-xr-x 2 root root 4.0K Aug  7 11:27 podaci
-rw-r--r-- 1 root root  157 Aug  7 11:22 text1.txt.gz
-rw-r--r-- 1 root root  391 Aug  7 11:22 text2.txt.gz
-rw-r--r-- 1 root root   66 Aug  7 11:20 text.txt.gz
```

**2.** Dekomprimirajmo prijašnje komprimirane datoteke, pomoću programa `gzip` i to ovako:

```
gzip -d text.txt.gz text1.txt.gz text2.txt.gz
```

Mogli smo koristiti i naredbu `gunzip` na sljedeći način:

```
gunzip text.txt.gz text1.txt.gz text2.txt.gz
```

ili koristiti regularne izraze s obje naredbe, s kojima bi označili sve datoteke čije ime završava sa: `.txt.gz` i to ovako:

```
gunzip *.txt.gz
```

**3.** Komprimirajmo sada datoteku `text2.txt`, ali tako da zadržimo (`-c`) i izvornu nekomprimiranu datoteku.
Pošto će sve ići na standardni izlaz (*standard output*), moramo to sve i preusmjeriti u izvorišnu `.gz` ili `.gzip` datoteku:

```
gzip -c text2.txt> text2.txt.gz
```

Pogledajmo ponovno sadržaj trenutnog direktorija (mape):

```
ls -alh

total 168K
drwxr-xr-x 3 root root 4.0K Aug  7 11:41 .
drwxr-xr-x 3 root root 4.0K Aug  7 11:18 ..
drwxr-xr-x 2 root root 4.0K Aug  7 11:27 podaci
-rw-r--r-- 1 root root  33K Aug  7 11:22 text1.txt
-rw-r--r-- 1 root root 112K Aug  7 11:22 text2.txt
-rw-r--r-- 1 root root  391 Aug  7 11:41 text2.txt.gz
-rw-r--r-- 1 root root 2.7K Aug  7 11:20 text.txt
```

Vidljivo je da je i izvorna datoteka ostala netaknuta te da je kreirana nova koju smo komprimirali (`text2.txt.gz`)

**4.** Pogledajmo kolika (`-l`) kompresija odnosno omjer sažimanja je postignut na našoj datoteci:

```
gzip -l text2.txt.gz

         compressed        uncompressed  ratio uncompressed_name
                391              114048  99.7% text2.txt
```

Postignuta je vrlo velika kompresija od `99.7%` jer se radi o tekstualnoj datoteci koja sadrži samo nizove brojeva koji se ponavljaju, te ih je lako komprimirati (sažeti).

**5.** Komprimirajmo sve datoteke u direktoriju `podaci`. Ali prvo pogledajmo sadržaj direktorija `podaci`:

```
ls -al podaci/

total 160
drwxr-xr-x 2 root root   4096 Aug  7 12:01 .
drwxr-xr-x 3 root root   4096 Aug  7 11:59 ..
-rw-r--r-- 1 root root  33408 Aug  7 11:27 text3.txt
-rw-r--r-- 1 root root 114048 Aug  7 11:27 text4.txt
-rw-r--r-- 1 root root   2688 Aug  7 11:27 text5.txt
```

Sada sve to komprimirajmo rekurzivno (`-r`). Dakle sve datoteke unutar svih poddirektorija (ako postoje) unutar vršnog direktorija `podaci` će također biti uključeni:

```
gzip -r podaci
```

Izlistajmo sada taj direktorij ponovno:

```
ls -al podaci/

total 20
drwxr-xr-x 2 root root 4096 Aug  7 12:03 .
drwxr-xr-x 3 root root 4096 Aug  7 11:59 ..
-rw-r--r-- 1 root root  157 Aug  7 11:27 text3.txt.gz
-rw-r--r-- 1 root root  391 Aug  7 11:27 text4.txt.gz
-rw-r--r-- 1 root root   67 Aug  7 11:27 text5.txt.gz
```

Vidljivo je da su sve datoteke komprimirane i kako izvorne nekomprimirane nisu zadržane!.



Program `gzip` može komprimirati sâmo datoteke, ali ne i direktorije s datotekama. Za tu namjenu koristimo naredbu `tar` s ugrađenom `gzip` metodom komprimiranja ili neke druge programe za komprimiranje ili dekomprimiranje.

**Izvor informacija**: **(K-12), `man gzip`.**

---

### 8.1.1.1.1. Naredba *dd* u kombinaciji sa naredbom *gzip*

***Slijedi napredna cjelina!***

U primjerima koji slijede, vidjet ćemo i kombinaciju rada programa `dd` sa `gzip`*om*. Naime, kako je program `dd` (*disk dump*) svestran, moguće ga je kombinirati s programom za komprimiranje poput `gzip`.

S *disk dumpom* (`dd`) smo već radili, a sada ćemo vidjeti još neke njegove primjere upotrebe.

Prvo se podsjetimo njegovih nekoliko osnovnih prekidača:

- `if` – označava ulaznu datoteku ili uređaj. Ne zaboravimo kako u Linuxu i disk i bilo koji uređaj predstavlja posebna datoteka koju možemo koristiti kao ulazni *uređaj*.
- `of` – označava izlaznu datoteku ili uređaj (s istom napomenom od gore).
- `bs` – označava veličinu blokova podataka koji će se kopirati.
- `count` – označava koliko blokova podataka treba kopirati. Primjerice `count=2 bs=1M` u konačnici znači kopiraj dva (2) puta po jedan (blok) veličine 1 MB što će nam dati ukupno 2MB.

U primjerima u kojima ćemo raditi, imat ćemo sljedeće okruženje:

- `/dev/sda1` – ovo je prva particija na SATA sistemskom disku (na koju je instaliran Linux).
- `/dev/sdb1` – ovo je prva particija na drugom SATA disku.
- `/BACKUP` – ovo je direktorij koji je *montiran* prema `/dev/sdb1` particiji diska.

**Primjeri**

**1.** Napravimo kopiju (*backup)* cijelog sistemskog diska, odnosno njegove prve particije (`/dev/sda1`) u datoteku imena: `Linux-image-sda1.img` u direktoriju `/BACKUP`, pomoću programa `dd`:

```
dd if=/dev/sda1 of=/BACKUP/Linux-image-sda1.img
```

Kada sve završi, dobit ćemo sliku diska (`/dev/sda1`) u datoteci: `/BACKUP/Linux-image-sda1.img`

**2.** Napravimo gotovo isto, ali komprimirajmo podatke prije nego se zapišu u konačnu datoteku, pomoću programa `gzip`. Dakle ovdje izlaz naredbe `dd` koja je kopirala podatke s diska, preko linux *pipea* (znak **|** ) preusmjeravamo kao ulaz prema naredbi `gzip` koja će ih na ulazu komprimirati te komprimirane (na izlazu), snimiti u datoteku:

```
dd if=/dev/sda1 | gzip -c > /BACKUP/Linux-image-sda1.img.gz
```

**2.1** U slučaju kada trebamo napraviti obrnuto: dakle iz komprimirane *image* datoteke, sve *odkomprimirati* i potom snimiti na odredišnu particiju, možemo napraviti sljedeće:

**OPREZ - obrisat ćete sve podatke s trenutne particije */dev/sda1*)**

```
gunzip -c /BACKUP/Linux-image-sda1.img.gz | dd of=/dev/sda1
```

U oba primjera, koristili smo *pipe* funkcionalnost, kako bi izlaz jednog programa, preuzeo i obradio drugi program.

**Izvor informacija**: **`man dd`**,**`man gzip`**,**`man gunzip`**.

---

### 8.1.1.2. Naredba *zcat*

***Slijedi napredna cjelina!***

Naredba `zcat` poziva funkcionalnosti naredbe `gzip`, a funkcionalno se ponaša kao naredba `cat`. Dakle `zcat` u letu dekomprimira sadržaj komprimirane datoteke te se nakon toga ponaša kao naredba `cat`. To možemo promatrati i kao pozivanje naredbe `gzip -c` koja sve šalje na standardni izlaz (*stdout*) te pozivanje naredbe `cat` nakon toga.

**Primjeri**

**1.** Izlistajmo sadržaj komprimirane datoteke `text2.txt.gz` koja u našem slučaju sadrži nîz brojeva.

```
zcat text2.txt.gz

1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
2222222222221111111111111114444444444444443333333333555555555555555
```

**2.** Ponovimo prijašnju naredbu s time kako sada želimo pomoću naredbe `grep` pronaći samo red koji sadrži: `2211`

```
zcat text2.txt.gz | grep 2211

2222222222221111111111111114444444444444443333333333555555555555555
```

Vidimo kako smo u ispisu dobili upravo red koji smo tražili.

**Izvor informacija**: **`man zcat`**.

---

### 8.1.1.3. Naredba *zless*

***Slijedi napredna cjelina!***

Naredba `zless` poziva funkcionalnosti naredbe `gzip`, a funkcionalno se ponaša kao naredba `less`. Dakle `zless` u letu dekomprimira sadržaj komprimirane datoteke te se nakon toga ponaša kao naredba `less`. To možemo promatrati i kao pozivanje naredbe `gzip -c` koja sve šalje na standardni izlaz (*stdout*) te pozivanje naredbe `less` nakon toga.

**Primjeri**

1.Izlistajmo sadržaj komprimirane datoteke `text2.txt.gz`  koja u našem slučaju sadrži nîz brojeva.

```
zless text2.txt.gz

1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
1111111111112222222222222222333333333333333444444444555555555555555
2222222222221111111111111114444444444444443333333333555555555555555
```

**Izvor informacija**: **`man zless`**.

---

#### 8.1.1.4. Naredba *zip*

Za razliku od `gzip`a, `zip` arhive mogu komprimirati/dekomprimirati i direktorije i datoteke. *ZIP* format je analogan kombinaciji naredbi `tar` i `gzip`. *ZIP* format poput *gzip* formata koristi *DEFLATE* algoritam za komprimiranje i dekomprimiranje. Unutar *zip* arhive, moguće je pohraniti cijelu strukturu direktorija i datoteka, ali bez njihovih ovlasti.

Česti prekidači naredbe `zip` su:

- `-r` – rekurzivno komprimiranje direktorija i svih pôddirektorija sa svim datotekama u njima.
- `-T` – testiranje ispravnosti arhive.
- `-x` – ne ubacuj odnosno ne uključuj (engl. *Exclude*) određene datoteke u arhivu.
- `-1` … `-9` – faktor/brzina komprimiranja : `-1` najbrže (i "najlošije"), … `-9` najsporije (i "najbolje").

Česti prekidači naredbe `unzip` su:

- `-t` – za testiranje arhive.
- `-o` – kod dekomprimiranja, pregazi postojeće datoteke bez pitanja.
- `-p` – šalji sadržaj dekomprimirane *zip* arhive u Linux "*Pipe*".

**Primjeri**

**1.** Komprimirajmo datoteke: `text.txt`, `text1.txt` i `text2.txt` te ih sve ubacimo u *ZIP* arhivu imena: `arhiva.zip`:

```
zip arhiva.zip  text.txt text1.txt text2.txt
```

**2.** Komprimirajmo samo datoteku `text1.txt` u *ZIP* arhivu imena: `arhiva1.zip`

```
zip arhiva1.zip text1.txt

  adding: text1.txt (deflated 100%)
```

**3.** Samo testirajmo (`-T`) arhivu imena: `arhiva.zip`

```
zip -T arhiva.zip

test of arhiva.zip OK
```

**4.** Sada samo ispišimo sadržaj *ZIP* arhive imena: `arhiva.zip`

```
unzip -l arhiva.zip

Archive:  arhiva.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     2688  08-07-2014 11:20   text.txt
    33408  08-07-2014 11:22   text1.txt
   114112  08-07-2014 16:38   text2.txt
---------                     -------
   150208                     3 files
```

**5.** Dekomprimirajmo sve datoteke iz *ZIP* arhive: `arhiva.zip`, s time da pregazimo postojeće datoteke (ako ih ima):

```
unzip -o arhiva.zip

Archive:  arhiva.zip
  inflating: text.txt
  inflating: text1.txt
  inflating: text2.txt
```

**6.** Dekomprimirajmo *ZIP* arhivu: `arhiva1.zip` koja sadrži samo jednu tekstualnu datoteku, te ju pošaljimo (preko "***Pipe***") na naredbu `more` (i to pomoću prekidača `-p`), na sljedeći način:

```
unzip -p arhiva1.zip | more

111111111111222222222222222233333333333333444444444555555555555555
111111111111222222222222222233333333333333444444444555555555555555
111111111111222222222222222233333333333333444444444555555555555555
111111111111222222222222222233333333333333444444444555555555555555
111111111111222222222222222233333333333333444444444555555555555555
111111111111222222222222222233333333333333444444444555555555555555
111111111111222222222222222233333333333333444444444555555555555555
```

**7.** Komprimirajmo cijeli direktorij: `podaci` i sve njegove poddirektorije i datoteke rekurzivno, u datoteku `podaci.zip`.

```
zip -r podaci.zip podaci

  adding: podaci/ (stored 0%)
  adding: podaci/text4.txt (deflated 100%)
  adding: podaci/text3.txt (deflated 100%)
  adding: podaci/text5.txt (deflated 99%)
```

**Izvor informacija**: (K-12), **`man zip`**,**`man unzip`**.

---

### 8.1.1.5. Naredba *bzip2*

Program `bzip2` je namijenjen za komprimiranje i dekomprimiranje, poput `gzip`-a. Dakle za rad s datotekama, ali ne i s direktorijima. Izlazna komprimirana datoteka pri tome ima ekstenziju `.bz2`. Za razliku od *gzip* i *zip* programa on koristi algoritam za komprimiranje i dekomprimiranje koji postiže puno bolje rezultate, ali proces komprimiranja traje nešto dulje. Sâm proces komprimiranje se sastoji od nekoliko algoritama, prvi je: "*Burrows–Wheeler transform*" algoritam, nakon kojeg se primjenjuje "*move-to-front transform*" odnsno ***MTF*** Algoritam, a na samom kraju se primjenjuje "*Huffman coding*" algoritam. Ova cjelokupna metoda se zove i "*block-sorting*" metoda komprimiranja. Programe koji mogu komprimirati direktorije i datoteke, potencijalno i uz snimanje ovlasti svih datoteka i direktorija unutar arhive, zovemo "***File Archiver***" programi odnosno programi za arhiviranje. Česti prekidači naredbe `bzip2` su:

- `-d` – dekomprimiraj; isto kao i pozivanje naredbe `bunzip2`.
- `-c` – izlaz je na standardni izlaz (*`stdout`*) tako da će izvorne datoteke biti sačuvane.
- `-k` – sačuvaj izvorne datoteke (inače će biti obrisane nakon što se kreiraju komprimirane datoteke).
- `-r` – rekurzivno komprimiranje svih datoteke unutar svih poddirektorija.
- `-t` – testirajmo komprimiranu arhivu.
- `-1` ... `-9` : faktor/brzina komprimiranja : `-1` najbrže (i "najlošije"), … `-9` najsporije (i "najbolje").

**Primjeri**: Pogledajmo naš trenutni direktorij (mapu):

```
ls -alh

total 164K
drwxr-xr-x 3 root root 4.0K Aug  8 14:47 .
drwxr-xr-x 4 root root 4.0K Aug  7 11:58 ..
drwxr-xr-x 2 root root 4.0K Aug  7 14:10 podaci
-rw-r--r-- 1 root root  33K Aug  7 11:22 text1.txt
-rw-r--r-- 1 root root 112K Aug  7 16:38 text2.txt
-rw-r--r-- 1 root root 2.7K Aug  7 11:20 text.txt
```

**1.** Komprimirajmo naše datoteke: `text.txt`, `text1.txt` i `text2.txt` jednu po jednu, upotrebom programa `bzip2`:

```
bzip2 text.txt text1.txt text2.txt
```

Pogledajmo što se dogodilo:

```
ls -alh

total 24K
drwxr-xr-x 3 root root 4.0K Aug  8 14:49 .
drwxr-xr-x 4 root root 4.0K Aug  7 11:58 ..
drwxr-xr-x 2 root root 4.0K Aug  7 14:10 podaci
-rw-r--r-- 1 root root   78 Aug  7 11:22 text1.txt.bz2
-rw-r--r-- 1 root root   86 Aug  7 16:38 text2.txt.bz2
-rw-r--r-- 1 root root   69 Aug  7 11:20 text.txt.bz2
```

Dakle izvorišne datoteke su prebrisane s novima; kao i kod *gzip*-a. Kako bismo to izbjegli trebali bi koristiti prekidač `-k`

**2.** Dekomprimirajmo ih sada sve zajedno (koliko god ih ima u trenutnom direktoriju):

```
bunzip2 *.bz2
```

**3.** Probajmo sada komprimirati datoteku `text2.txt` u datoteku `text2.txt.bz2` korištenjem izlaza *bzipa2* na standardni izlaz (`-c`) i preusmjeravanja odnosno redirekcije (`>`) u komprimiranu datoteku (`text2.txt.bz2`) i to ovako:

```
bzip2 -c text2.txt> text2.txt.bz2
```

Sve ostale operacije i prekidači su gotovo identični kombinaciji naredbi: `gzip` i `gunzip`.

**Izvor informacija**: (K-12), **`man bzip2`** i **`man bunzip2`**.

---

### 8.1.1.6. Naredba *bzcat*

***Slijedi napredna cjelina!***
Naredba bzcat poziva funkcionalnosti naredbe bunzip2, a funkcionalno se ponaša kao naredba cat. Dakle bzcat u letu dekomprimira sadržaj komprimirane datoteke te se nakon toga ponaša kao cat. To možemo promatrati i kao pozivanje naredbe bzip2 -c koji sve šalje na standardni izlaz (***stdout***) te pozivanje naredbe cat nakon toga.
Ovo ponašanje je identično naredbi zcat.

**Primjeri**
**1.** Ispišimo sadržaj datoteke komprimirane sa bzip2 imena text2.txt.bz2 upotrebom naredbe bzcat:

```
bzcat text2.txt.bz2

111111111111222222222222222233333333333333444444445555555555555555
111111111111222222222222222233333333333333444444445555555555555555
111111111111222222222222222233333333333333444444445555555555555555
111111111111222222222222222233333333333333444444445555555555555555
111111111111222222222222222233333333333333444444445555555555555555
222222222222111111111111111444444444444443333333333555555555555555
. . .
```

**2.**Ponovimo prijašnju naredbu s time kako sada želimo pomoću grep naredbe pronaći samo redak koji sadrži: 2211

```
bzcat text2.txt.bz2 | grep 2211

222222222222111111111111111444444444444443333333333555555555555555
```

**Izvor informacija**: **man bzcat**.

---

### 8.1.1.7. Naredba *xz*

Naredba xz koristi takozvani *Lempel–Ziv–Markov Chain algoritam* (***LZMA***) koji se koristi za komprimiranje podataka bez gubitaka. Ovaj algoritam je bio u razvoju od 1996. godine od strane ***Igora Pavlova***, a prvi je put korišten u **7z** formatu **7-Zip** arhivera. Ovaj algoritam koristi shemu kompresije uporabom takozvanog rječnika, donekle sličnu algoritmu **LZ77** koji su objavili ***Abraham Lempel*** i ***Jacob Ziv*** 1977. godine i ima visok omjer kompresije te promjenjivu veličinu rječnika kompresije (do 4 GB), dok i dalje zadržava brzinu dekompresije sličnu drugim često korištenim algoritmima kompresije. U većini slučajeva, ***xz*** postiže veći stupanj kompresije od alternativa kao što su ***gzip*** i ***bzip2***. Brzina dekompresije je veća od ***bzip2***, ali sporija od ***gzipa***. Međutim kompresija može biti puno sporija od ***gzip***-a i sporija je od ***bzip2*** za visoke razine kompresije, a najkorisnija je kada će se komprimirana datoteka koristiti više puta. Primjerice noviji kernel moduli se komprimiraju sa ***xz***, a u procesu učitavanja prvo se dekomprimiraju u memoriji. Uobičajena exstenzija datoteka komprimiranih sa ***xz*** je obično .xz. Baš kao ***gzip*** i ***bzip***, ***xz*** može komprimirati samo pojedinačne datoteke (ili tokove podataka) kao ulaz. Dakle on ne može povezati više datoteka u jednu arhivu. Za tu namjenu se prvo koristi program za arhiviranje, kao što je primjerice tar.
Naredba xz uglavnom već dolazi s većinom distribucija Linuxa, ali ako nije instalirana, možete ju instalirati s naredbom:

```
yum -y install xz
```

**Slijede primjeri upotrebe**

**1. Komprimiranje pojedine datoteke**
Ova procedura će komprimirati file.txt i stvoriti file.txt.xz, ali i ukloniti izvornu datoteku file.txt.

```
xz file.txt
```

U slučaju kada želimo komprimirati navedenu datoteku, ali i zadržati izvornu, to moramo učiniti na sljedeći način:

```
xz -k file.txt
```

Odnosno istu stvar možemo napraviti i upotrebom redirekcije i prekidača (-c) za komprimiranje:

```
xz -c file.txt > file.txt.xz
```

**2. Dekomprimiranje već komprimirane xz datoteke**
U ovom primjeru, dekomprimirat ćemo datoteku file.txt.xz, ali i ovdje će izvorna datoteka biti uklonjena.

```
xz -d file.txt.xz
```

Ovdje možemo koristiti i naredbu unxz koja zapravo u pozadini poziva izvornu naredbu xz s prekidaćem -d.

```
unxz file.txt.xz
```

I ovdje kada želimo dekomprimirati datoteku, ali i zadržati izvornu, to moramo učiniti na sljedeći način (prekidač -k):

```
xz -dk file.txt.xz
```

**3. Informacije o razini komprimiranja datoteke**
Ako imamo takvu potrebu, moguće je dobiti informacije o razini komprimiranja za određenu datoteku; pogledajmo

```
xz -l file.txt.xz
Strms  Blocks   Compressed Uncompressed  Ratio  Check   Filename
    1       1     287.9 KiB  1,465.4 KiB  0.196  CRC64   file.txt.xz
```

**4. Upotreba više programskih nîti prilikom komprimiranja**
Naredba `xz` podržava višenîtni rad to jest upotrebu više procesorskih jezgri za komprimiranje. Mi ćemo koristiti sve dostupne jezgre procesora, s prekidačem `--threads=0` (`0` označava sve jezgre, a moguće je navesti i željeni broj [*1, 2, 3, n*]).

```
xz --threads=0 file.txt
```

S ovime se drastično može ubrzati proces komprimiranja.

**5. Kombinacijom s programom `tar`, za arhiviranje direktorija**
Kao što smo vidjeli `xz` ne može napraviti arhivu koja se sastoji od direktorija i datoteka, već može samo komprimirati (ili dekomprimirati) samo jednu datoteku.
Ako imamo potrebu arhivirati direktorije, moramo koristiti naredbu `tar`, koja podržava i ***xz*** mehanizam za komprimiranje.
Kreirajmo ***tar*** arhivu cijelog `/home` direktorija sa svim poddirektorijima i datotekama:

```
tar cJvf home-dir.tar.xz /home
```

Arhivska datoteka će biti kreirana u direktoriju (mapi) u kojoj se trenutno nalazimo odnosno iz koje smo pokrenuli naredbu!.
Dakle dobili smo ***tar*** arhivu, stabla `/home` direktorija, imena `home-dir.tar.xz`, koja je komprimirana pomoću **xz**.

**6. Testiranje (provjera) arhive ili komprimirane datoteke**
Svaka datoteka komprimirana pomoću ***xz*** sadrži i provjerni zbroj (*CRC*), kako bi se mogao provjeriti integritet podataka:

```
xz -tv home-dir.tar.xz

home-dir.tar.xz (1/1)
  100 %          1,288 B / 30.0 KiB = 0.042
```

**Izvori informacija: (1121),(1122), `man xz`,`man unxz`.**

## 8.1.1.8. Naredba zstd

***Zstd*** je brzi algoritam za kompresiju bez gubitaka podataka i alat za kompresiju podataka, sa sintaksom naredbenog retka sličnom naredbama ***gzip*** i ***xz***. Temelji se na obitelji ***LZ77***, s daljnjim fazama entropije ***FSE*** i ***huff0***. *Zstd* nudi vrlo konfigurabilnu brzinu kompresije, s brzim načinima rada, s više od 200 MB/s po jezgri procesora, a snažniji načini rada približavaju se ***lzma*** omjerima kompresije. On također ima vrlo brz dekoder, s brzinama većim od 500 MB/s po jezgri procesora. Naziv je dobio od *Zstandard*, a jedan od ciljeva njegovog razvoja su bili scenariji kompresije u stvarnom vremenu i s boljim omjerima kompresije, što je postignuto na razini ***zlib*** biblioteka. *Zstandardov* format je dokumentiran u **RFC8878**.
Ako već nije instaliran na sustavu, instalirajmo ga na sljedeći način:

```
yum -y install zstd
```

Sada smo dobili nekoliko naredbi: `zstd` (za komprimiranje i dekomprimiranje, `unzstd` (za dekomprimiranje) i neke druge.
Ekstenzije za ***zstd*** su obično `.zst` ili `.tar.zst`. *Zstandard* (***zstd***) je najbrži od većine prethodno navedenih algoritama.
Višenitni rad (engl. *multi-threading*) odnosno sposobnost upotrebe više jezgri procesora istovremeno je još jedna prednost *zstd-a* nad programima ***gzip*** i ***bzip2***.
Pogledajmo nekoliko primjera njegove upotrebe. Komprimirajmo datoteku imena: `dokument.doc`, s njim:

```
zstd dokument.doc
```

Izlazna arhivska datoteka dobiva ekstenziju `.zst`. Konkretno dobit ćemo datoteku imena: `dokument.doc.zst`.
Potom kasnije možemo dekomprimirati (`-d`) prethodno kreiranu datoteku, koja je automatski dobila ekstenziju `.zst`:

```
zstd -d dokument.doc.zst
```

Međutim ***zstd*** ne može kreirati arhivu direktorija (mapa) i direktorija, ali srećom tu nam pomaže naredba `tar`. U primjeru koji slijedi, kreirat ćemo arhivu naziva `arhiva.tar.zst`. Tar arhiva će sadržavati sve datoteke unutar našeg direktorija `/home/hrvoje/`, a koja će potom biti komprimirana sa ***zstd*** kompresijom. Pogledajmo kako to učiniti:

```
tar -I zstd -cf arhiva.tar.zst /home/hrvoje/
```

I sada ćemo napraviti obrnuti proces, to jest napraviti ***zstd*** dekompresiju ***tar*** arhive, u trenutni radni direktorij:

```
tar -I zstd -xvf arhiva.tar.zst
```

Moguće je postojeću arhivu i testirati (`-t`), pa pogledajmo test integriteta prethodno kreirane arhive:

```
zstd -t arhiva.tar.zst
```

***Postoje i desetci drugih opcija i parametara koje možemo koristiti!***

**Izvori informacija: `man zstd`, `man unzstd`, RFC8878.**

### 8.1.2. Naredba *rsync*

***Slijedi napredna cjelina!***
Program odnosno naredba `rsync` se koristi za obavljanje operacija sigurnog kopiranja podataka odnosno datoteka i direktorija (mapa). Možemo reći kako se `rsync` alat koristi za sinkronizaciju datoteka i direktorija s jednog mjesta na drugo, na učinkovit način. *Backup* odnosno odredišna lokacija pri tome može biti na lokalnom disku (poslužitelju) ili na udaljenom poslužitelju.

Neki od prekidača/opcija naredbe `rsync` su:

- `-a` – arhivski način rada (isto kao da smo koristili: `-rlptgoD`); ne uključuje `-A`, `-H` i `-X`.
- `-H` – sačuvaj ***hard linkove*** - ovo može biti vremenski zahtjevno jer ih se prvo u (pozadini) treba pronaći.
- `-S` – prepoznaju se ***sparse*** datoteke i s njima se postupa na ispravan način.
- `-x` – ne prelazi na drugi datotečni sustav („*This tells rsync to avoid crossing a filesystem boundary when recursing*")
- `-e` – specificiraj udaljenu ljusku (*shell)* za upotrebu (pr. *ssh*)
  - `--delete` – pri tome briše udaljene datoteke s odredišnog direktorija (one koje nisu na strani koja ih šalje, samo za one direktorije koji se sinkroniziraju).
  - `--rsync-path` – potencijalno definiraj *rsync* program koji se treba pokrenuti na udaljenoj strani/računalu.
- `-h` - ispisuj sve u ljudima razumljivom formatu (kB, MB, GB)
  - `--progress` – prikaži napredak tijekom izvršavanja.
- `-l` – kopiraj simboličke linkove kao simboličke linkove.
- `-p` – sačuvaj ovlasti (engl. *Permissions*).
- `-A` – sačuvaj ***ACL*** liste (podrazumijeva `-p`).
- `-X` – sačuvaj proširene atribute (Engl. *Extended attributes*).
- `-z` – komprimiraj podatke tijekom transfera na odredišno računalo.
  - `--compress-level=NUM` – eksplicitno definiraj razinu kompresije (`NUM`).
  - `--skip-compress=LIST` – **preskoči** datoteke s određenim sufiksom/ekstenzijom: *`LISTa`*
    - Primjer: `--skip-compress=gz/jpg/mp[34]/7z/bz2`
- `-r` – kopiraj direktorije rekurzivno (sve pôddirektorije).
- `-u` – ne kopiraj one datoteke koje već postoje na odredištu i vrijeme modifikacije im je novije nego s izvora. Ako je vrijeme promijene isto ili, ako je veličina drugačija, prekopiraj ih.
- `-d` – kopiraj i direktorije (mape).
- `-v` – ispis s više detalja (engl. *Verbose*).

Na većini Linuxa ova naredba je već instalirana. U slučaju kada nije, instalirajmo ju sa slijedećom naredbom:

```
yum install rsync
```

**Koje su metode upotrebe odnosno primjene, naredbe** `rsync`:

1. Kopiranje odnosno sinkronizacija datoteka i direktorija iz jednog direktorija na drugi (na istom računalu).
2. Kopiranje odnosno sinkronizacija datoteka i direktorija s jednog računala na drugo računalo, korištenjem `rsync` metode za kopiranje preko mreže.
3. Kopiranje odnosno sinkronizacija datoteka i direktorija s jednog računala na drugo računalo, korištenjem metode za kopiranje preko `SSH` protokola.

## Slijede primjeri upotrebe naredbe rsync

## 1. Lokalno kopiranje/sinkronizacija

**1.1**. Za potrebe sinkronizacije datoteka i direktorija unutar jednog računala odnosno kopiranje datoteka unutar jednog direktorija, rekurzivno, u neki drugi direktorij, možemo koristiti sljedeći primjer.

Recimo da unutar direktorija `/root/test/` imamo neke podatke (datoteke) koje želimo na siguran i ispravan način prekopirati/sinkronizirati u direktorij `/root/arhiva-1/`. To ćemo napraviti pomoću sljedeće `rsync` naredbe:

```
rsync -avh  /root/test/ /root/arhiva-1/


sending incremental file list

novi/
novi/test.txt

sent 173 bytes  received 38 bytes  422.00 bytes/sec
total size is 22.25K  speedup is 105.44
```

Sada su sve datoteke i direktoriji unutar `/root/test/` prekopirani u `/root/arhiva-1/` direktorij te su sačuvane sve njihove ovlasti jer smo koristili prekidač `-a`.

Navedeni prekidač objedinjuje sljedeće opcije:
- Ovlasti (`-p`).
- Rekurzivno kopiranje sadržaja direktorija (`-r`) i sve direktorije (`-d`).
- *Soft* linkove (`-l`).
- Vlasnika i pripadajuću grupu (`-o` i `-g`) kojoj pripadaju datoteke.
- Vremenske oznake (`-t`).
- Kopira sve "*device*-e" odnosno posebne vrste datoteka (`-D`) koje predstavljaju razne uređaje.

**1.2**. Za potrebe sinkronizacije datoteka i direktorija unutar jednog računala odnosno kopiranje datoteka unutar jednog direktorija, rekurzivno, u neki drugi direktorij, s time da želimo zadržati i:
- Ovlasti (engl. *Permissions*) - sve što je u upotrebi s prekidačem `-a` te dodatno:
  - Proširene ovlast (engl. *Extended atributes* + *SELinux*) `-X`
  - ***ACL*** liste: `-A`
  - *Hard* linkove `-H`

Recimo i kako unutar direktorija `/root/test/` imamo neke podatke (datoteke) koje želimo na siguran i ispravan način prekopirati odnosno sinkronizirati u direktorij `/root/arhiva-1/` na sljedeći način:

```
rsync -avhXAH  /root/test/ /root/arhiva-1/


sending incremental file list

sent 138 bytes  received 16 bytes  308.00 bytes/sec
total size is 22.25K  speedup is 144.47
```

**1.3.** Kopiranje/sinkronizacija sâmo jedne datoteke. U slučaju kada moramo sinkronizirati samo jednu datoteku u odredišni direktorij i pri tome sačuvati izvorišnu datoteku (na izvoru s kojeg kopiramo). Dakle izvorišna datoteka je: `/root/test/generator.sh`, a odredišni direktorij (mapa) je: `/root/arhiva-1/`. To ćemo postići sa:

```
rsync -avh /root/test/generator.sh  /root/arhiva-1/


sending incremental file list

sent 37 bytes  received 12 bytes  98.00 bytes/sec
total size is 68  speedup is 1.39
```

**1.4** Kopiranje/sinkronizacija s***parse*** datoteka i direktorija unutar jednog računala odnosno kopiranje datoteka koje se nalaze unutar jednog direktorija, rekurzivno u neki drugi direktorij. S dodatkom kako imamo i posebne ***sparse*** datoteke koje se moraju kopirati na poseban način. Recimo kako unutar direktorija `/root/test/` imamo neke datoteke među kojima su i ***sparse*** datoteke koje želimo na siguran i ispravan način prekopirati/sinkronizirati u direktorij `/root/arhiva-1/`.

U ovom slučaju imamo samo jednu ***sparse*** datoteku imena: `sparse-file`.
Zbog potrebe prepoznavanja i rada sa ***sparse*** datotekama, moramo koristiti prekidač `-S` (veliko slovo S) ili `--sparse`.

```
rsync -avhS  /root/test/ /root/arhiva-1/

sparse-file

sent 10.74G bytes  received 35 bytes  161.48M bytes/sec
total size is 10.74G  speedup is 1.00
```

Ovaj proces će trajati kao da se kopira datoteka koja nije ***sparse,*** dakle pune veličine, ali će na odredištu završiti kao ***sparse*** odnosno smanjena. Naime naredba `rsync` mora proći kroz cijelu ***sparse*** datoteku, pošto ona ne zna koji njeni dijelovi su popunjeni "nulama" odnosno koji dijelovi su prazni, a koji sadrže podatke, što iziskuje dosta vremena.

Zbog ovakvog (ne optimiziranog) rada naredbe `rsync` sa ***sparse*** datotekama, preporuča se `rsync` kombinirati s naredbom `tar` koja bi trebala odraditi ***sparse*** dio. Pogledajte primjere upotrebe `tar` naredbe u slučaju upotrebe sa ***sparse*** datotekama.



Za više informacija o ***sparse*** datotekama, pogledajte poglavlje:
**4.6 Prorijeđene (*Sparse*) datoteke.**

U svim primjerima do sada obratite pažnju na poruku, koja se pojavljuje nakon pokretanja naredbe `rsync`:

```
rsync:sending incremental file list.
```

To znači kako se sve što se kopira/sinkronizira, radi inkrementalno, što znači da se ne kopiraju/sinkroniziraju datoteke koje se nisu mijenjale. Inkrementalni rad prema tome znatno ubrzava proces kopiranja odnosno sinkronizacije.

## 2. Kopiranje/sinkronizacija lokalnih podataka na udaljeno računalo (preko mreže)

**2.1.** Za potrebe sinkronizacije datoteka i direktorija s jednog računala odnosno kopiranje datoteka s jednog direktorija, rekurzivno, na drugo računalo i to u neki specifičan direktorij na njemu, s time da želimo zadržati i:

- Ovlasti (engl. *Permissions*) - sve što je u upotrebi s prekidačem `-a` te dodatno:
  - Proširene ovlasti (engl. *Extended atributes + SELinux*) `-X`
  - *ACL* liste: `-A`
  - *Hard* linkove: `-H`

Zbog kopiranja podataka preko mreže, ponekad je dobro i uključiti i kompresiju podataka pri prijenosu, sa prekidačem `-z`.
S time kako je naša IP adresa: 192.168.100.1, a adresa udaljenog računala, na koji radimo kopiranje/sinkronizaciju: 192.168.100.254. Odredišni direktorij na udaljenom poslužitelju s IP adrese: 192.168.100.254 je `/BKP/`.

```
rsync -avhXAHz /root/test/ root@192.168.100.254:/BKP/

root@192.168.100.254's password:
sending incremental file list

sent 135 bytes  received 13 bytes  42.29 bytes/sec
total size is 22.25K  speedup is 150.32
```

Nakon što smo pokrenuli naredbu, `rsync` ona se spaja na udaljeno računalo (192.168.100.254) kao trenutni korisnik (`root`). Potom nas traži lozinku za `root` korisnika na tom (udaljenom) računalu. ***Za transport se standardno koristi SSH protokol.***
Zatim kreće sinkronizacija datoteka i direktorija s našeg računala; iz direktorija `/root/test/` na udaljeno računalo u njegov direktorij `/BKP/`. Vezano za komprimiranje, u slučaju kada na našoj izvorišnoj strani (odakle kopiramo), imamo i neke komprimirane datoteke ili datoteke koje nema potrebe niti smisla komprimirati, tada ih možemo isključiti iz procesa komprimiranja.
Ako ***ne želimo*** komprimirati datoteke sa sljedećim ekstenzijama:

- `.gz` – to bi bile već komprimirane datoteke sa *GZIP algoritmom.*
- `.jpg` – slike koje su već komprimirane pa ih nema smisla ponovno komprimirati.
- `.mp3` ili `.mp4` – audio ili video koji je isto već komprimiran.
- `.7z` – ovo je *7 ZIP* dakle isto ne želimo ponovno komprimirati jer je već komprimirano - nema efekta.
- `.bz2` – ovo je *BZIP* 2 kompresija - isto ne želimo ponovno raditi nepotrebno.

To možemo napraviti na sljedeći način (sve u jednom retku):

```
rsync-avhXAHz  --skip-compress=gz/jpg/mp[34]/7z/bz2 /root/test/
root@192.168.100.254:/BKP/

root@192.168.100.254's password:
sending incremental file list

sent 135 bytes  received 13 bytes  42.29 bytes/sec
total size is 22.25K  speedup is 150.32
```

U gore navedenom primjeru podaci se kopiraju na drugo udaljeno računalo na način da se već komprimirane datoteke ne komprimiraju te se sve kopira preko mreže putem ***SSH*** protokola, dakle sigurno i kriptirano.

**2.2.** Napravimo sve isto, također kroz kriptirani ***SSH*** tunel prema drugom računalu, na koje sve kopiramo.



Za ovu vježbu nam je također potreban korisnički račun koji može koristiti pristup preko ***SSH*** protokola.
Ponekad je `SSH` pristup korisniku `root` zabranjen pa je potrebno rekonfigurirati `SSH` servis/*daemon* kako bi se `root` korisniku omogućio `SSH` pristup.

Za korištenje ***SSH*** protokola za spajanje (koje je sada standardno) možemo koristiti još i prekidač `-e ssh`, uz koji možemo navoditi i neke opcije koje ćete vidjeti u primjerima koji slijede.

Kako bismo pojednostavili prikaz, izbacit ćemo dio u kojemu se isključuju već komprimirane datoteke. Krenimo s primjerom:

```
rsync -avhXAHz -e ssh /root/test/ root@192.168.100.254:/BKP/

root@192.168.100.254's password:
sending incremental file list

sent 135 bytes  received 13 bytes  42.29 bytes/sec
total size is 22.25K  speedup is 150.32
```

Procedura je ista; nakon upisane lozinke, kreće prijenos podatka.

Za one koji žele potpuni primjer kao i prethodni uz ***SSH***, to bi bilo ovako (sve jednom retku):

```
rsync -avhXAHz -e ssh --skip-compress=gz/jpg/mp[34]/7z/bz2 /root/test/
root@192.168.100.254:/BKP/

root@192.168.100.254's password:
sending incremental file list
sent 135 bytes  received 13 bytes  59.20 bytes/sec
total size is 22.25K  speedup is 150.32
```

Kopiranje u svim slučajevima iz primjera **2.**x je moguće i u drugom smjeru: udaljeno računalo → lokalno računalo.

**2.3. Sada idemo u drugom smjeru**: s našeg računala preko ***SSH*** tunela, prema drugom računalu na koje kopiramo, ali preko drugog TCP porta za *SSH* (ne standardnog TCP porta 22). Za ovo nam je potreban korisnički račun koji može koristiti ***SSH*** pristup i naravno razmijenjeni *SSH* ključevi između oba računala, jer ne želimo unositi lozinku za pristup *ssh* servisu na udaljenom poslužitelju, svaki put kada pokrećemo ovu `rsync` naredbu.

Pri tome sve napisano mora biti u jednom retku:

```
rsync -avhXAHz --exclude 'data/cache' -e "ssh -p11222" /var/www/html/dokuwiki/
root@192.168.100.254:/BKP/www/dokuwiki/

receiving incremental file list
sent 52.51K bytes  received 83.75M bytes  1.12M bytes/sec
total size is 1.82G  speedup is 21.70
```

U ovom slučaju smo sinkronizirali (kopirali) naš lokalni direktorij `/var/www/html/dokuwiki/` u kojemu se nalazi naša cijela *wikipedija*, na udaljeno računalo (192.168.100.254), preko *ssh* kanala, i to preko drugog ***SSH*** porta, tj. `11222` jer smo tako konfigurirali ***SSH*** servis na odredišnom poslužitelju (192.168.100.254), zbog sigurnosnih razloga.

U prethodnom primjeru, dodatno smo odlučili kako pôddirektorij s našeg poslužitelja, pune putanje: `/var/www/html/dokuwiki/data/cache/` ne želimo sinkronizirati jer su u njemu privremeni podaci koje ne želimo stalno kopirati/sinkronizirati. Vidljivo je kako smo ovdje naveli sâmo relativnu putanju do tog direktorija: `data/cache` jer `rsync` kreće od vršne putanje: `/var/www/html/dokuwiki/`.

Za ovu namjenu smo koristili prekidač: `--exclude`. Ostale prekidače (`-avhXAHz`) smo već prije objasnili.
Osim ***SSH*** protokola za siguran prijenos podataka, moguće je koristiti i nesigurniju metodu prijenosa na način da se na udaljenom računalu mora pokrenuti **rsync** servis (na ***TCP*** portu ***873***), na koji se spajamo s našim ***rsync*** programom (klijentom).

## 3. Napredne mogućnosti

O ovoj cjelini vidjet ćemo i jedan napredniji primjer upotrebe naredbe `rsync` u kojem ćemo obrisati datoteke na odredištu, ako su one obrisane i na izvoru (odakle radimo kopiranje).

Za ovu potrebu možemo koristiti prekidač `--delete`. Osim toga dodati ćemo i mogućnosti kako bi se vidio napredak kopiranja, prekidač: `--progress` koji mora ići u kombinaciji sa prekidačem `-h`.

Prije nego što smo pokrenuli sinkronizaciju, na izvoru (naše računalo), obrisali smo datoteku: `file.out.txt`.

```
rsync -avhXAHz --delete --progress  /root/test/ root@192.168.100.254:/BKP/

root@192.168.100.254's password:
sending incremental file list

deleting file.out.txt

sent 144 bytes  received 13 bytes  44.86 bytes/sec
total size is 22.25K  speedup is 141.71
```

Vidim poruku kako je datoteka: `file.out.txt` obrisana na odredištu tj. na poslužitelju: 192.168.100.254.

Mogući su još i deseci drugih opcija s kojima se možete poigrati i sâmi.

**Izvori informacija: (289),(290),(291),(292),(K-12), `man rsync` .**

### 8.1.3. Relax-and-Recover (ReaR)

Pojam *Relax-and-Recovery* (**ReaR**) odnosi se na metode (i alat) za izradu sigurnosne kopije (*backup*) i oporavak sustava. *ReaR* je dizajniran da olakša proces oporavka sustava nakon neželjenih događaja poput kvara hardvera ili softvera, oštećenja datoteka ili neuspjelog ažuriranja sustava. On radi na principu stvaranja sigurnosne kopije (*backup*-a) sustava i stvaranja medija sposobnog za pokretanje sustava s kojeg se može ponovno pokrenuti sustav u slučaju potrebe za oporavkom.

Naime kada se dogodi kvar na hardveru ili neka softverska greška te sustav ostane u neupotrebljivom stanju, u načelu su potrebna tri koraka da ga se vrati u potpuno funkcionalno stanje, a potencijalno i na novom hardveru. Pri tome pretpostavljamo da ste napravili sigurnosnu kopiju (*backup*) cijelog sustava. Ti koraci su:

**1.** Pokretanje sustava za spašavanje odnosno povrat iz sigurnosne kopije (*backupa*), na novom hardveru.
**2.** Repliciranje izvornog rasporeda particija i formatiranje particija na disku novog računala.
**3.** Povrat svih sistemskih i korisničkih datoteka (servisi i korisničke aplikacije).

Problem je u tome što većina softvera za izradu sigurnosnih kopija (*backup*) rješava samo treći problem. Za rješavanje prvog i drugog problema, na svim *Red Hat* kompatibilnim (i nekim drugim) distribucijama Linuxa dostupan je *Relax-and-Recover* (*ReaR*), uslužni program otvorenog programskog kôda. Pri tome se navedeni *ReaR* koristi na sljedeći način:

- **Kreiranjem sigurnosne kopije (Backup)** – *ReaR* kreira sigurnosnu kopiju cijelog sustava, uključujući diskovne particije i datoteke, ali i druge relevantne informacije koje su potrebne za obnovu sustava.
- **Stvaranje medija sposobnog za pokretanje sustava** – nakon što je sigurnosna kopija sustava stvorena, *ReaR* omogućuje stvaranje medija sposobnog za pokretanje sustava (tzv. *bootable* medij), kao što su DVD, USB diskovi ili ISO slike, koji sadrže sve potrebne alate za potpuni oporavak sustava.
- **Opisivanje i konfiguriranje postupka oporavka** – moguće je konfigurirati različite aspekte procesa oporavka, kao što su lokacija sigurnosnih kopija, mrežne postavke, način oporavka, itd.
- **Oporavak sustava** – u slučaju potrebe za oporavkom, sustav se može pokrenuti s medija sposobnog za pokretanje sustava, stvorenog od strane *ReaR-a*. Zatim se odabire sigurnosna kopiju koju želimo koristiti za oporavak i *ReaR* će automatski provesti proces obnavljanja sustava prema konfiguraciji koju smo unaprijed definirali.

**ReaR** koristi sljedeće datoteke:

- `/etc/rear/local.conf` – osnovna konfiguracijska datoteka *ReaR* sustava.
- `/usr/share/rear/conf/default.conf` – konfiguracijska datoteka *ReaR* sustava s popisom i opisom svih parametara i opcija koje se mogu koristiti.
- `/var/log/rear/` – ovdje se smještaju dnevničke (*log*) datoteke *ReaR* sustava.
- `/usr/share/rear/` – ovdje se nalaze sve skripte *ReaR* sustava.

Sada možemo instalirati *ReaR* sustav:

```
yum -y install rear
```

Osnovni dio konfiguracije se svodi na uređivanje konfiguracijske datoteke `/etc/rear/local.conf`.
Za početak, ovdje ćemo dodati samo nekoliko opcija, koje su nam dovoljne u našem scenariju:

```
OUTPUT=ISO
BACKUP=NETFS
BACKUP_URL=nfs://192.168.1.250/mnt/Drugi-disk/Net-bkp/
```

Pogledajmo značenje nekih od ovih opcija:

- `OUTPUT=` – ovdje se definira izlazni format sigurnosne kopije (*backupa*), koji može biti: `ISO` - zadano, `PXE` - za pokretanje sustava preko mreže, `USB` – USB diskovna slika, `RAWDISK` – za diskovnu sliku koja podržava *UEFI* i *Legacy* boot , `OBDR` – za tračne uređaje.
- `BACKUP_URL=` – ovdje se definira gdje će se snimiti izlazna sigurnosna kopija (*backup*). Neke od mogućnosti su:
  - `file://` – snimanje na lokalni disk, standardno je to u direktorij: `/var/lib/rear/output/`.
  - `nfs://` – snimanje na mrežni *NFS* disk. Mi konkretno koristimo ovu opciju jer imamo *NFS* mrežni poslužitelj na IP adresi **192.168.1.250**, s dijeljenim direktorijem: `/mnt/Drugi-disk/Net-bkp/`.
  - `cifs://` – snimanje na mrežni *CIFS* disk.
  - `ftp://` ili `ftps://` – snimanje na mrežu na odredišni *FTP* ili *FTPS* poslužitelj.
  - `http://` ili `https://` – snimanje na mrežu na odredišni *HTTP* ili *HTTPS* poslužitelj.
  - `sftp://` – snimanje na mrežu na odredišni *SFTP* poslužitelj.
  - `rsync://` – snimanje na mrežu na odredišni *Rsync* poslužitelj, …
- Još jedna važna opcija je `BACKUP=` – za definiciju metode s kojom će se raditi povrat podataka. Ovdje je podržan i poveći broj softvera za tu namjenu, pogledajmo ih:
  - `NBU` – za povrat će se koristiti *Symantec NetBackup*.
  - `AVA` – za povrat će se koristiti *EMC Avamar*.
  - `RBME` ili `RBME` – za povrat će se koristiti *Rsync Backup Made Easy* ili *RSYNC*.
  - `BACULA` ili `BAREOS` – za povrat će se koristiti *BACULA* ili *BAREOS*.
  - `NETFS` – za povrat se koristi jednostavna metoda (koristi običnu ***tar*** arhivu). ***← mi koristimo ovu metodu***

 Uz opciju `BACKUP=NETFS` potrebno je koristiti i `BACKUP_URL=` jer se s njom definira odredišni sustav za *backup*. U konkretnom slučaju s `NETFS` to znači da će u odredišnom *NFS* mrežnom direktoriju biti će kreirane:

- ISO diskovna slika za pokretanje sustava (*system boot*).
- Dodatna `.tar.gz` komprimirana arhiva svih datoteka sa sustava (stvarni *backup* podataka).

U našem slučaju, pošto smo sigurnosnu kopiju (*backup*) direktno snimali na *NFS* mrežni disk, prethodno smo morali instalirati NFS klijent i potrebne biblioteke na naš sustav:

```
yum -y install nfs-utils libnfs libnfs-utils nfsv4-client-utils nfs4-acl-tools
```

Provjerimo radi li nam *NFS* mrežni disk, montiranjem *NFS* mrežnog diska u direktorij `/mnt/nfs/`:

```
mount -t nfs 192.168.1.250:/mnt/Drugi-disk/Net-bkp /mnt/nfs
```

Sada ćemo u novom montiranom direktoriju kreirati direktorij `rear`:

```
cd /mnt/nfs
mkdir rear
```

I sada zapravo na NFS poslužitelju imamo *NFS* putanju: **`/mnt/Drugi-disk/Net-bkp/rear`** koju smo i definirali u *ReaR* konfiguraciji, a koje će se koristiti za pohranu buduće ISO datoteke i *TAR* arhive (`backup.tar.gz`).

**Izrada sigurnosne kopije cijelog sustava (računala)**

Da bismo izradili sigurnosnu kopiju cijelog našeg sustava i ISO datoteku, s koje kasnije možemo pokrenuti *ReaR* sustav i napraviti povrat cijelog sustava iz sigurnosne kopije, trebamo samo pokrenuti naredbu `rear` na sljedeći način:

```
rear -v mkbackup

Relax-and-Recover 2.6 / 2020-06-17
Running rear mkbackup (PID 1457)
Using log file: /var/log/rear/rear-Rocky-9.log
Running workflow mkbackup on the normal/original system
Using backup archive '/var/tmp/rear.i7eUeeZ5pi8Uh6c/outputfs/Rocky-9/backup.tar.gz'
Using autodetected kernel '/boot/vmlinuz-6.6.11-1.el9.elrepo.x86_64' as kernel in the recovery
Creating disk layout
Overwriting existing disk layout file /var/lib/rear/layout/disklayout.conf
GRUB found in first bytes on /dev/sda and GRUB 2 is installed, using GRUB2 as a guessed
bootloader for 'rear recover'
Verifying that the entries in /var/lib/rear/layout/disklayout.conf are correct ...
Creating recovery system root filesystem skeleton layout
To log into the recovery system via ssh set up /root/.ssh/authorized_keys or specify
SSH_ROOT_PASSWORD
Copying logfile /var/log/rear/rear-Rocky-9.log into initramfs as '/tmp/rear-Rocky-9-…
Copying files and directories
Copying binaries and libraries
Copying all kernel modules in /lib/modules/6.6.11-1.el9.elrepo.x86_64 (MODULES)
Copying all files in /lib*/firmware/

Making ISO image
Wrote ISO image: /var/lib/rear/output/rear-Rocky-9.iso (548M)
Copying resulting files to nfs location
Saving /var/log/rear/rear-Rocky-9.log as rear-Rocky-9.log to nfs location
Copying result files '/var/lib/rear/output/rear-Rocky-9.iso

Making backup (using backup method NETFS)
Creating tar archive '/var/tmp/rear.i7eUeeZ5pi8Uh6c/outputfs/Rocky-9/backup.tar.gz'
Archived 1755 MiB [avg 14153 KiB/sec] OK
Archived 1755 MiB in 128 seconds [avg 14042 KiB/sec]
Exiting rear mkbackup (PID 1457) and its descendant processes ...
Running exit tasks 🡨 Ovaj komentar označava da je sve uredno završilo !
```

🡨 Rezultat jest ISO datoteka, backup datoteka (*TAR*) i pripadajuće datoteke kreirane u novi direktorij na *NFS* mrežnom dijeljenom disku, unutar poddirektorija koji je kreiran u tom procesu, a koji nosi ime našeg računala: **Rocky-9**.

Nama je, pošto smo već ručno montirali navedeni *NFS* mrežni disk, sve vidljivo i ovako (*stavili smo opise uz ispis*):

```
ls -al /mnt/nfs/rear/Rocky-9/

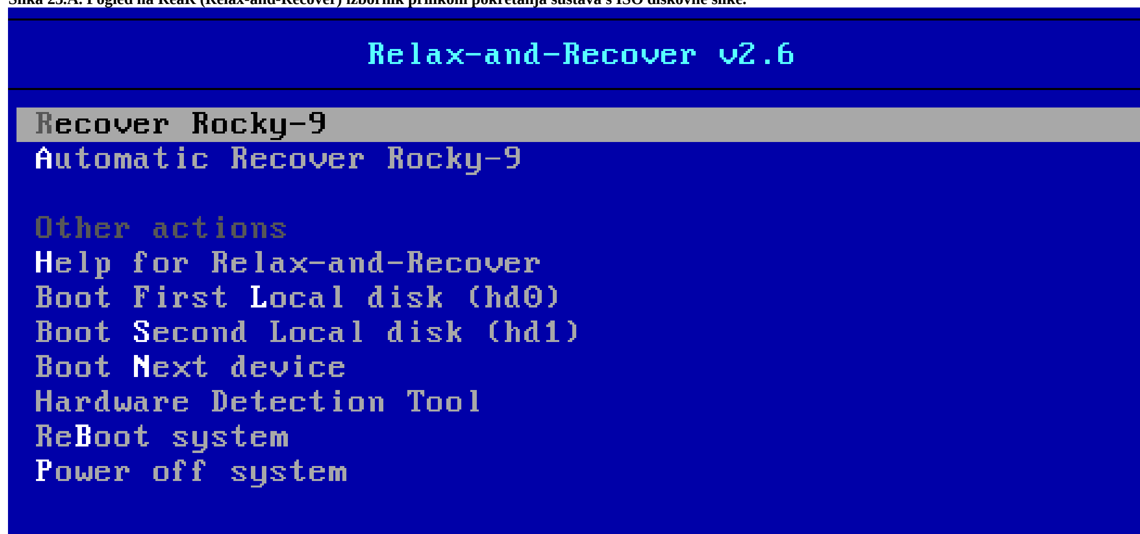
1.8G Mar 10 13:42 backup.tar.gz 🡨 Backup podataka
210K Mar 10 13:48 backup.log    🡨 Log za backup podataka
548M Mar 10 13:50 rear-Rocky-9.iso 🡨 ISO ReaR backup
201K Mar 10 13:50 rear-Rocky-9.log 🡨 Log za ISO ReaR backup
```

**Povrat (recovery) sustava iz ISO datoteke**

Za pokretanje porata cijelog sustava, koji smo prethodno napravili, imamo dvije opcije: ili ISO diskovnu sliku (`rear-Rocky-9.iso`) snimiti na DVD ili pomoću programa poput: ***Rufus***, ***Win32diskimager***, **Ventoy** i sličnih, snimiti ISO sliku na USB disk s kojega moramo pokrenuti novo računalo (*na kojem radimo recovery proceduru*). Krenimo redom:

1. Pokrenuti novo računalo odnosno ono na koje radimo povrat sustava s *ReaR backupa* s DVD-a ili USB-a.
2. Pojavit će se izbornik u kojemu trebamo odabrati: "**`Recover Rocky-9`**".

**Slika 25.A. Pogled na ReaR (Relax-and-Recover) izbornik prilikom pokretanja sustava s ISO diskovne slike.**




3. Nakon nekoliko trenutaka *ReaR* će se pokrenuti u naredbenom retku.
4. U *ReaR* naredbenom retku, potrebno je logirati se kao `root`.
5. Tek ovdje pokrećemo proceduru povrata (engl. *recovery*). Stoga je potrebno upisati:

```
rear recover
```

← Potom će se pokrenuti procedura izrade particije vašeg diska te formatiranja particije s datotečnim sustavom. Ovdje ne moramo ništa odgovarati jer je automatski postavljeno odnosno zadano da se nakon 30.s. nastavi dalje.

6. U ovom koraku, pošto smo mi konkretno koristili `BACKUP=NETFS` s pripadajućim *NFS* poslužiteljem, *ReaR* se spaja na udaljeni mrežni *NFS* disk (**A**) gdje se nalazi arhiva sa svim podacima. Potom se kreiraju particije i formatiraju s datotečnim sustavom (**B**) te se sadržaj arhive s *NFS-a* kopira na naš novokreirani datotečni sustav (**C**).

**Slika 25.B. Pogled na ReaR (Relax-and-Recover) prilikom povrata podatak iz sigurnosne kopije.**




7. Kada je *ReaR* završio rad s particijama i kopiranjem podataka iz arhive s *NFS* poslužitelja, pojavit će se novi naredbeni redak, iz kojega možemo ugasiti računalo. Tijekom ovog procesa kreirat će se i **GRUB2 boot loader**.

```
Rear> shutdown -h now
```



***Nakon što je računalo ugašeno uklonite ReaR USB ili DVD te ponovno pokrenite računalo.***
***Nakon nekoliko minuta, nakon što sustav osvježi sve labele, sustav je spreman odnosno recovery procedura je završila !***

**Izvori informacija: (1607),(1608),(K-12), `man rear`.**

# 8.2. Rad s tračnim uređajima

***Slijedi napredno poglavlje!***
Tračni uređaj je uređaj za pohranu podataka koji koristi magnetsku vrpcu odnosno traku kao medij za pohranu podataka. Spremanje odnosno čuvanje podataka na magnetskim vrpcama obično se koristi za dugotrajnu arhivsku pohranu podataka.
Na vrpcu se podaci zapisuju i čitaju sekvencijalno, za razliku od pogona tvrdog diska, koji pohranjuje i čita podatke izravnim pristupom. Kod čitanja ili zapisivanja podatak na klasični disk, diskovni pogon (glava) može se pozicionirati na bilo koji položaj na disku za nekoliko milisekundi, dok tračni pogon mora fizički premotati vrpcu, kako bi pročitao svaki pojedini blok podataka.

Kao rezultat toga, tračni uređaji imaju vrlo sporo prosječno vrijeme pristupa podacima. Ipak, tračni uređaji mogu brzo prenositi podatke s trake, kada je postignut željeni položaj odnosno kada je vrpca premotana i pozicionirana na točnu traženu poziciju. Na primjer, od 2020. godine ***Linear Tape-Open*** (**LTO**) tračni uređaji (LTO 9 konkretno) podržavaju kontinuirane brzine prijenosa podataka do 400 MB/s to jest 1.000 MB/s komprimirano, što je brzina prijenosa itekako usporediva s tvrdim diskovima.
Stoga se za potrebe izrade sigurnosnih kopija (engl. *backup*), najviše i koriste tračni uređaji odnosno same trake kao mediji za pohranu podataka.

Pogledajmo neke od slabo poznatih činjenica o trakama:

- Podaci se na vrpce (trake) zapisuju sekvencijalno odnosno u nizu, jedan za drugim.
- Trake mogu spremiti velike količine podataka (pr. **LTO 8**: 12 TB, tj. za **LTO 9** je to 18 TB po traci).
- Najjeftiniji su pouzdani medij za pohranu velike količine podataka.
- I danas se koriste za pohranu podataka (*backup*) koja se obično negdje pohranjuje na određeno (duže) vrijeme.

Trenutno na tržištu postoji cijeli niz standarda tračnih uređaja i pripadajućih traka, poput: **DDS** (*Digital Data Storage*), **DLT** (*Digital Linear Tape*) i **SDLT**, **LTO** (*Linear Tape-Open*) i drugih. Pri tome je **LTO** otvoren standard, čiji razvoj podržavaju najveći proizvođači tračnih uređaja, poput tvrtki: ***Hewlett Packard Enterprise***, ***IBM i Seagate*** te tvrtka ***Quantum***, svi unutar *LTO* konzorcija koji i razvija ovaj standard.

Pogledajmo specifikaciju osnovnih karakteristika **LTO** traka, jer se ova vrsta traka najviše i koristi:

| **Naziv LTO standarda** | **LTO 1** | **LTO 2** | **LTO 3** | **LTO 4** | **LTO 5** | **LTO 6** | **LTO 7** | **LTO 8** | **LTO 9** | **LTO 10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Datum prve upotrebe** | 2000.g. | 2003.g. | 2005.g. | 2007.g. | 2010.g. | 2012.g. | 2015.g | 2017.g | 2021.g. | U pripremi |
| **Nazivni Kapacitet (bez kompresije)** | 100 GB | 200 GB | 400 GB | 800 GB | 1.5 TB | 2.5 TB | 6 TB | 12 TB | 18 TB | 48 TB ? |
| **Brzina zapisivanja na traku: [MB/s] nekomprimirano** | 20 | 40 | 80 | 120 | 140 | 160 | 300 | 360 | 400 | 1.100 ? |
| **[MB/s] komprimirano** | 40 | 80 | 160 | 240 | 280 | 400 | 750 | 900 | 1.000 | ? |
| **Hardverska kompresija i omjer kompresije** | ALDC 2:1 | ALDC 2:1 | ALDC 2:1 | ALDC 2:1 | ALDC 2:1 | LTO-DC 2.5 : 1 | 2.5 : 1 | 2.5 : 1 | 2.5 : 1 | Planirano 2.5 : 1 |

Osim navedenog spomenimo još nekoliko važnih stvari o tračnim uređajima i trakama.
Naime trake su izdržljive te omogućavaju do nekoliko tisuća ubacivanja i izbacivanja iz tračnog uređaja.
Osim toga, one omogućavaju i nekoliko stotina prepisivanja cijele trake.

Vezano za LTO uređaje (pogone) i trake (vrpce) odnosno takozvane uloške, važno je znati i sljedeće:

- ✓ Sve do i uključujući **LTO-7**, takozvani *Ultrium (LTO)* pogon može čitati podatke s uloška (trake) u vlastitoj generaciji i dvije prethodne generacije. Primjerice LTO-8 pogoni (uređaji) mogu čitati LTO-7 i LTO-8 uložak (traku), ali ne i LTO-6 uložak.
- ✓ *Ultrium* pogon može pisati podatke na uložak (traku) u vlastitoj generaciji i na uložak iz prethodne generacije u formatu prethodne generacije.
- ✓ Neki LTO-8 pogoni mogu pisati prethodno nekorištene LTO-7 trake s povećanim, nekomprimiranim kapacitetom od 9 TB (tip M (M8)). Samo novi, neiskorišteni LTO-7 ulošci mogu se inicijalizirati kao LTO-7 Tip M. Nakon što se uložak inicijalizira kao Tip M, ne smije se mijenjati natrag u 6 TB LTO-7 uložak. LTO-7 Type M ulošci (trake) samo su inicijalizirane na Tip M u LTO-8 pogonu. LTO-7 diskovi ne mogu čitati LTO-7 Type M uloške.
- ✓ Određeni *Ultrium* pogon ne može koristiti uložak novije generacije. Na primjer, LTO-2 uložak nikada ne može koristiti LTO-1 pogon i iako se može koristiti u LTO-3 pogonu, radi kao da je u LTO-2 pogonu.

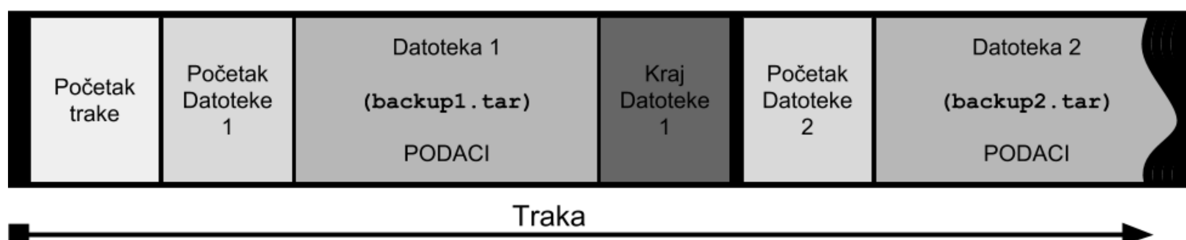
Slika **25.B.** na sljedećoj stranici pokazuje kako izgledaju:

- **DDS4** tračni uređaj (oznaka **1**).
- **LTO 2** tračni uređaj (oznaka **2**).
- Kao i pripadajuće trake za pohranu podataka:
  - **DDS4** (oznaka **3**).
  - **LTO2** (oznaka **4**).
  - **LTO1** (oznaka **5**).

**Slika 25.B. Pogled na tračni uređaj i razne vrste traka. Autor fotografije: Asmir Bašić**.

Kako smo to već spomenuli, podaci se na trake zapisuju u nîzu jedan za drugim, kako je vidljivo na slici 25.1, dolje.

**Slika 25.1. Pogled na zapisivanje podataka na traku (vrpcu.)**




Naime vidljivo je kako se na samu traku (vrpcu) kao medij za pohranu podataka, zapisuju datoteke u nîzu, jedna za drugom.
S time, kako bi se tračni uređaj mogao pozicionirati na pravilnu poziciju, postoje i kontrolni zapisi, kao i zapisi koji označavaju početak i kraj svake zapisane datoteke na traci odnosno vrpci.
Za rad s tračnim uređajima i trakama koristi se naredba `mt` (engl. ***m**agnetic **t**ape control*).

Neke od opcija i parametara naredbe `mt` su sljedeći:

- `-f` – *file device*, slijedi ime uređaja **`/dev/`**...
- `rewind` – premotaj traku na početak.
- `retension` – premotaj na kraj trake pa ponovno na početak (zbog izjednačavanja napetosti trake).
- `status` – ispiši status tračnog uređaja.
- `reserve` – rezerviraj (i nakon završetka snimanja).

Slijedi nastavak osnovnih opcija i parametara naredbe `mt`:

- `release` – oslobodi traku za drugi rad.
- `erase` - obriši cijelu traku.
- `offline` - izbaci traku.

Kako bismo uopće pronašli tračni uređaj, koristit ćemo naredbu `lsscsi` na sljedeći način:

```
lsscsi -g

[0:0:0:0]    storage HP       P220i            6.34  -          /dev/sg0
[0:1:0:0]    disk    HP       LOGICAL VOLUME   6.34  /dev/sda   /dev/sg1
[6:0:1:0]    tape    IBM      ULT3580-TD5      0104  /dev/st0
```

Vidimo kako je naš tračni uređaj (`tape`) uređaj onaj najdonji u ispisu, s oznakom: `/dev/st0`.
Iste informacije možemo vidjeti ispisom sljedeće posebne */proc* datoteke:

```
cat /proc/scsi/scsi
```

Sada moramo instalirati program za rad s trakom:

```
yum -y install mt-st
```

Zatim ubacimo novu traku (vrpcu) u tračni uređaj.

**Krenimo s primjerima:**

**1.** Pogledajmo status trake i tračnog uređaja (`/dev/st0`):

```
mt -f /dev/st0 status

SCSI 2 tape drive:
File number=0, block number=0, partition=0.
Tape block size 0 bytes. Density code 0x8c (EXB-8505 compressed).
Soft error count since last status=0
General status bits on (41010000):
 BOT ONLINE IM_REP_EN
```

Ovdje sve izgleda uredno jer nemamo nikakvih grešaka.

**2.** Premotajmo traku na tračnom uređaju `/dev/st0` na početak, pomoću naredbe `mt`:

```
mt -f /dev/st0 rewind
```

**3.** Provjerimo na kojem bloku se traka trenutno nalazi

```
mt -f /dev/st0 tell

At block 0
```

Dakle traka je sada na početnom bloku (`0`).

**4.** Snimimo sve podatke (datoteke) iz direktorija (mape) `/VIRTUALKE` na našu traku:

```
tar -czf /dev/st0 /VIRTUALKE/
```

**4.1** Sada ispišimo sve datoteke na traci, na tračnom uređaju: `/dev/st0`

```
tar -tzf /dev/st0

VIRTUALKE/
VIRTUALKE/vm-harddisk.qcow2
VIRTUALKE/vm-disk1.qcow2
```

Vidimo samo ono što smo snimili dakle konkretno slike diskova (*disk image*) od virtualnih računala.

**5.** Zatim snimimo i sve iz vršnog direktorija (mape) `/home/` na traku:

```
tar -czf /dev/st0 /home/
```

**5.1** Sada ispišimo sve datoteke na traci, na tračnom uređaju: `/dev/st0` (skratili smo ispis):

```
tar -tzf /dev/st0

home/
home/hrvoje/
home/hrvoje/.bash_logout
```

Vidimo samo ono što smo dodatno snimili (dosnimili), dakle konkretno samo sadržaj direktorija (mape) `/home/`.

**6.** Vratiti se sâmo jedan zapis ranije možemo napraviti s naredbom:

```
mt -f /dev/st0 bsfm 1
```

Odnosno u našem slučaju se možemo vratiti na sami početak trake jer imamo sâmo jedan zapis prije:

Skakanje na prvi slijedeći zapis (jedan više) radimo s naredbom:

```
mt -f /dev/st0 fsf 1
```

**7.** Napravimo povrat direktorija `/home` s trake na disk (ako se nalazimo na tom zapisu):

```
tar -xzf /dev/st0 home
```

**8.** Obrisati cijelu traku u tračnom uređaju: `/dev/st0` možemo napraviti sa:

```
mt -f /dev/st0 erase
```

**Izvori informacija: (990), `man mt`, `man tar`.**

# 9. Proces menadžment

U ovom poglavlju upoznat ćemo se s nekoliko važnih mehanizama u svakom operativnom sustavu, vezanih za cijeli životni vijek programa te komunikaciju svakog programa s operativnim sustavom, kao i komunikaciju među programima.
Svaki ***proces*** predstavlja pokrenuti program odnosno naredbu ili skriptu na linuxu ili bilo kojem drugom operativnom sustavu. Dakle svaki pokrenuti: program, naredba, naredbeni redak (*shell)* ili neka skripta se smatra jednim Linux procesom. O cijelom životnom vijeku procesa se brine Linux *task/process scheduler*. Svaki proces u grubo može pripadati u dvije kategorije:

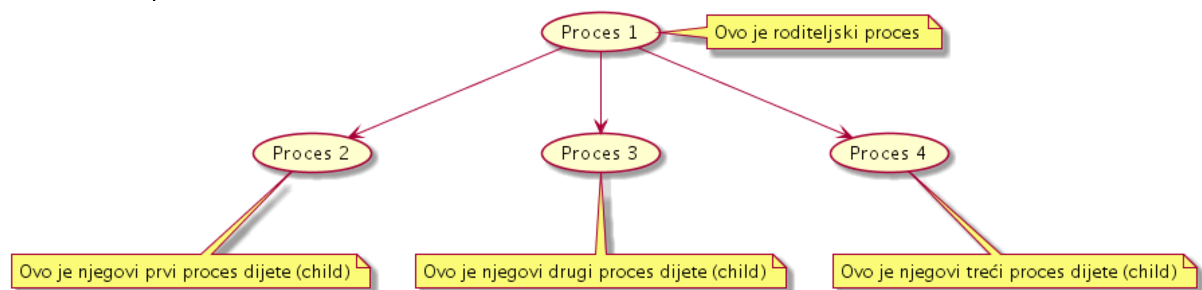
- ***Normalni*** *proces*: ovoj kategoriji pripada velika većina procesa odnosno pokrenutih programa.
- ***Proces koji se izvršava u realnom vremenu (engl. Real time)***: odnosno procesi koji moraju reagirati iznimno brzo na sve promijene ili događaje. Ovakvi procesi se obrađuju na drugačiji način od "normalnih" procesa jer su u pravilu to posebni sistemski procesi koji zahtijevaju vrlo brzi odziv sustava.

Ako pak gledamo procese prema načinu rada ili pokretanja, možemo ih rasporediti u nekoliko kategorija:

- ***Roditeljski*** *proces* (engl. *parent process*) je proces koji može pokrenuti neki drugi proces. Dakle svaki proces koji pokrene neki drugi pôd proces je *roditeljski* proces, tom pokrenutom procesu. Svi procesi osim posebnih: `init` ili `systemd` procesa, imaju barem jedan roditeljski proces.
- ***Proces dijete*** (engl. *child process*) je svaki onaj proces koji je pokrenut od strane *roditeljskog* procesa. Proces dijete, nasljeđuje određene atribute roditeljskog procesa. Pri tome, bilo koji proces može kreirati pôd proces (proces dijete).
- ***Proces bez roditelja*** (engl. *orphan process*) je onaj proces koji je nastavio s radom, iako je njegov roditeljski proces zaustavljen ili ugašen, zbog bilo kojeg od razloga:
    - Neplanirano: zbog greške ili rušenja roditeljskog procesa.
    - Planirano: jer ovakav proces treba nastaviti raditi, iako je proces koji ga je kreirao (roditelj), ugašen.
      Ovo može biti u slučaju kada se radi o procesu koji radi kao servis, kojem više ne treba proces roditelj poput naše ljuske iz koje smo ga pokrenuli. Drugi primjer je proces koji se mora nastaviti izvršavati čak i ako se prekine rad *ljuske* iz koje smo ga pokrenuli (tzv. `nohup` rad, koji ćemo kasnije spomenuti).
- ***Servis (****engl. daemon)* proces odnosno proces koji je aktivan kao *servis* je onaj koji radi u pozadini, bez potrebe za direktnom kontrolom od strane korisnika. Obično su roditeljski procesi (*daemon)* procesa: ili `init` ili `systemd` proces, a u nekim slučajevima može biti *shell* proces (ljuska) iz koje ga pokrećemo.
- ***Zombi (****engl. zombie)* proces je proces koji je ostao u stanju u kojem je trebao biti zaustavljen, ali i dalje radi, iako više nije u tablici/listi procesa. Drugi primjer je vrlo kratki djelić sekunde u kojem se ovakav proces počinje gasiti, pa više nema ***PID*** broj, niti se nalazi u tablici/listi procesa, ali njegovi resursi još nisu oslobođeni (u datom trenutku).

Veza između roditeljskog procesa i procesa djeteta (množine) je vidljiva na slici **26.**

**Slika 26. Veza između procesa**




Vezano za rad s procesima, naredbama `ps` i `top` možemo izlistati aktivne (pokrenute) programe odnosno procese, čiji status ćemo proučiti u tekstu koji slijedi. Važno je razumjeti kako svaki proces identificira nîz identifikatora i parametara od kojih nas sada zanimaju samo sljedeći.

Kasnije ćemo kroz primjere vidjeti i kako to izgleda u radu (primjerice poglavlja: **9.3.1.1.** i **10.7.2.3.1.**)

- `UID` – označava tko je vlasnik procesa ***UID*** znači *„User ID" odnosno identifikator korisnika.*
- `PID` – jedinstveni je identifikacijski broj procesa (*Process ID [**PID**])* koji sustav dodjeljuje svakom novom procesu.
- `PPID` – je identifikator procesa roditelja (*parent*) koji je pokrenuo ovaj proces odnosno *„Parent Process ID"*.
- `STAT` – prikazuje u kojem je stanju rada određeni proces.
- `STIME` – prikazuje vrijeme kada je proces pokrenut.
- `TIME` – prikazuje vrijeme koliko je CPU radio na procesu (obrađivao ga).
- `CMD` – naredba ili program (ili skripta) koja je pokrenula ovaj proces.

Ako gledamo polje `STAT`, primjerice ispisa naredbi `ps` ili `top`, ali i općenito, procesi mogu biti u nekoliko mogućih stanja:

- `R` - *`Running`* – stanje označava da je proces pokrenut ili kako je u fazi pokretanja, pri čemu se čeka dodjeljivanje određenom procesoru odnosno konkretno jezgri procesora (CPU-a) na obradu.
- *`Waiting`* – ovo je stanje u kojem proces čeka na neki događaj ili traži pristup nekom sistemskom resursu (disk, mreža, CPU, …). Ovdje postoje dvije podvrste navedenog stanja:
  - `S` - *`interruptible`* – odnosno procesi koji mogu biti prekinuti (pauzirani i sl.) dok su u stanju čekanja. Označava procese koji su u stanju čekanja na druge procese ili na svoje *pod procese* ili procesne nîti.
  - `D` - *`uninterruptible`* – ovo su oni procesi koji ne smiju biti prekinuti dok su u stanju čekanja jer čekaju na neki hardverski zahtjev ili pristup nekom uređaju pa stoga ne smiju biti prekidani u radu.
- `T` - *`Stopped`* – označava proces koji je zaustavljen, na način u kojem je proces primio neki od signala za zaustavljanje. Primjerice procesi koji su u stanju *debugging*a su isto vidljivi kao *`Stopped`*.
  Za signale koji se mogu poslati procesima pogledajte poglavlje: **9.2 Procesi i signali koje im možemo poslati**.
- `Z` - *`Zombie`* – ovo su zaustavljeni procesi čiji resursi zbog nekih razloga još nisu oslobođeni pa su ostali kao "*mrtvi*" procesi. Proces u ovom stanju obično zauzima vrlo malo resursa i to često sâmo memoriju potrebno za pohranjivanje njegovog ***Proces Deskriptora*** kao i rezervaciju ***PID*** broja. Tijekom normalnog zaustavljanja procesa u nekom djeliću vremena svaki proces se nađe u tom stanju, ali već nakon nekoliko trenutaka, što je toliko brzo da nam je to uglavnom nevidljivo, proces u konačnici biva potpuno zaustavljen, a svi sistemski resursi koji su pripadali procesu, oslobođeni. Jedan od oslobođenih resursa pri tome je i ***PID*** broj, kako bi bio slobodan za upotrebu za neki novi pokrenuti program odnosno proces.

Stanje procesa ima još jednu dodatnu oznaku nakon prve koju smo objasnili, a ona može biti:

- `<` - proces s visokom prioritetom.
- `N` - proces s niskim prioritetom.
- `L` - proces ima označene i zaključane regije memorije (*Memory pages*). Koristi se za *realtime* i *I/O* procese.
- `s` - ovaj proces je voditelj grupne sesije (engl. *Session leader*).
- `l` - višenitni proces (engl. *Multi-threaded*).
- `+` - predstavlja prednju procesnu grupu (engl. *Foreground*).

## Primjer upotrebe naredbe `ps`

U primjeru upotrebe naredbe `ps` koja ispisuje pokrenute procese možemo koristiti i prekidač `-a` kako bismo vidjeli sve detalje. Sve naše komentare smo dodali u ispis zatamnjeno:

```
ps -ef

UID   PID PPID C STIME TTY TIME     CMD
root  1      0 0 Nov14 ?   00:00:01 /sbin/init
root  2      0 0 Nov14 ?   00:00:00 [kthreadd]
... Slijede kernel procesi ...
root  3      2 0 Nov14 ?   00:00:00 [migration/0]
root  4      2 0 Nov14 ?   00:06:03 [ksoftirqd/0]
... Ispod su "korisnički" procesi/programi ...
root 150     1 0 Nov14 ?  00:00:00 /sbin/udevd -d
root 555     1 0 Nov14 ?  00:00:06 /sbin/rsyslogd -i /var/run/syslogd.pid -c 5
root 783     1 0 Nov14 ?  00:00:04 /usr/sbin/sshd
```

Vidimo kako je `init` proces pokrenuo sve ostale procese, koji su logički pôd njim. Njegov ***PID*** broj je jedan (`1`), a njegov roditeljski (*parent*) proces je nula (`0`) što označava sâm *kernel* jer njega (***init***) uvijek i isključivo pokreće sâmo i isključivo kernel.

Na izlistanju statusa vidimo procese koje je `init` pokrenuo. To su zapravo svi ostali procesi odnosno programi na sustavu jer je upravo ***init*** zadužen za pokretanje svih programa i servisa na sustavu (Linuxu). Ovdje vidimo i:

- ***PID*** `150` - `udevd` servis (*daemon*) zadužen za prepoznavanje hardvera.
- ***PID*** `555` - `syslog` servis (*daemon*) zadužen je za *logiranje* poruka.
- ***PID*** `783` - `ssh` servis (*daemon*) zadužen je za udaljeni mrežni pristup preko *SSH* protokola.

Vidljivo je za procese s ***PID*** brojevima: `150`, `555` i `783`, kako je svima njima ***PPID***=`1`, što je ***PID*** broj od `init` procesa.
To znači kako ih je sve pokrenuo `init` proces jer im je on roditeljski proces. Na ***RedHat/CentOS*** **7.x/8.x** i nekim drugim distribucijama Linuxa je `init` proces zamijenjen sa `systemd` procesom, ali o tome kasnije.



Vezano za naredbu **`ps`**, pogledajte i poglavlja:
**10.7.2.3.1. Naredba *ps*.**
**12.3.3.4. Naredbe *ps* i *top* za memoriju.**

## Slijedi napredni dio

Zanimljivo je i to što postoji još jedan poseban proces, čiji roditeljski proces je isto sâm *kernel*, kao i za `init` proces (odnosno `systemd` za *RedHat/CentOS* **7**.x+). To je proces imena: `kthreadd` koji ima ***PID*** broj dva (`2`). On je poseban proces koji radi na vrlo niskoj razini unutar takozvanog *kernel prostora*, odnosno unutar direktnog adresnog prostora kernela. Ovaj proces je zadužen za pokretanje novih kernel nîti odnosno kernel procesa (engl. *Kernel threads*). Namjena ovih posebnih kernel procesa je njihovo izvršavanje na vrlo niskoj razini odnosno ispod razine svih drugih programa/aplikacija, a njihovu zadaću čine razne funkcionalnosti koje se logički nalaze u komponentama kernela. Posebni kernel procesi imaju u nazivu uglate zagrade poput `[kthreadd]` po čemu ih se također može identificirati. Ti procesi koje zapravo indirektno pokreće sâm kernel, preko procesa `kthreadd` su procesi koji upravljaju osnovnim funkcijama operacijskog sustava. Upoznajmo se i s nekima od njih:

- Za funkcionalnosti *kernela*, vezane za procese i sistemske pozive, *timere*, signale prekida (*interrupts*), komunikaciju s ***I/O*** sustavom odnosno pod komponentama i slično, zadužen je `kworker` kernel *proces/thread.*
- Za kontrolu pripadnosti CPU jezgri i memoriji za pojedini proces, zadužen je kernel proces: `cpuset`.
- Za raspodjelu procesa na izvršavanje, na CPU jezgre, zadužen je: `migration` kernel *proces/thread.*
- Od raznih dijelova diskovnog podsustava, tu primjerice imamo slijedeće kernel procese:
    - `aio` – zadužen je za asinkroni pristup diskovnom podsustavu.
    - `kswapd` – zadužen je za *swap* funkcionalnosti.
    - `flush` – zadužen je za zapisivanje podataka iz disk među memorije (*cache*), periodički, na disk.
    - `kblocks` – zadužen je za kontrolu i pristup diskovnom sustavu na najnižoj razini.
    - `kintegrityd` – zadužen je za integritet podataka koji se spremaju prema diskovnom podsustavu.
    - `scsi_eh_*` – on je dio diskovnog podsustava, zadužen za SCSI funkcionalnosti (to vrijedi i za *SATA/SAS i SCSI* diskove, kao i *VIRTIO*). Postoji po jedan ovakav proces za svaki *SCSI/SAS/SATA* diskovni kanal.
- Za sustav za baratanje softverskim signalima prekida (*soft interruptima*) je zadužen: `ksoftirqd`.
- Za dio sustava virtualne memorije, zaduženi su primjerice: `kmalloc`, `khugepaged`, `ksmd`.
- Za mrežni sustav, primjerice za *agregaciju*/*bonding*/povezivanje mrežnih kartica je zadužen: `bond`.
- Osim navedenih, postoji i cijeli nîz drugih kernel *thread* procesa, svaki za svoju specifičnu namjenu.

**Vratimo se na primjere.**

Pogledajmo primjer upotrebe naredbe `ps` sa prekidačima `-auxf`, koji nam daju prikaz cijelog stabla svih procesa (programa):

```
ps -auxf

USER    PID  %CPU %MEM   VSZ    RSS TTY STAT  START   TIME COMMAND
root     1   0.0  0.0  19236   1452  ?   Ss   Oct01   0:00 init
root    150  0.0  0.0  10644    584  ?   S<s  Oct01   0:00  \_ /sbin/udevd -d
root    555  0.0  0.0 183556   1444  ?   Sl   Oct01   0:00  \_ /sbin/rsyslogd -i /var/run/sys…
root    783  0.0  0.0  66620   1228  ?   Ss   Oct01   0:00  \_ /usr/sbin/sshd
root    784  0.0  0.0  96244   3956  ?   Ss   09:29   0:00  |    \_ sshd: root@pts/0
```

Ovdje također vidimo kako je `init` proces pokrenuo sve ostale procese (koji su pôd njim) te kako je on u stanju `S`.
Isto tako vidimo i kako je proces `sshd` (***PID*** `783`) pokrenuo pôd proces koji ima ***PID*** broj: `784`.
Isto stablo procesa, možemo vidjeti još ljepše, s naredbom `pstree`, ako ju imamo instaliranu:

```
pstree

init─┬─atd
     ├─atop
     ├─cron
     ├─ksmtuned───sleep
     ├─kthreadd─┬─aio/0
     │          ├─ksoftirqd/0
     │          ├─events/0
     │          ├─watchdog/0
     │          ├─scsi_eh_0
     │          ├─nfsiod
  ... ... ... ... ... ...
     ├─udevd
     ├─rsyslogd
```

Izlistanje je skraćeno zbog lakšeg razumijevanja. Naime ovdje je vidljivo koje procese direktno pokreće `init` proces, a koje procese pokreće `kthreadd` proces. Dodatno je i sâmo stablo s granama procesa sada sigurno razumljivije.



Naredba **`ps`** može nam prikazati i ***SELinux*** parametre (`-Z`). Pogledajte poglavlje:
**28.2. SELinux sustav.**

Vezano za procese i sistemske servise pogledajte i poglavlje:
**7.3.3. Sistemski servisi (i procesi).**

## Primjer upotrebe naredbe top

Pogledajmo i primjer upotreba naredbe top koja nam daje puno više informacija o svakom pokrenutom programu (procesu):

```
top

15:24:20 up 1 day, 5:45, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 143 total, 1 running, 142 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.3%us, 0.2%sy, 0.0%ni, 98.4%id, 0.1%wa, 0.0%hi, 0.0%si,
0.0%st
Mem: 16332608k total, 7480892k used, 8851716k free, 89428k buffers
Swap: 16776184k total, 0k used, 16776184k free, 1695512k cached
PID USER PR NI VIRT   RES  SHR    S %CPU %MEM TIME+   COMMAND ←↓
1   root 20 0  19 348  1524 1212  S 0.0  0.0  0:01.27 init
2   root 20 0  0       0    0     S 0.0  0.0  0:00.00 kthreadd
3   root RT 0  0       0    0     S 0.0  0.0  0:00.01 migration/0
4   root 20 0  0       0    0     S 0.0  0.0  6:03.23 ksoftirqd/0
```

Opis značenja, za treći (označeni) red od gore, odnosno **Cpu(s):** slijedi:

- %us – daje nam postotak koliko CPU vremena (resursa) koriste korisnički programi, u tzv. „*user-space*“ dijelu memorije. Dakle ovdje se vode statistike „*korisničkih*“ odnosno *normalnih* programa (procesa) i servisa.
- %sy – daje nam postotak koliko CPU vremena koriste kernel programi, u tzv. „kernel-*space*“ dijelu memorije.
- %ni – nam prikazuje koliko CPU vremena koriste korisnički programi kojima je mijenjan prioritet (*niceness*).
- %wa – nam govori koliko se CPU vremena koristi za ulazno/izlazne (*I/O*) operacije, prema diskovnom sustavu.
- %hi – nam govori koliko se CPU vremena koristi za obradu hardverskih signala prekida (*IRQ*).
- %si – nam govori koliko se CPU vremena koristi za obradu softverskih signala prekida (*soft IRQ*).

Opis značenja, prema stupcima ispisa (najdonji dio ispisa←↓ naredbe top od gore) slijedi:

- PID – ovo je jedinstveni identifikacijski broj (*ID*) svakog pojedinog procesa odnosno „*Process ID*“.
- USER – ovo je korisničko ime korisnika koji je pokrenuo taj proces (engl. *Username*).
- PR – prikazuje efektivni odnosno stvarni prioritet procesa; sustav ga izračunava kao: PR **=** ***20*** **+ (**NI**)**. (**RT**=Realtime)
- NI - *Nice* – ova vrijednost modificira normalni prioritet procesa: od **-20** za najprioritetnije do (+)**20** za manje važne procese odnosno one s najmanjim prioritetom, oduzimajući/dodajući se na PR vrijednost. Pogledajte poglavlje: **9.1.**
- VIRT – ovo je ukupna količina virtualne memorije koju proces može koristiti (RES+***SWAP***+*memory mapped files*).
- RES – ovo je rezidentna veličina (u **kb**) i to “*Non-swapped*” fizička memorija koju proces koristi, bez *swap*-a.
- SHR – ovo je veličina dijeljene memorije (*Shared memory size*) u **kb** – dijeljena memorija je memorija koja se može dijeliti i s drugim procesima.

Slijedi nastavak opisa polja:

- S – označava status procesa, moguće su slijedeće vrijednosti:
  - R - “*Running*” – pokrenuti proces.
  - D - “*Sleeping*” – nije ga moguće prekinuti (tzv. “*Non interrupted*”).
  - S - „*Sleeping*“ – moguće ga je prekinuti (tzv. “*Interrupted*”).
  - T - “*Traced or stopped*” – što znači da se prati ili je zaustavljen.
  - Z - “*Zombie*” ili “*hung*” – označava takozvani *zombi* ili zablokirani proces.
- %CPU – je postotak CPU vremena koje je proces koristio u vrijeme kada je naredba top zadnji puta osvježena. Linux 100% opterećenje jedne CPU jezgre označava kao 100%, dok opterećenje dvije CPU jezgre od 100% označava kao sumu; dakle 200% itd.
- %MEM – je postotak memorije (RAM) koje je proces koristio u vrijeme kada je naredba top zadnji put osvježena.
- TIME+ – je kumulativno CPU vrijeme koje su proces i njegovi pod procesi (*child*/djeca) koristili.
- COMMAND – je ime procesa ili naredbe koja je pokrenuta.
  Pritisnuti slovo/tipku c za prebacivanje između imena procesa i putanje naredbe koja je pokrenula proces.

Sve navedene opcije i polja koja smo ovdje tek naveli ćemo detaljnije upoznati u cjelinama koje slijede.



Pogledajte i poglavlje:
**10.7.2.3.2 Naredba top** za osnove naredbe **top**.

**Izvor informacija: (K-2),(K-5),** **man top**,**man ps**,**man 7 aio**.

# 9.1. Prioriteti i procesi

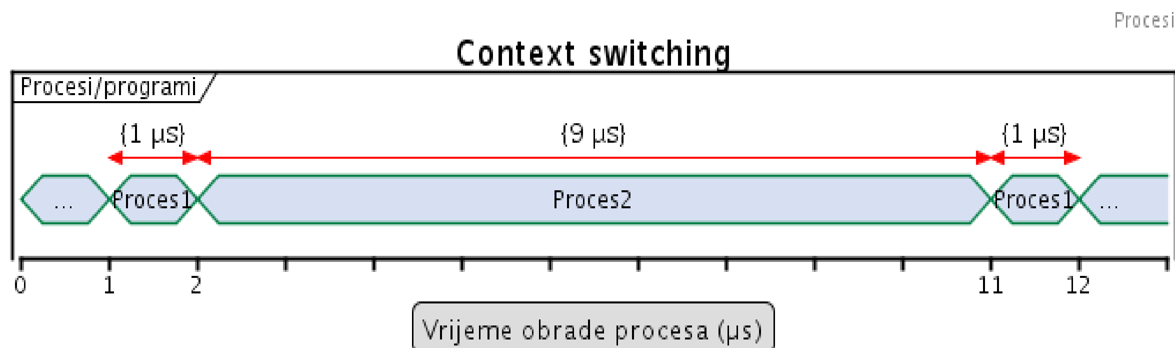
***Slijedi napredno poglavlje!***
Na primjeru centralnog procesora (***CPU***) s jednom jezgrom, možemo vidjeti kako je CPU u svakom trenutku odnosno najmanjoj jedinici vremena, u stanju obraditi sâmo jedan proces ili programsku nît unutar određenog procesa. Ono što se zapravo događa, prije sâmog trenutka pokretanja procesa je to da se prvo postavlja prioritet svakog pojedinog procesa.

Prioritet se postavlja od strane sustava ili naknadno od strane korisnika ili administratora. Ovisno o prioritetu, za svaki proces se računa koliko vremena, unutar jednog vremenskog intervala će CPU odrađivati taj proces. Kako bi se više programa istovremeno moglo pokretati, CPU radi preklapanje (engl. *switching*) ili takozvani ***context switching*** između obrade više procesa odnosno programa. Možemo sve ovo zamisliti, kao da svaki procesor (CPU) ima malu sklopku s kojom se prebacuje na obradu s jednog procesa na drugi proces (program).
Zamislimo pojednostavljeni primjer u kojem imamo samo dva pokrenuta programa (procesa):

- Prvi prioriteta 90%.
- Drugi prioriteta 10%.

Procesor odnosno procesorska jezgra (CPU) u svakom segmentu vremena (neka je to primjerice 10 µs) obrađuje: prvi proces 9µs, a drugi 1 µs. U sljedećem segmentu vremena, što je sljedećih 10 µs: ponovno će se prvi proces obrađivati 9µs, a drugi 1 µs i tako dalje, u krug, za sve procese odnosno programe pokrenute na sustavu.
Ovaj primjer pogledajte na vremenskom dijagramu na slici 27. dolje.

**Slika 27.** ***Context switch odnosno*** **prebacivanje obrade s procesa na proces**




Analogija ovome je vidljiva i u višenîtnim (engl. *multi threading*) programima. Oni se na razini operativnog sustava obrađuju isto kao zasebni procesi jer kada dođu do razine za obradu na *CPU*, ponovno se događa ista stvar. Dakle svaka nît (engl. *thread*) programa dobiva određeni prioritet i obrađuje se samo djelić vremena, pa se obrada prebacuje na drugu programsku nît i tako u krug, sve dok se ne odrade sve programske nîti, a potom sve ponovno iz početka, kako je vidljivo na slici **28.**

Mehanizam prebacivanja između više konteksta (izvršavanja više programa) možemo raščlaniti u nekoliko koraka:

1. **Pohrana trenutnog konteksta prije prebacivanja na drugi**: uključuje pohranu registara procesora, statusnih zastavica, svih drugih važnih podataka koji definiraju trenutno stanje procesa i sve privremene podatke procesa, u RAM memoriju.
2. **Učitavanje konteksta novog programa:** nakon što je stari kontekst pohranjen, sustav učitava kontekst novog programa koji će biti izvršen. To uključuje učitavanje podataka novog procesa iz RAM memorije u registre procesora.
3. **Ažuriranje tablica procesa:** sustav mora ažurirati razne tablice i strukture podataka koje se odnose na informacije o procesima ili njihovim nitima. To uključuje promjene u rasporedu izvođenja, dodavanje novog zadatka u popis aktivnih procesa, ažuriranje podataka o vremenu izvođenja, i drugo.
4. **Pokretanje novog konteksta (programa):** nakon što je novi kontekst odnosno program učitan (koraci **2.** i **3.**) i sve potrebne informacije su ažurirane, sustav izvršava (pokreće) novi program odnosno kontekst.
5. **Prebacivanje na novi kontekst ili nastavak izvršavanja:** ovaj korak završava samim izvršavanjem konteksta, ili može dovesti do novog ***context switchinga***, ako se dogodi neki važan događaj (pr. hardverski ili softverski signal prekida i sl.) ili se planira izvođenje drugog konteksta (programa) koji je u redu čekanja na izvršavanje.

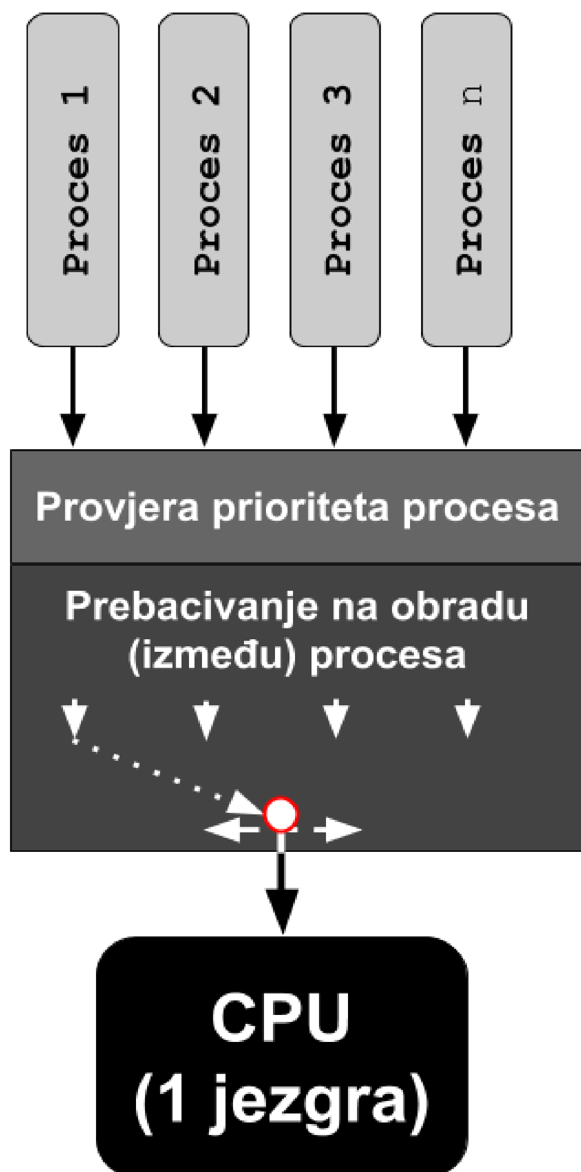


Za ograničenja procesa pomoću prioriteta, pogledajte nastavak ove cjeline, a za **context switching** mehanizam:
**9.3.1. Context switching.**
Za ograničenja procesa prema jezgrama procesora (CPU-a), pogledajte poglavlje:
**10.7.2.1.1. CPU afinitet (CPU Affinity)**.



Za *NUMA* arhitekturu procesora pogledajte poglavlje:
**10.7.2.2.2. NUMA CPU i RAM afinitet.**

Slika 28. Pogled na obradu procesa.




**Sumirajmo značajke procesa i njihovih prioriteta u radu:**

- Svaki pokrenuti program (proces) ima određeni prioritet.
- Standardni prioritet sistemskih programa je često nula (`0`) .
- Standardni prioritet korisničkih programa je `19` ili `20` (ovisno o sustavu):
    - Prioriteti se kreću od `-20` do (+)`20` (odnosno `19`).
    - Najveći prioritet je `-20`, a najmanji (+)`20` .

Moguće je mijenjati prioritet svakog procesa, u trenutku njegovog pokretanja ili naknadno, što se odnosi na razinu prioriteta za korisničke programe.

Promjene direktno utječu na parametar koji se zove ***nice***, a koji mehanizmu koji se zove *process scheduler* odnosno *task scheduler*, prosljeđuju zahtjev za promjenom *nice* vrijednosti određenog procesa.
Pojednostavljeno, promjena ***nice*** vrijednosti izvršavanja procesa zapravo utječe na vrijeme koliko dugo će se taj proces (program) izvršavati, sve dok se *task scheduler* ne prebaci na izvršavanje drugog procesa (programa).

Na najnižoj razini rada, iako to zvuči konfuzno, *nice* vrijednost se ne smije miješati sa stvarnim prioritetom koji koristi mehanizam za prebacivanje rada procesa (*task schedulera*), koji aplikacijama omogućuje da odrede redoslijed i vrijeme u kojem se proces (program) i njegove programske niti planiraju izvoditi.
Za razliku od stvarnog prioriteta, *nice* vrijednost je samo savjet *task scheduleru*, koji on eventualno može i zanemariti (iako to uglavnom nije slučaj). Terminološki, ***POSIX*** definira baratanje s *nice* vrijednosti u smislu njene vrijednosti.
Za detalje o odnosu *nice* vrijednosti i stvarnog (efektivnog) prioriteta koji se postavlja na procese, dobro proučite cjeline koje slijede.



Za primjere, kako se svaki proces inicijalizira u sustavu virtualne memorije, pogledajte poglavlje:
**12.4 Anatomija programa u memoriji** te poglavlje:
**12.3.3.4 Naredbe** ps **i** top **za memoriju.**

---

**Slijede primjeri**

**Postavljanje nice vrijednosti (*efektivnog prioriteta*) procesa upotrebom naredbe** `nice`
Pokrenimo proces, konkretno skriptu `test.sh` s *nice* vrijednosti uvećanom za **10**, što znači **+10** u odnosu na trenutnu vrijednost (što mu efektivno smanjuje vrijednost za **10**):

```
nice -n 10 test.sh
```

Isto možemo postići i na sljedeći način:

```
nice -10 test.sh
```

Naredba `nice` za svoju vrijednost, ako se ne koristi prekidač `-n` traži znak `-` ispred broja.
Ako se pak želi odabrati negativna vrijednost, tada treba dodati dva minusa: `--10`, pa bi to onda bilo:

```
nice --10 test.sh
```



***Ne zaboravite da sustav za korisničke programe efektivni prioritet (vidljiv kao `PR`) izračunava kao:***

**`PR` = 20 + `NI` vrijednost.**

**Promjene na već pokrenutim procesima**

Recimo da smo već pokrenuli skriptu ili proces naveden gore i da naknadno želimo povećati njegovu *nice* vrijednost za **5**, a njegov ***PID*** broj je `10456`. Prvo pogledajmo koju *nice* vrijednost skripta sada ima, pomoću naredbe `ps` i prekidača `-o` te dodatnih parametara koje želimo prikazati.

Prvo pogledajmo opis nekih od parametara naredbe `ps` kako slijedi, a vezano za prioritete i *nice* vrijednosti:

| Parametar | Opis parametra |
|---|---|
| `Pri` | Prikazuje prioritet procesa, kako se uobičajeno prikazuju (bez detalja na najnižoj razini). |
| `pri_baz` | Prikazuje efektivni (stvarni) prioritet procesa, na najnižoj razini. Pogledajte poglavlje: **9.3.** |
| `Nice` | Prikazuje ***nice*** vrijednost odnosno podatak je li se mijenjao efektivni prioritet procesa. |
| `Pid` | Prikazuje identifikacijski broj procesa *(Process ID).* |
| `ppid` | Prikazuje identifikacijski broj roditeljskog procesa (*Parent Process ID).* |
| `cmd` | Prikazuje izvršenu naredbu s kojom naredbom je proces (program) pokrenut. |

Sada ispišimo prioritet, ***nice*** faktor i ***PID*** broj za naš proces čiji ***PID*** broj je `10456`, sve pomoću naredbe `ps` na sljedeći način:

```
ps -o pri,nice,pid 10456
PRI   NI    PID
 19    0    10456
```

Vidimo da je njegov prioritet gotovo najniži i iznosi `19`, a `19` je obično standardan prioritet za korisničke procese (nekada je to i vrijednost `20`). Vidimo i kako se nije mijenjao *nice*, jer je sada nula (`0`), pa je efektivni prioritet: `PRI=19` (u našem slučaju).

**Promjena prioriteta već pokrenutih procesa, upotrebom naredbe `renice`:**

Sada promijenimo *nice* vrijednost našeg pokrenutog procesa, na način da ga želimo povećati za `5`:

```
renice 5 -p 10456
```

Provjerimo ponovno prioritet i *nice* vrijednost za naš proces (***PID*** `10456`), nakon ove promjene:

```
ps -o pri,nice,pid 10456
PRI   NI    PID
 14    5   10456
```

Sada vidimo da se efektivni prioritet sa `19` smanjio na `14`, što nam govori i *nice* koji je sada `5`, prema računici: `PRI=19-5=14`.

Ako pak sve želimo vratiti u izvorno stanje stavimo "*nice*" na nulu (`0`).

```
renice 0 -p 10456
```

Želimo li naknadno promijeniti prioritet svih procesa čiji je vlasnik korisnik imena: `pero` i svih korisnika čiji su vlasnici iz grupe; `korisnici`, u prioritet `+10` od izvornog, napravimo sljedeće:

```
renice 10 -u pero -g korisnici
```



Za dodatne optimizacije i detalje o efektivnim prioritetima i ***nice*** vrijednosti, pogledajte poglavlje:
**9.3. Task/process scheduler**, te posebno naredbu **`chrt`** i druge primjere.

Pogledajte i poglavlje:
**4.5.7. Datoteka *limits.conf* i druga ograničenja sustava** (dio konfiguracije s parametrima: **`NICE`** i **`PRIORITY`**).

**Izvori informacija: (1203),(K-4), (K-5), `man ps`, `man nice`, `man renice`.**

# 9.2. Procesi i signali koje im možemo poslati

Svakom procesu koji je pokrenut možemo poslati i određene signale, kojih postoji popriličan broj. Signali imaju svoju funkciju, ali i svoje ime te pripadajući broj. Dakle slanjem ovih signala, operativni sustav odnosno potencijalno i sâm korisnik, može poslati procesu signal na koji proces (program) mora odgovoriti. I sâm proces može slati signale; kako drugim procesima, tako i operativnom sustavu, kako bi im nešto signalizirao. Pa tako nekim signalima privremeno ili trajno zaustavljamo proces, s drugima ga možemo i nasilno ugasiti, a s nekima signalizirati određeno stanje procesa (programa).

Neke od ovih signala možemo trenutno aktivnom programu (procesu/skripti/naredbi) poslati i kombinacijom tipki. Tako primjerice kombinacija tipki: **CTRL C** trenutno pokrenutom programu šalje *SIGINT* signal odnosno signal za prekid rada programa.

Pogledajmo neke od najkorištenijih signala od strane korisnika:

- `SIGHUP` - (`1`) – ”*Hang UP*” obično se proces zaustavi ili se za servise oni restartaju i učitaju potencijalno promijenjenu konfiguraciju (engl. *re-read config*).
- `SIGINT` - (`2`) – signal za prekid rada procesa (engl. *Interrupt*).
- `SIGKILL` - (`9`) – nasilno zaustavlja proces (engl. *Kill*).
- `SIGTERM` - (`15`) – standardno zaustavlja proces (engl. *Terminate*).

Listu svih podržanih signala, od strane vašeg Linuxa možete vidjeti sa naredbom `kill` na sljedeći način:

```
kill -l
```

Pogledajmo tablicu signala podržanih na linux kernelu 2.6.x (i novijim):

| **Redni br. signala i oznaka** | **Redni br. signala i oznaka** | **Redni br. signala i oznaka** | **Redni br. signala i oznaka** | **Redni br. signala i oznaka** |
|---|---|---|---|---|
| (1) SIGHUP | (2) SIGINT | (3) SIGQUIT | (4) SIGILL | (5) SIGTRAP |
| (6) SIGABRT | (7) SIGBUS | (8) SIGFPE | (9) SIGKILL | (10) SIGUSR1 |
| (11) SIGSEGV | (12) SIGUSR2 | (13) SIGPIPE | (14) SIGALRM | (15) SIGTERM |
| (16) SIGSTKFLT | (17) SIGCHLD | (18) SIGCONT | (19) SIGSTOP | (20) SIGTSTP |
| (21) SIGTTIN | (22) SIGTTOU | (23) SIGURG | (24) SIGXCPU | (25) SIGXFSZ |
| (26) SIGVTALRM | (27) SIGPROF | (28) SIGWINCH | (29) SIGIO | (30) SIGPWR |
| (31) SIGSYS | (34) SIGRTMIN | (35) SIGRTMIN+1 | (36) SIGRTMIN+2 | (37) SIGRTMIN+3 |
| (38) SIGRTMIN+4 | (39) SIGRTMIN+5 | (40) SIGRTMIN+6 | (41) SIGRTMIN+7 | (42) SIGRTMIN+8 |
| (43) SIGRTMIN+9 | (44) SIGRTMIN+10 | (45) SIGRTMIN+11 | (46) SIGRTMIN+12 | (47) SIGRTMIN+13 |
| (48) SIGRTMIN+14 | (49) SIGRTMIN+15 | (50) SIGRTMAX-14 | (51) SIGRTMAX-13 | (52) SIGRTMAX-12 |
| (53) SIGRTMAX-11 | (54) SIGRTMAX-10 | (55) SIGRTMAX-9 | (56) SIGRTMAX-8 | (57) SIGRTMAX-7 |
| (58) SIGRTMAX-6 | (59) SIGRTMAX-5 | (60) SIGRTMAX-4 | (61) SIGRTMAX-3 | (62) SIGRTMAX-2 |
| (63) SIGRTMAX-1 | (64) SIGRTMAX | | | |

Nakon što se logirate na Linux sustav te pokrenete neku naredbu unutar ljuske (*shella*), ta naredba odnosno program će se izvršavati, sve dok ne izađete iz te ljuske: namjerno ili, ako vam se veza na udaljenu ljusku poput veze preko ***ssh*** protokola ne prekine. Naime ono što se događa kod odlogiranja sa sustava, je da se svim procesima koje ste pokrenuli šalje signal: `SIGHUP` (`1`) i oni se svi gase.

***Trajni rad programa (Nohup rad) bez prekida u radu***
Ako imate potrebu da se neki proces odnosno naredba/program, nastavi izvršavati i nakon što se odlogirate, to se može postići na sljedeći način. Recimo da se radi o našoj skripti imena: `vazno_za_rad.sh`.
Tada ju upotrebom naredbe `nohup` možemo pokrenuti na sljedeći način:

```
nohup vazno_za_rad.sh &
```

Na ovaj način smo pokrenuli našu skriptu u pozadini (*background)* sa znakom `&` na kraju, a zbog pokretanja s `nohup` otporna je na odlogiranje ili prekid veze s ljuskom odnosno *shellom*.
Pogledajmo i opis nekih drugih signala, uz čije razumijevanje ćete bolje shvatiti čemu sve služe.

Nabrojat ćemo ih samo nekoliko:

- `SIGABRT` i `SIGIOT` – ovi se signali mogu slati procesu, kako bi ga se obavijestilo da prekine s radom (*abort* ili *terminate).* Ovaj signal, proces obično šalje sâm sebi kada poziva funkciju ***abort()*** u *C* jeziku, ali se može slati i drugom procesu.
- `SIGALRM`, `SIGVTALRM` i `SIGPROF` – ovi signali se šalju procesu kada im istekne vrijeme potrebno za određeni poziv odnosno kada im istekne vrijeme za neki *timer* ili brojač. `SIGALRM` se šalje kada istekne neki brojač. `SIGVTALRM` – šalje se kada istekne CPU vrijeme za ovaj proces. `SIGPROF` se šalje kada je CPU vrijeme za proces ili sustav, iniciran od strane procesa, istekao.
- `SIGBUS` – ovaj signal se šalje procesu, kada je on izazvao neku grešku na sabirnici (engl. *Bus*) ili u pristupu kroz sabirnicu, poput slučaja u kojem proces traži pristup nepostojećoj memorijskoj adresi.
- `SIGCHLD` – signal se šalje procesu kada je pôd proces (proces dijete) zaustavljen, pauziran ili prekinut u radu (engl. *interrupted*). Jedna od upotreba je i u slučajevima kada se nalaže operativnom sustavu, kada je pôd proces završio, da oslobodi njegove resurse, ako se on nije uredno ugasio.
- `SIGCONT` – je signal kojim se daje instrukcija operativnom sustavu da proces koji je pauziran, pomoću signala: `SIGSTOP` ili `SIGTSTP`, da se pauzirani proces sada može nastaviti izvršavati.
- `SIGFPE` – je signal koji se šalje procesu kada pokuša raditi nemoguće aritmetičke operacije poput dijeljenja s nulom i slično.
- `SIGKILL` – je signal s kojim smo se upoznali, a on bezuvjetno ubija proces. Njega može pozvati i sustav, prema procesu koji pokušava odraditi operacije koje su: nelegalne, nepoznate ili izvan svojih ovlasti.
- `SIGPIPE` – ovaj signal se šalje procesu koji pokušava slati podatke na *Unix*/*Linux* cjevovod (***Pipe***) u slučaju kada nema drugog kraja cjevovoda.
- `SIGSYS` – ovaj signal se šalje procesu koji pokušava pozvati neki sistemski poziv s krivim argumentom ili kada proces pokušava raditi nedozvoljene radnje.

**Pogledajmo i primjere slanja signala procesima, od strane korisnika:**
Normalno ćemo zaustaviti skriptu, čiji ***PID*** broj je: `13245`. Zaustaviti ćemo ju sa signalom: `SIGTERM` (`15`), što je standardna metoda zaustavljanja (pomoću naredbe `kill`). Ova nenasilna metoda, nalaže programu (aplikaciji/skripti) da se uredno ugasi. To znači kako aplikacija uredno može snimiti sva svoja stanja, privremeno otvorene datoteke i slično, uz malu napomenu, kako aplikacije koje su u stanju u kojemu ne žele da ih se prekida, mogu ignorirati ovaj signal za zaustavljanje odnosno gašenje.

```
kill -15 13245
```

ili

```
kill 13245
```

Uredno stopirajmo istu skriptu, koja ima proces ***PID***: `13245`, ali ovaj puta sa `SIGHUP` (`1`) signalom.

```
kill -1 13245
```

Uredno znači normalno gašenje programa pomoću `SIGHUP` u kojem se programu nalaže normalno, nenasilno gašenje u kojemu program uredno može snimiti sve što treba, osloboditi privremene datoteke, osloboditi sve druge resurse i uredno se ugasiti. S time da aplikacije koje su u stanju u kojemu ne žele da ih se trenutno prekida, mogu ignorirati ovaj signal.

Ova metoda je specifična i po tome jer se automatski šalje aplikacijama koje se izvode na terminalu, kada se korisnik odspoji od tog terminala (kako bi se sve aplikacije odnosno procesi zaustavili jer korisnik više nije spojen na sustav).
Sada nasilno zaustavimo našu skriptu koja ima proces ***PID*** broj: `13245`, sa signalom `SIGKILL` (`9`):

```
kill -9 13245
```

Ova metoda na najnasilniji mogući način ubija proces, koji nema vremena snimiti svoja stanja, otvorene datoteke i slično.
Ova metoda se koristi kod procesa/programa koji ne odgovaraju na niti jedan drugi signal pa ih se mora na silu ugasiti.
Postoji i još jedna metoda, poput metode: `SIGKILL` (`9`), a to je metoda slanjem signala: `SIGQUIT` (`3`), koji radi isto što i navedeni signal (`9`), s time što dodatno, ako je to omogućeno na sustavu, snima i takozvanu *core dump* datoteku, za kasniju analizu (*debugiranje*) programa. Sada probajmo, ponovno *ubiti* naš proces, ali sa signalom: `SIGQUIT` (`3`):

```
kill -3 13245
```



Za više informacija o „*core dump*“ funkcionalnosti sustava, pogledajte poglavlje:
**16. Kernel Dump/Crashdump/core dump.**

I određene kombinacije tipki na tipkovnici zapravo šalju određene signale trenutno pokrenutim programima odnosno procesima:

- **CTRL C** - šalje signal za prekid: `SIGINT` (`2`), na osnovu kojeg se trenutno pokrenuti program mora ugasiti.
- **CTRL Z** - šalje signal: „*terminal stop*“: `SIGTSTP` (`20`), koji standardno pauzira trenutno pokrenuti program.
- **CTRL ** - šalje „*quit*“ signal: `SIGQUIT (3)`, zaustavlja trenutno pokrenuti program te snima „*dump core*“ datoteku.
- **CTRL T** - šalje „info“ signal: `SIGINFO`, koji (ako je podržan) od strane OS-a, traži informacije o pokrenutoj naredbi.
- **CTRL X** - šalje „interrupt“ signal: `SIGINT (2)`, isto ponašanje kao **CTRL C**.
- **CTRL S** - šalje „suspend“ signal: `SIGSTOP (19)`, koji zaustavlja program (suspend), koji se kasnije može aktivirati sa `SIGCONT(18)`.
- **CTRL Q** - šalje „continue“ signal: `SIGCONT (18)`, koji nastavlja izvršavanje programa pr. zaust. sa `SIGSTOP`.

## Naredba trap

Osim klasičnog slanja signala nekom procesu, moguće je korištenjem naredbe `trap`, uhvatiti željeni signal u svrhu aktiviranja naše željene naredbe ili skripte. Signale možemo promatrati kao asinkrone obavijesti koje se šalju na vašu skriptu ili program kada se dogode određeni događaji. Većina tih obavijesti odnosi se na događaje za koje se nadamo da se nikada neće dogoditi, poput neispravnog pristupa memoriji ili lošeg sistemskog poziva. Međutim, postoji nekoliko ovakvih „događaja" odnosno signala s kojima ćemo se pozabaviti.

Prvo ćemo hvatati signal `SIGINT`* (2) u našoj *bash* ljusci, i pri tome, ako ga netko pošalje, ispisati će se poruka „*PORUKA*".

```
trap "echo PORUKA" SIGINT
```

Pogledajmo hvatamo li u našoj *bash* ljusci signal `SIGINT`* (2) pomoću *trap* funkcionalnosti. Stoga pokrenimo `trap` ovako:

```
trap
trap -- 'echo PORUKA' SIGINT
trap -- '' SIGTSTP
trap -- '' SIGTTIN
trap -- '' SIGTTOU
```

Vidimo da ga* *hvatamo* i sada, ako stisnemo ***CTRL C***, dobit ćemo poruku „PORUKA".
Ako izađemo iz naše ljuske sve *trap* postavke koje smo definirali se brišu.



***Bash** ljuska koristi poseban pseudo signal imena: **EXIT**, koji se generira svaki puta kada neka **shell** skripta završi zbog neke greške u logici ili greške u nekoj od naredbi u nîzu, unutar sâme skripte.*

Drugi primjer upotrebe bi bio da napravimo skriptu koja će u slučaju kada je došlo do bilo kakve pogreške u skripti pokrenuti naredbu za brisanje log datoteke imena: `/tmp/output.txt`.

```
#!/bin/bash
trap "rm -f /tmp/output.txt" EXIT
yum -y update > /tmp/output.txt
```

U navedenoj skripti, u slučaju kada naredba s kojom se nadograđuje sustav (`yum -y update`) i koja sve poruke o nadogradnji zapisuje u datoteku: `/tmp/output.txt` pukne ili izbaci neku grešku, a potom izađe iz sâme skripte, nikada neće moći izaći iz skripte, a da se pritom ne pokrene naša ***trap*** zaštita koja će prvo obrisati datoteku: `/tmp/output.txt` (s naredbom `rm`) i tek onda završiti s radom skripte.

Ista aktivacija ***trap**-a* će se dogoditi i ako nasilno prekinemo izvršavanje ove skripte (sa ***CTRL C***).
Primjer skripte je preuzet iz izvora informacija: **(731).**

**Izvori informacija: (293),(727),(728),(729),(730),(731),(K-4),(K-5), `man trap`, `man 7 signal`, `man kill`, `man signal`,`man nohup`.**

---

# 9.2.1. Napredno o procesima i signalima

***Slijedi napredna cjelina!***
Kada *Linux Process* odnosno *Task Scheduler*, primijeti da postoji signal koji je potrebno proslijediti procesu, on daje procesoru (CPU) vrijeme odnosno vremenski okvir za obradu tog signala. Točnije dogodi se *Context Switch* na proces koji mora obraditi taj signal. Tada se signal dostavlja procesu (programu). Kada proces primi taj signal, on mora odgovoriti sustavu odnosno *Linux kernelu*, što da radi s njim. Kernel tada provjerava, za svaki pojedini proces, što će sa konkretnim signalom.

Pri tome signali za svaki proces mogu biti kategorizirani u tri skupine:

- **Signal može biti ignoriran (engl. *Ignored*) (*`SigIgn`*)**. Ovo može biti zbog razloga što konkretni proces nema metodu s kojom bi znao kako obraditi konkretni signal. Ako je signal ignoriran to znači kako se slanjem tog konkretnog signala procesu neće ništa dogoditi. Određeni signali se **NE MOGU** ignorirati, a to su signali: `SIGKILL` i `SIGSTOP` kao i posebni hardverski signali (engl. *Hardware Exceptions*).
- **Signal može biti "*uhvaćen*" (engl. *Caught*) (*`SigCgt`*)**. Tada proces registrira određenu funkciju (ovisno o konkretnom signalu) s Linux kernelom. Kada se taj signal poziva, tada Linux kernel poziva tu konkretnu funkciju. Ako signal nije `SIGKILL` ili `SIGSTOP` za proces, tada proces može odraditi što je traženo od strane signala i to preko funkcije s kojom je povezan s kernelom za taj signal.
- **Signal može biti blokiran (engl. *Blocked*) (*`SigBlk`*)** i tada neće nikada doći do konkretnog procesa.
  Ne mogu se blokirati signali `SIGKILL` i `SIGSTOP`.

Signali `SIGKILL` i `SIGSTOP` ne mogu nikako i nikada biti ignorirani ili blokirani.
To je stoga što ova dva signala daju mogućnost sustavu da ubije ili uredno zaustavi proces u bilo kojem trenutku.
Prema tome, gore navedena dva signala ne mogu biti u kategorijama "*Ignored*" ili "*Blocked*" već samo u kategoriji "*Caught*".
Dakle kategorija "*Caught*" je standardna kategorija koja normalno prihvaća i obrađuje točno navedene signale, za svaki proces.
Svaki proces ima definiranu *Signalnu masku* u kojoj je definirano, za svaki pojedini signal, u koju od gore navedenih kategorija pripada.
To znači kako će neki signali za pojedini proces moći biti ignorirani (*SigIgn*), dok će drugi biti uhvaćeni i obrađeni (*SigCgt*), a treći će biti blokirani (*SigBlk*). Pod Linuxom za svaki pokrenuti proces možemo iščitati signalnu bît masku.
*Signalna Bît maska* se nalazi unutar posebnog direktorija `/proc/`**`PID`**`/status` s time da ***PID*** označava ***PID*** (*Process* ID) broj pokrenutog procesa. Navedena `status` datoteka sadrži brojne podatke o (svakom) pojedinom procesu (kako smo rekli - prema njegovom ***PID***-u).

Nas trenutno zanimaju samo tri retka iz ove datoteke:

- *SigBlk* – za blokirane signale.
- *SigIgn* – za ignorirane signale.
- *SigCgt* – za signale koji se odrađuju.

Pogledajmo primjerice što je vidljivo za proces s ***PID*** brojem `1`, a to je prvi Linux proces `init` koji pokreće sve procese na Linuxu. Ispis je skraćen odnosno filtriran samo za ova tri reda koja su nam zanimljiva:

```
cat /proc/1/status | grep ^Sig | egrep "Cgt|Blk|Ign"

SigBlk: 0000000000000000
SigIgn: 0000000000001000
SigCgt: 00000001a0016623
```

Sve vrijednosti su heksadecimalne, pa ih prvo moramo pretvoriti u binarne.

Zanimaju nas svi signali koji se normalno obrađuju (*SigCgt*) stoga ih pogledajmo:
**`HEX: 00000001a0016623`** u binarnom je to: **`000110100000000000010110011000100011`**

Pošto smo rekli kako je to *bît maska*, a to znači, kako svaki bit predstavlja određenu poziciju odnosno brojčanu oznaku signala.

Pogledajte ovu bît masku za naš primjer:

```
000110100000000000010110011000100011
   || |            | ||  ||   |   ||
   || |            | ||  ||   |   ||
   || |            | ||  ||   |   | \
   || |            | ||  ||   |   |  \--> (1) SIGHUP
   || |            | ||  ||   |    \
   || |            | ||  ||   |     \--> (2) SIGINT
   || |            | ||  ||    \
   || |            | ||  ||     \--> (6) SIGABRT
   || |            | ||  | \
   || |            | ||  |  \--> (10) SIGUSR1
   || |            | ||   \
   || |            | ||    \--> (11) SIGSEGV
   || |            | | \
   || |            | |  \--> (14) SIGALRM
   || |            |  \--> (15) SIGTERM
   || |             \--> (17) SIGCHLD
   ||  \--> (30) SIGPWR
   | \
   |  \--> (32) Nije definiran (u ovoj inačici kernela)
    \
     \--> (33) Nije definiran (u ovoj inačici kernela)
```

Dakle naš pokrenuti proces s ***PID*** brojem**: `1`,** odnosno proces: `init`, odgovara samo na slijedeće signale:

- (**1**) SIGHUP, (**2**) SIGINT i (**6**) SIGABRT
- (**10**) SIGUSR1 i (**11**) SIGSEGV
- (**14**) SIGALRM i (**15**) SIGTERM
- (**17**) SIGCHLD, (**30**) SIGPWR, ali i signale koji nisu definirani u trenutnoj inačici kernela: (**32**) i (**33**).

**Izvori informacija: (111),(218),(672),(1534),(1535)**, **`man 5 proc`**,**`man 2 syscalls`**,**`man 7 signal`**.

## 9.2.2. Signali i sistemski pozivi

U Linuxu, signali su zahtjevi ili posebni prekidi koji se šalju programu da ga upozore da se dogodio važan događaj. Događaji mogu varirati od korisničkih zahtjeva, preko sistemskih zahtjeva, sve do grešaka. Signali se ponekad opisuju i kao softverski prekidi, jer u većini slučajeva prekidaju normalan tijek izvođenja programa, a njihova aktivacija je uglavnom nepredvidiva. S druge strane *sistemski pozivi* (engl *syscall*) su temeljno sučelje između aplikacija i Linux kernela.

*Sistemski poziv* je zahtjev programa (koji se izvršava u korisničkom prostoru) prema kernelu odnosno jezgri operativnog sustava. Naime, jezgra (kernel) operativnog sustava pruža mnoge usluge prema korisničkim programima s jedne strane, a s druge strane prema svim komponentama operativnog sustava i s hardverom. Primjerice kada vaš program želi pisati ili čitati iz datoteke, osluškivati ili ostvariti mrežnu vezu, izbrisati ili stvoriti direktorij ili čak završiti svoj rad, program u konačnici za te zadatke koristi *sistemske pozive*.

To možemo promatrati i na sljedeći način. Kernel pruža skup sučelja pomoću kojih procesi koji se izvode u korisničkom prostoru mogu komunicirati sa sustavom. Ova sučelja aplikacijama daju kontrolirani pristup hardveru, mehanizmu za pokretanje novih programa (stvaranje novih procesa) i komunikaciju s postojećima te mogućnost traženja drugih resursa od operativnog sustava.



Za više informacija o sistemskim pozivima pogledajte i poglavlje:
**26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije.**

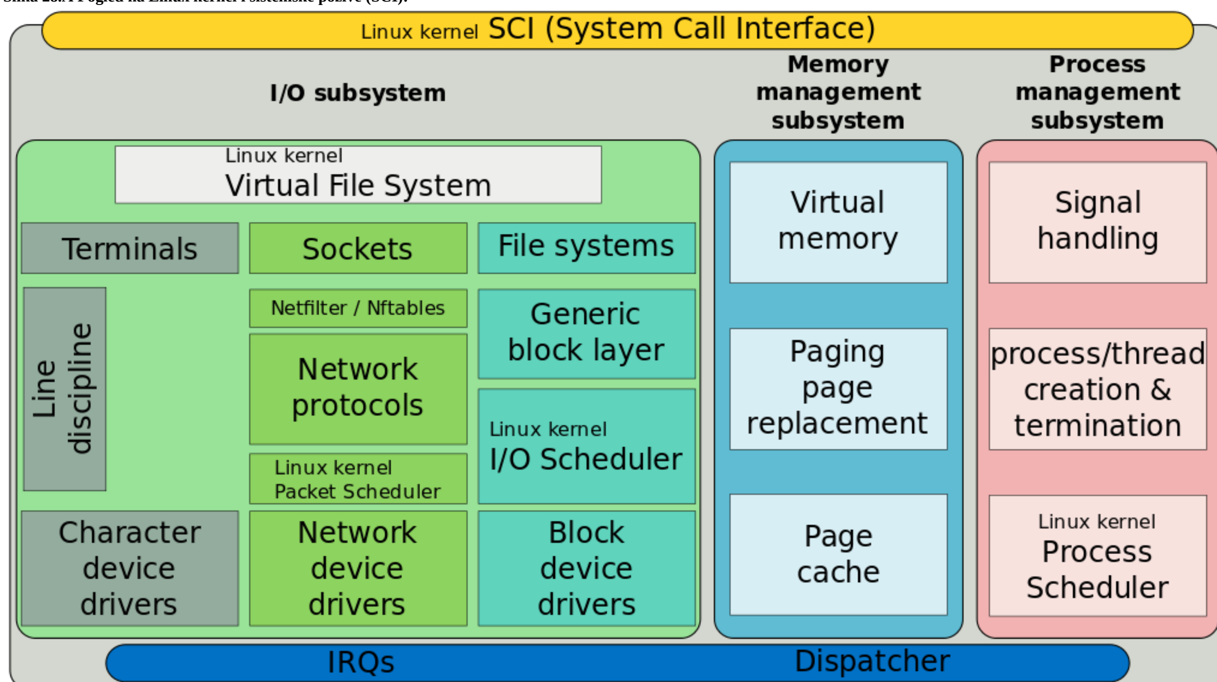
Ova programska sučelja odnosno takozvani ***API-ji*** (engl. *Application Programming Interfaces*) djeluju kao posrednici između aplikacija i kernela. Pri tome aplikacije izdaju različite zahtjeve, a kernel ih ispunjava (izvršava) ili vraća pogrešku. Međutim sistemski pozivi i API-ji nisu ista stvar. API-ji su u osnovi definicije funkcija koje određuju kako dobiti određenu uslugu (ili servis) od operativnog sustava. Sistemski pozivi (zvani i **SCI**) općenito se ne pozivaju izravno, već putem funkcija, obično u ***glibc*** ili eventualno nekoj drugoj biblioteci.

Dakle sistemski pozivi su funkcije kernel prostora koju programi korisničkog prostora pozivaju da se obradi neki zahtjev prema sustavu. Linux kernel pruža skup ovih funkcija, a svaka arhitektura računala odnosno procesora (pr. x86,x86-64, ARM, RISC-V, ...) nudi svoj skup funkcija.



Primjerice ***x86_64*** arhitektura pruža **322** sistemska poziva, a ***x86*** arhitektura pruža **358** različitih sistemskih poziva.

**Slika 28.A Pogled na Linux kernel i sistemske pozive (SCI):**




**Autor ilustracije: ScotXW (Wikipedia, licenca: CC BY SA)**

## Razine privilegija

Arhitektura većine modernih procesora, s izuzetkom nekih specijaliziranih sustava, uključuje sigurnosni model u komunikaciji između aplikacija i jezgre operativnog sustava to jest kernela i u konačnici hardvera.
Na primjer, najčešće korišteni prstenasti model navodi višestruke razine privilegija pod kojima se softver (aplikacija/program) može izvršavati: program je obično ograničen na vlastiti (izolirani) adresni prostor tako da ne može pristupiti ili modificirati druge pokrenute programe ili sam operativni sustav i obično mu je onemogućeno izravno manipuliranje hardverom odnosno uređajima ili strukturama na nižoj softverskoj razini poput memorijskih međuspremnika sustava (*buffers*, *cache*, …).

Međutim, mnoge aplikacije trebaju pristup tim komponentama, tako da operativni sustav preko sistemskih poziva pruža dobro definiranu (sigurnu) implementaciju za takve operacije. Operativni sustav izvršava se na najvišoj razini privilegija i omogućuje aplikacijama da zatraže usluge putem sistemskih poziva, koji se često aktiviraju signalima prekida (engl. *Interrupts*).

Sam signal prekida automatski postavlja CPU na povišenu razinu privilegija i zatim prosljeđuje kontrolu kernelu, koji određuje hoće li pozivnom programu biti odobrena tražena usluga. Ako je usluga odobrena, kernel izvršava određeni skup instrukcija nad kojima pozivni program nema izravnu kontrolu, te vraća razinu privilegija na onu pozivajućeg programa, a zatim vraća kontrolu pozivajućem programu.

## Zaštitni slojevi (prstenovi)

U računalstvu, hijerarhijske zaštitne domene, često zvane zaštitni prstenovi, mehanizmi su za zaštitu podataka i funkcionalnosti sustava od grešaka i zlonamjernog ponašanja. Pri tome operativni sustavi omogućuju različite razine privilegija pristupa resursima sustava. Sustav takozvanih zaštitnih prstenova (slojeva) obično čine dvije ili više hijerarhijskih razina ili slojeva privilegija unutar arhitekture računalnog sustava.
To je hardverski definirano arhitekturama procesora koje pružaju različite načine rada procesora na razini hardvera ili mikrokoda. Prstenovi su raspoređeni u hijerarhiji od najpovlaštenijih (najpouzdaniji, obično označeni brojem nula) do najmanje povlaštenih (najmanje pouzdani, obično s najvećim brojem prstena). Na većini operativnih sustava, prsten 0 (engl. *ring 0*) je razina s najvišim privilegijama i najizravnije komunicira s fizičkim hardverom odnosno procesorom, memorijom, ulazno-izlaznim uređajima i slično.



Za više informacija o kernelu i njegovoj arhitekturi pogledajte poglavlje:
**11.1. Kernel.**

Za više primjera o upotrebi sistemskih poziva i signala, pogledajte poglavlja:
**9.3.1. Context switching**.
**9.3.1.1. Još detaljniji rad procesa i nîti.**
**9.4. Komunikacija između procesa.**
**9.4.3.1. Unix socketi.**
**9.6. Procesne grupe.**
**10.4.4.2. API i ABI.**
**10.7.1.3. CPU registri još detaljnije.**
**11. Od kojih komponenti se sastoji Linux.**
**11.1.1. Rad s kernelom.**
**11.1.1. Rad s kernelom.**
**11.1.1.2. Napredno: Rad s kernel modulima.**
**11.2.3.3. Koje dijeljene biblioteke su nam potrebne za koji program.**

## Sumirajmo - za što se sve koriste sistemski pozivi

Sistemski pozivi se koriste za:

- **Upravljanje datotekama i direktorijima**: kreiranje, otvaranje/zatvaranje, čitanje/pisanje, brisanje, promjene dozvola i vlasništva, … .
- **Upravljanje procesima**: kreiranje (pokretanje) i završavanje procesa, čekanje na završetak podređenih procesa, promjena prioriteta procesa , dohvat informacija o procesima, …
- **Upravljanje memorijom**: alokacija/dealokacija, zaštita, sinkronizacija memorije, …
- **Međuprocesnu komunikaciju**: rad sa signalima, pipe i FIFO datoteke, socketi, dijeljena memorija, semafori , …
- **Rukovanje datotečnim sustavom**: montiranje/demontiranje datotečnih sustava, sinkronizacija podataka, dohvat informacija o datotečnom sustavu, …
- **Upravljanje vremenom i datumom**: dohvat trenutnog vremena, postavka vremena, *sleep* funkcije.
- **Kontrola pristupa i sigurnost**: upravljanje korisničkim ID-jevima (UID, GID), autentikacija i autorizacija, zaštita datoteka i direktorija, rad sa sigurnosnim oznakama, ….
- **Mrežna komunikacija**: stvaranje i upravljanje mrežnim sučeljima, čitanje i slanje podataka preko mreže, upravjanje mrežnim protokolima i njihovim opcijama, …
- **Upravljanje uređajima**: pristup hardveru i I/O operacijama, …
- **Upravljanje greškama**: dohvat kodova grešaka, rukovanje iznimkama i signalima.
- **Upravljanje resursima sustava**: ograničavanje resursa, zauzimanje/iskorištavanje procesorskog vremena, …
- . . .

**Izvori informacija: (111),(218),(672),(1534),(1535)**, `man 5 proc`,`man 2 syscalls`,`man 2 syscall`, `man 7 signal`.

# 9.3. Task/process scheduler

***Slijedi napredno poglavlje (9.3.x)!***
*Raspodjeljivač (planer) procesa tj. Task scheduler* također poznat i kao *process scheduler* je mehanizam zadužen za obradu procesa odnosno pokrenutih programa. Zbog činjenice kako se na vrlo niskoj razini, na jednoj jezgri centralnog procesora (CPU) u određenom vremenskom okviru, može obraditi sâmo jedan proces (program), ovaj mehanizam je vrlo važan kako za performanse, ali u osnovi i za rad cijelog operativnog sustava. Dodatna komplikacija je u tome, što se neki od procesa privremeno mogu zaustaviti, dok se drugi trebaju nastaviti obrađivati, a potom se privremeno zaustavljeni procesi mogu ponovno pokrenuti i nastaviti gdje su stali s radom. U svakom slučaju cijela *višezadaćnost* odnosno *multitasking* ovise o ovom mehanizmu.
U normalnom radu kada *task scheduler* radi *context switching* odnosno završava s obradom prvog procesa jer mu je istekao vremenski okvir u kojem se smije izvršavati, proces odlazi u stanje u kojem je privremeno zaustavljen od strane *task schedulera*. Ovo stanje procesa kada je zaustavljen od *task schedulera*, zbog prebacivanja na obradu drugog procesa se naziva ***preemption*** *stanje*. Zbog ovog naziva se i Linux mehanizam *task schedulera* zove „***Preemptive multitasking***" mehanizam odnosno preventivni višezadaćni rad. Vremenski period u kojem se proces može izvršavati je definiran za svaki proces unaprijed, na osnovi prioriteta koji mu je određen. Zapravo se u pozadini primjenjuje određeni algoritam koji izračunava navedene vremenske okvire izvršavanja za svaki proces. Isti mehanizmi koji se primjenjuju na procese, primjenjuju se i na sve programske nîti (engl. *Threads*) unutar svakog pojedinog procesa. Na još nižoj razini, radi se klasifikacija procesa prema nekoliko glavnih kategorija:

- ***I/O procesi***: klasifikacija se radi za procese koji komuniciraju sa ulazno/izlaznim (***I/O)*** podsustavom: disk, CD/DVD, tipkovnica, miš, … . Ovakvi procesi najviše vremena šalju ili primaju zahtjeve; obično za čitanje ili pisanje. Pokreću se vrlo često, ali u vrlo kratkim vremenskim intervalima. Dakle u pravilu se razlikuju od drugih vrsta procesa.
- ***Procesorski zahtjevni procesi***: oni koriste svoje vremenske okvire za izvršavanje programskog kôda i cilj im je da rade što duže unutar svog vremenskog okvira, sve do trenutka kada postanu privremeno zaustavljeni odnosno u stanju "*preemptive*" od strane *task schedulera*.

Postoji i dodatna kategorizacija:

- ***Interaktivni procesi***: su vrste procesa koji dosta vremena troše čekajući na ***I/O*** zahtjeve poput unosa s tipkovnice ili miša. *Scheduler* mora biti u stanju vrlo brzo odgovoriti na zahtjev prema ovim procesima. Prosječno dozvoljeno kašnjenje je do oko 100 ms, a da korisniku to ne postane usporeno odnosno vidljivo sporog odziva (engl. *Non responsive*) odnosno *ne responzivno*.
- ***Batch procesi***: ovi procesi se pokreću u pozadini te ne moraju biti *responzivni* kao interaktivni procesi. Oni su u načelu nižeg prioriteta. Primjeri su analizatori log datoteka, programi za konverziju multimedijalnih podataka i sl.
- ***Real-time procesi***: ovi procesi ne smiju biti prekinuti od strane procesa nižeg (manjeg) prioriteta i *task scheduler* mora garantirati najbrža vremena odziva za ove procese.

## Raspodjeljivač (planer) procesa *(engl. task/process scheduler)*

Do linux kernela 2.6.23 se koristio tzv. „***O" Scheduler*** to jest *planer/raspodjeljivač* procesa. Od linux kernela 2.6.23 koristi se tzv. **CFS** planer (engl. *Completely Fair Scheduler*). **CFS** je podijeljen u nekoliko dijelova odnosno logičkih cjelina:

- ***Modular Scheduler Core***: sastoji se od dodatnih "*scheduling*" klasa. Klase predstavljaju različite metode raspoređivanja procesa (engl. *Scheduling Policy*) koje nazivamo i politikama raspodjele rada procesa.
- ***Completely Fair Scheduler***: je sâma jezgra *scheduler-a* i brine se o tome kako bi svaki proces uvijek dobio pošteni (engl. *fair*) segment procesorskog vremena.
- ***Group scheduling***: zadužen je za grupiranje procesa u grupe kako bi se *scheduling* mogao odrađivati, uvažavajući parametre za svaku grupu. Grupe mogu biti osnovane prema korisničkim računima (***UID***) ili prema specifičnim internim grupama kernela.

***Važno je razumjeti da je rekonfiguracijom sustava moguće koristiti i neki drugi scheduler, primjerice: BFS ili noviji MuQSS.***

Za svaki proces se u početku određuje politika raspodjele rada (engl. *Scheduling Policy)* na osnovu koje se određuju parametri za stavljanje u nîz za obradu (engl. *Run queue*). Linux podržava nekoliko politika raspodjele rada procesa:

- `SCHED_FIFO` – koristi se za vremenski kritične procese/aplikacije. Koristi se *First In-First Out* scheduling algoritam (za ulazno/izlazne (***I/O***) procese).
- `SCHED_BATCH` – koristi se za CPU intenzivne procese/aplikacije (za ***Batch*** procese).
- `SCHED_IDLE` – koristi se za procese vrlo niskog prioriteta.
- `SCHED_OTHER` – koristi se za većinu Linux procesa (on je negdje između ostalih).
- `SCHED_RR` – sličan kao `SCHED_FIFO`, ali koristi ***Round Robin*** scheduling algoritam.
- `SCHED_DEADLINE` – za aplikacije u stvarnom vremenu [pr. audio/video pozivi] (dostupan tek od kernela 3.14.x).

Naime kada raspodjeljivač/planer procesa (*proces scheduler*) za pojedini proces odabere politiku raspodjele rada procesa i proces uđe u nîz za izvršavanje (*run queue*) unutar *task schedulera* te dođe na obradu, *scheduler* pamti trenutno vrijeme kada je proces krenuo prema CPU-u na obradu. Tada se proces šalje na CPU i obrađuje se. *Task scheduler* odbrojava vrijeme koje je izračunato, koliko se proces smije izvršavati i to na osnovi svih parametara pomoću kojih je izračunat vremenski okvir za rad konkretnog procesa. Kada se vrijeme za izvršavanje približi kraju, proces se pauzira (stavlja se u *preemptive* stanje) te se drugi proces, koji je sljedeći stavljen u red na obradu, šalje na CPU. Važno je razumjeti da svaki raspodjeljivač/planer procesa (*proces scheduler*) [pr. **CFS**], ali i svaka politika raspodjele (*Scheduling Policy*) imaju svoj algoritam za određivanje načina obrade procesa.
Također je važno razumjeti da globalno možemo imati odabran samo jedan raspodjeljivač/planer procesa (*proces scheduler*), a on na nižoj razini, čak i za svaki proces, može koristiti drugu politiku raspodjele rada to jest takozvani *scheduling policy*.



Važno je razumjeti da statistika prioriteta procesa; primjerice kod naredbe: top ono što vidimo pod stupcem PR zapravo predstavlja efektivni (stvarno) primijenjeni prioritet procesa koji sustav izračunava tako da standardnoj vrijednosti prioriteta **20** nadodaje ***nice*** vrijednost. Odnosno: PR = 20 + NI (u slučaju ako je standardan prioritet 20 [a nekada je i 19]). Međutim sustav na najnižoj razini radi s prioritetima od **1** do **139**. Vrijednosti od **1** do **99** su rezervirane za „*Realtime*" procese na razini kernela. Potom slijede vrijednosti za korisničke programe tj. one koje normalno pokrećemo u radu. Navedene vrijednosti nazivamo ***nice*** vrijednosti, a one mogu biti od **-20** za najveći prioritet do (+)**19** za najmanji, pri čemu je standardna vrijednost **20**. Nama vidljive vrijednosti se zapravo propagiraju kao **100** za najveći prioritet, sve do **139** za najmanji prioritet.

**Slijede primjeri**

Provjerimo koja *politika raspodjele* je primijenjena na naš proces *SSHD* (*SSH* servis). Za ovu namjenu koristimo naredbu chrt sa prekidačem -p nakon kojega slijedi ***PID*** broj procesa. Naš *SSH* servis im ***PID*** (provjerili smo sa: pidof sshd): 1995

```
chrt -p 1995

pid 1995's current scheduling policy: SCHED_OTHER
pid 1995's current scheduling priority: 0
```

Vidimo da je odabran: SCHED_OTHER. Također vidimo kako je dodatno odabran i prioritet 0.

Neke *politike raspodjele* imaju mogućnost odabira i dodatnog prioriteta, koji se koristi za napredno optimiziranje.

Pogledajmo u kojim okvirima je moguće optimizirati prioritete unutar svake *politike raspodjele*. Provjerimo to sa chrt:

```
chrt -m

SCHED_OTHER min/max priority    : 0/0
SCHED_FIFO min/max priority     : 1/99
SCHED_RR min/max priority       : 1/99
SCHED_BATCH min/max priority    : 0/0
SCHED_IDLE min/max priority     : 0/0
```

Vidljivo je da je optimizacija prioriteta za naš sustav, a ovisno o inačici kernela moguća samo za: SCHED_FIFO i SCHED_RR i to od **1** (najveći prioritet) do **99** (najmanji prioritet). Moguće je i ručno promijeniti *politiku raspodjele*, za određeni proces, ali ovdje treba biti oprezan.

Za promjenu *politike raspodjele*, naredba je ista, sâmo što je potrebno dodati prekidač koji označava novi algoritam:

- -b – odabir **SCHED_BATCH**, u statistici vidljivo kao: **B**, za uporabu proučite naredbu chrt.
- -f – odabir **SCHED_FIFO,** u statistici vidljivo kao: **FF**, za uporabu proučite naredbu chrt.
- -i – odabir **SCHED_IDLE**, u statistici vidljivo kao: **IDL**, za uporabu proučite naredbu chrt.
- -o – odabir **SCHED_OTHER**, u statistici vidljivo kao: **TS**, za uporabu proučite naredbu chrt.
- -r – odabir **SCHED_RR**, u statistici vidljivo kao: **RR**, za uporabu proučite naredbu chrt.
- -d – odabir **SCHED_DEADLINE**, dobar za aplikacije u stvarnom vremenu, za uporabu proučite naredbu chrt.

Dakle za naš slučaj, ako želimo odabrati "**SCHED_BATCH**" algoritam, to ćemo napraviti ovako (za proces s ***PID*** brojem 1995):

```
chrt -b -p 1995
```

Ove vrijednosti nije dobro mijenjati osim ako niste stvarno sigurni što radite.

Sada probajmo promijeniti *politiku raspodjele* u ***RR***, a pri tome pokrenuti novi proces koji će imati vrlo nizak broj prioriteta (odnosno broj koji upada unutar *Realtime* prioriteta*: 1-99*) i to za novi program odnosno skriptu imena: skripta.sh.

Pri tome ćemo toj skripti tijekom pokretanja, broj prioriteta smanjiti za *40*, što znači da će se prioritet povećati i postati „***Reatime***".

To ćemo napraviti na sljedeći način:

```
chrt -rr 40 skripta.sh
```

Ako pomoću naredbe top pogledamo PR (*priority*) stupac, vidjet ćemo nešto poput:

```
  PID USER     PR    NI    VIRT     RES     SHR S  %CPU %MEM    TIME+     COMMAND
50488 root    -41    0  183676    5364    3576 S   0.0  0.0   0:00.02  skripta.sh
```

Vidimo da je prioritet -41 jer se on ovdje izračunava kao: *PR= -1 - novi REALTIME prio* dakle PR= -1 -40 = -41.

I drugi programi uglavnom koriste ovakvu brojčanu oznaku.

Pogledajmo unutar statistike samog programa/skripte na razini kernela, na slijedeći način (***PID*** ovog programa je 50488):

```
cat /proc/50488/sched | grep prio
prio                                 :                    59
```

Vidimo da je efektivna vrijednost prioriteta; kako ga vidi kernel, broj 59; *Stvarni PRIO= standardni (19) + novi (40) = 59*.

Pošto je ovo broj unutar granice 1 do 99, jasno je da se radi o „***Realtime***" prioritetu odnosno onome vrlo visokog prioriteta.

Sada ćemo s naredbom ps prikazati i algoritam i prioritet naše skripte (50488) za koju smo promijenili i prioritet i algoritam:

```
ps -o pri_baz,nice,pid,class  50488

BAZ  NI   PID  CLS
 59   -  50488 RR
```

U ispisu smo koristili parametar pri_baz (stupac BAZ) pa smo dobili ispis efektivnog prioriteta odnosno prioriteta kako ga vidi kernel, potom vidimo algoritam (stupac class) koji se koristi; konkretno: **RR** (**SCHED_RR**), što se vidi u stupcu: CLS za naš proces (skriptu). Umjesto pri_baz, mogli smo koristiti i druge načine prikaza prioriteta, poput: priority (prikazao bi 41), pri (prikazao bi 80), ali i mnogih drugih*. Ako pak želimo detaljnije informacije o *task scheduler*-u, za cijeli operativni sustav, kao i statuse za svaki pojedini proces ugrubo, možemo pogledati sadržaj datoteke: /proc/sched_debug.

Za **CFS** pogledajmo ovu statistiku, čiji ispis smo skratili. Ovu datoteku možete vidjeti samo, ako vam je kernel kompiliran s opcijom: `CONFIG_SCHED_DEBUG=y` i pripadajućim opcijama: `CONFIG_DEBUG_KERNEL`, `CONFIG_PROC_FS` i drugima.

```
cat /proc/sched_debug

now at 18401716.231694 msecs
  .jiffies                                   : 4313069012
  .sysctl_sched_latency                      : 10.000000
  .sysctl_sched_min_granularity              : 2.000000
  .sysctl_sched_wakeup_granularity           : 2.000000
  .sysctl_sched_child_runs_first             : 0.000000
  .sysctl_sched_features                     : 3183
  .sysctl_sched_tunable_scaling              : 1 (logaritmic)
```

Detalje je moguće vidjeti i na razini svakog procesa, prema njegovom ***PID*** broju (jer pojedini procesi mogu u posebnim slučajevima koristiti različite algoritme, kako svo i vidjeli). Statistike na razini svakog pojedinog procesa se nalaze u datoteci: `/proc/PID/sched`, pri čemu je `PID` direktorij ***PID*** broj konkretnog procesa.

Podatke o tome koji je mehanizam politike raspodjele (*Scheduling Policy*), odabran za koji proces, dobivamo na sljedeći način:

```
ps -cTe

    PID    SPID CLS PRI TTY          TIME CMD
      1       1 TS   19 ?        00:00:00 systemd
      2       2 TS   19 ?        00:00:00 kthreadd
     37      37 FF  139 ?        00:00:00 watchdogd
```

U stupcu `CLS` za svaki proces je vidljivo s kojim je mehanizmom politike raspodjele (*Scheduling Policy*) proces i pokrenut.

Tako bi primjerice za naš proces iz gornjeg primjera (s ***PID*** brojem: `50488`), bilo sljedeće (i ovdje smo skratili ispis):

```
cat /proc/50488/sched

skripta.sh (50488, #threads: 1)
-------------------------------------------------------------------
se.exec_start                                :       1025020.040554
se.vruntime                                  :          6306.910656
se.sum_exec_runtime                          :             1.271729
se.sleep_start                               :       1025020.040554
se.sleep_max                                 :        954878.791664
se.exec_max                                  :             0.314730
se.slice_max                                 :             0.381025
se.wait_max                                  :             0.159499
se.wait_count                                :                    7
se.iowait_sum                                :           924.802882
se.iowait_count                              :                  103
avg_atom                                     :             0.181675
avg_per_cpu                                  :             1.271729
nr_switches                                  :                    7
nr_voluntary_switches                        :                    5
nr_involuntary_switches                      :                    2
se.load.weight                               :                 1024
policy                                       :                    0
prio                                         :                  120
clock-delta                                  :                   79
```



*Ako se malo vratimo na priču o stvarnim prioritetima procesa, u polju:* `prio`*, vidimo da je stvarni prioritet ovog procesa* `120`*. Dakle ovdje se radi o onom prioritetu, kako ga vidi kernel, a koji može imati vrijednost od* `1` *do* `139`.

Objasnit ćemo samo nekoliko polja koja smo vidjeli u ispisu, da dobijete dojam o tome koje sve informacije se ovdje spremaju za svaki pojedini proces:

- `se.vruntime` – označava vrijeme kada će se ovaj proces ponovno pokrenuti (*context switch*) (u *nano* sekundama).
- `se.sum_exec_runtime` – označava CPU vrijeme potrošeno na izvršavanje ovog procesa.
- `se.iowait_sum` – označava koliko vremena je proces proveo čekajući na I/O sustav (u *nano* sekundama).
- `nr_switches` – označava koliko je bilo prebacivanja konteksta/procesa (*context switch*).
- `se.load.weight` – vrijednost koja se izračunava na osnovi postavljene *Nice* vrijednosti (ako je postavljena).
- `prio` – vrijednost prioriteta procesa (spomenuli smo ju gore u napomeni).



Za više detalja o statistikama i analizi rada programa, u poglavlju:
**15. Analiza rada linux sustava i programa**.
Za više detalja o naredbi **`sysctl`** i njenoj konfiguracijskoj datoteci: **`sysctl.conf`** pogledajte poglavlje:
**4.5.8. Naredba *sysctl*.**

Dodatno, moguće su i optimizacije cijelog sustava, pomoću poznatih unosa unutar direktorija: `/proc/sys/kernel/` tj. unutar datoteka od kojih svaka ima svoju namjenu odnosno svaka datoteka je zadužena za određenu opciju ili parametar. Sve ove opcije i parametri se mogu mijenjati ručno (trenutno) ili trajno, pomoću `sysctl` unosa u datoteci: `/etc/sysctl.conf`.
Neke od opcija koje imamo ovdje, dostupne su preko ***sysctl*** sustava su (ovisno o inačici linux kernela):

| Sysctl varijabla | Vrijednost | Opis |
|---|---|---|
| **kernel.sched_latency_ns** | **24000000** | Ciljano je kašnjenje za zadatke vezane za CPU. Povećanje ove varijable povećava vremenski odsjek zadatka vezanog za CPU. Vremenski odsjek zadatka je njegov udio u razdoblju planiranja: *vremenski odsjek = razdoblje zakazivanja* ***x*** *(težina zadatka/ukupna težina zadataka u redu za izvođenje)*. Težina zadatka ovisi o ***nice*** razini zadatka i politici raspoređivanja. Minimalna težina zadatka za SCHED_OTHER zadatak je 15, što odgovara *nice* vrijednosti 19. Maksimalna težina zadatka je 88761, što odgovara *nice* vrijednosti -20. Vremenski odsjeci postaju manji kako se opterećenje povećava. Ova vrijednost također specificira maksimalnu količinu vremena tijekom kojeg se smatra da je zadatak u stanju spavanja pokrenut. Povećanje ove varijable povećava količinu vremena koje zadatak u buđenju može koristiti prije nego što bude preuzet (*preempt*), čime se povećava latencija *schedulera* za zadatke vezane za CPU. |
| **kernel.sched_migration_cost_ns** | **500000** | Količina vremena nakon posljednjeg izvršenja za koju se u odlukama o migraciji smatra da je zadatak "*cache hot*". Manja je vjerojatnost da će "vrući" zadatak biti migriran, pa povećanje ove varijable smanjuje migracije zadataka. |
| **kernel.sched_min_granularity_ns** | **10000000** | Minimalna granularnost prednosti za zadatke vezane za CPU. Za detalje pogledajte **sched_latency_ns**. |
| **kernel.sched_nr_migrate** | **32** | Kontrolira koliko se zadataka može premjestiti na druge procesorske jezgre putem softverskih signala prekida (*softirq*). Povećanje ove vrijednosti daje povećanje performansi za programske nîti *SCHED_OTHER*, ali na račun povećanja latencije za zadatke u stvarnom vremenu. |
| **kernel.sched_rr_timeslice_ms** | **100** | Definira se veličina kvanta vremena u milisekundama za round-robin mehanizam. |
| **kernel.sched_rt_period_us** | **1000000** | Definira razdoblje u µs (mikrosekundama) koje se smatra 100% propusnosti CPU-a. Promjene vrijednosti ovog razdoblja moraju biti vrlo dobro osmišljene jer je predugo ili premalo razdoblje jednako opasno. |
| **kernel.sched_rt_runtime_us** | **950000** | Ukupna propusnost dostupna za sve zadatke u stvarnom vremenu. Zadana vrijednosti je 950.000 µs (0,95 s) ili, drugim riječima, 95% propusnosti CPU-a. Postavljanje vrijednosti na **-1** znači da zadaci u stvarnom vremenu mogu koristiti do 100% CPU vremena. Ovo je dovoljno samo kada su zadaci u stvarnom vremenu dobro osmišljeni i nemaju očita upozorenja kao što su neograničene petlje dohvaćanja (*pooling loops*). |
| **kernel.sched_schedstats** | **0** | Omogućuje/onemogućuje statistike planera (*schedulera*). Omogućavanje ove značajke uzrokuje male troškove u performansama planera, ali je korisno za otklanjanje pogrešaka i/ili podešavanje performansi sustava. |
| **kernel.sched_time_avg_ms** | **1000** | Ovaj parametar postavlja razdoblje u kojem je prosječno vrijeme provedeno u izvršavanju zadataka u stvarnom vremenu. Taj prosjek pomaže **CFS**-u u donošenju odluka o balansiranju opterećenja i daje pokazatelj koliko je CPU zauzet zadacima visokog prioriteta u stvarnom vremenu. Optimalna postavka za ovaj parametar uvelike ovisi o radnom opterećenju i ovisi, između ostalog, o tome koliko se često zadaci u stvarnom vremenu izvode i koliko dugo. |
| **kernel.sched_tunable_scaling** | **1** | Ona kontrolira može li se planer (*scheduler*) prilagoditi ***sched_latency_ns***. Vrijednosti su **0** = ne prilagođavati, **1** = logaritamsko podešavanje, a **2** = linearno podešavanje. Izvršena prilagodba temelji se na broju CPU jezgri i povećava se logaritamski ili linearno kako se podrazumijeva u dostupnim vrijednostima. To je zbog činjenice da je s upotrebom više CPU jezgri vidljivo smanjenje u percipiranoj latenciji. |
| **kernel.sched_wakeup_granularity_ns** | **15000000** | Granularnost prevencije buđenja. Povećanje ove varijable smanjuje prevenciju buđenja, a smanjenjem se poboljšava latencija buđenja i propusnost za kritične zadatke kašnjenja, osobito kada se komponenta opterećenja kratkog ciklusa mora natjecati s komponentama vezanim za CPU. |

## Task/proces scheduler (planer) i novosti od 2023. godine (napredni dio).

**Completely fair scheduler (CFS)**
Kako smo već govorili task/process *scheduler completely fair scheduler* (**CFS**) je u upotrebi od kernela 2.6.23 od 2007. godine i od tada je, uz brojne tekuće izmjene, prilično dobro obavljao svoj posao. Međutim njegove mane sve više dolaze do izražaja, naročito u specifičnim slučajevima odnosno za specifične procese (pokrenute programe).
Jedan od ključnih ciljeva dizajna CFS-a bio je, kao što se može razumjeti iz njegovog naziva, pravednost (engl. *fairness*) odnosno osiguravanje da svaki proces (pokrenuti program) na sustavu dobije pravičan udio CPU vremena za izvršavanje. Ovaj se cilj postiže praćenjem koliko je vremena dobio svaki proces te pokretanjem onih procesa koji su dobili manje procesorskog vremena za izvršavanje od ostalih, pri čemu je vrijeme izvođenja svakog procesa skalirano njegovim tzv "***nice***" prioritetom.

Ispostavilo se da je CFS-ov bazni mehanizam pravednosti dovoljan za rješavanje mnogih problema s rasporedom izvršavanja programa od strane CPU-a. Međutim postoje i mnoga ograničenja odnosno slučajevi u kojima njegove odluke u odabiru i načinu izvršavanja procesa nisu najbolje. Primjerice on bi između ostalog trebao i maksimalno povećati korist upotrebe pred memorija procesora ali i sustava, što zahtijeva minimiziranje prebacivanja izvršavanja procesa između više jezgri CPU-a.
U isto vrijeme, trebao bi iskorištavati i sve dostupne CPU jezgre, ako ima posla za njih. Upravljanje napajanjem je također dodatna komplikacija; ponekad optimalne odluke za propusnost sustava moraju biti u drugom planu zbog očuvanja trajanja baterije. Hibridni sustavi kod kojih nisu sve jezgre CPU-a jednake, dodaju dodatne komplikacije. I tako dalje.

Jedno važno mjesto gdje su potrebna poboljšanja je rukovanje zahtjevima latencije. Nekim procesima možda nije potrebno puno CPU vremena za izvršavanje, ali kada im to vrijeme treba, potrebno im je brzo. Drugi će možda trebati više CPU vremena, ali mogu pričekati ako bude potrebno. CFS ne daje procesima način da izraze svoje zahtjeve za kašnjenjem (navedenom *latencijom*); "***nice***" vrijednosti (prioriteti) mogu se koristiti kako bi se procesu dalo više CPU vremena, ali to nije ista stvar. Dok se klase za planiranje u stvarnom vremenu mogu koristiti za rad osjetljiv na kašnjenje, ali rad u klasi u stvarnom vremenu je privilegirana operacija, a ovakvi procesi u stvarnom vremenu mogu loše utjecati na rad ostatka sustava.
Dakle poznata su određena ograničenja CFS-a koje nije moguće (jednostavno) riješiti bez temeljnog redizajna.

**Earliest Eligible Virtual Deadline First (EEVDF) scheduler**
Algoritam znan kao "*Earliest Eligible Virtual Deadline First*" (**EEVDF**) nije nov; opisali su ga u svom radu iz 1995. godine *Ion Stoica* i *Hussein Abdel-Wahab*. Njegov naziv sugerira nešto slično algoritmu *Earliest Deadline First* koji koristi ***deadline scheduler***, ali, za razliku od njega, **EEVDF** nije planer (*scheduler*) u stvarnom vremenu (*realtime*), pa radi na drugačiji način.

Poput *CFS*-a, *EEVDF* pokušava pravedno raspodijeliti dostupno CPU vrijeme među procesima koji se natječu za njega.
Ako, na primjer, postoji pet procesa koji se pokušavaju pokrenuti na jednoj jezgri procesora (CPU-a), svaki od tih procesa trebao bi dobiti 20% raspoloživog vremena. "***Nice***" vrijednost danog procesa može se koristiti za prilagodbu izračuna njegovog poštenog vremena izvršavanja; proces s nižom vrijednošću ***nice*** (a time i višim prioritetom) ima pravo na više CPU vremena na račun onih s višim ***nice*** vrijednostima. U ovom djelu mehanizma nema nikakvih razlika odnosno novina.

Zamislite vremenski period od jedne sekunde; tijekom tog vremena, u našem scenariju s pet procesa, svaki je proces trebao dobiti 200 ms CPU vremena. Zbog niza razloga, stvari nikad ne ispadnu baš tako; neki procesi će dobiti previše vremena, dok će drugi biti promijenjeni. Za svaki proces EEVDF izračunava razliku između vremena koje je proces trebao dobiti i koliko je zapravo dobio; ta se razlika naziva "**lag**" (kašnjenje). Proces s pozitivnom vrijednošću kašnjenja nije dobio svoj pravičan udio i trebao bi se zakazati prije nego onaj s negativnom vrijednošću kašnjenja.

Nadalje, proces se smatra "prihvatljivim" ako i samo ako je njegovo izračunato kašnjenje veće ili jednako nuli; bilo koji proces s negativnim kašnjenjem neće biti kvalificiran za izvršavanje. Za bilo koji neprihvatljivi proces, postojat će vrijeme u budućnosti kada će vrijeme na koje ima pravo sustići vrijeme koje je stvarno dobio i ponovno će postati prihvatljiv; a to vrijeme se smatra "prihvatljivim vremenom".

Drugi čimbenik koji se koristi je "*virtual deadline*", što je najranije vrijeme do kojeg je proces trebao primiti svoje CPU vrijeme za izvršavanje. Taj se rok izračunava dodavanjem vremenskog odsječka dodijeljenog procesu njegovom prihvatljivom vremenu. Primjerice proces s vremenskim odsječkom od 10 ms, a čije je prihvatljivo vrijeme 20 ms u budućnosti, imat će virtualni rok od 30 ms u budućnosti.

Srž EEVDF-a, kao što se može vidjeti u njegovom nazivu, jest da će se prvi pokrenuti proces s najranijim virtualnim rokom izvršavanja. Izbor zakazivanja izvršavanja je stoga vođen kombinacijom pravednosti (vrijednost kašnjenja koja se koristi za izračun prihvatljivog vremena) i količine vremena koju svaki proces trenutno ima zbog toga.

S navedenim, implementacija bržeg pristupa za procese osjetljive na kašnjenje događa se prirodno. Kada *scheduler* (planer) izračunava vremenski odsječak za svaki proces, on uzima u obzir dodijeljenu vrijednost latencije tog procesa; proces s nižom postavkom kašnjenja (a time i strožim zahtjevima za kašnjenjem) dobit će kraći vremenski odsječak za izvršavanje.
Procesi koji su relativno indiferentni prema latenciji dobit će duže odsječke. Imajte na umu da će količina procesorskog vremena dana za bilo koja dva procesa (s istom ***nice*** vrijednošću) biti ista, ali će proces niske latencije to dobiti u većem broju kraćih odsječaka vremena za izvršavanje.



Od sredine 2023. godine **EEVDF** pokazuje bolje rezultate od **CFS** do te granice da će se [(**1480**)] ubrzo uvrstiti u postojeći kernel. Prema planu, od kernela 6.6. **EEVDF** će biti zadani *task/proces scheduler* **[(1533)]**.

**Izvori informacija: (221),(1204),(1205),(1206),(1207),(1208),(1209),(1210),(1211),(1479),(1480),(1532),(1533),(K-5),**
**`man chrt`, `man proc`,`man ps`,`man 2 sched_setaffinity`, `man sched_setscheduler`,`man 7 sched`.**

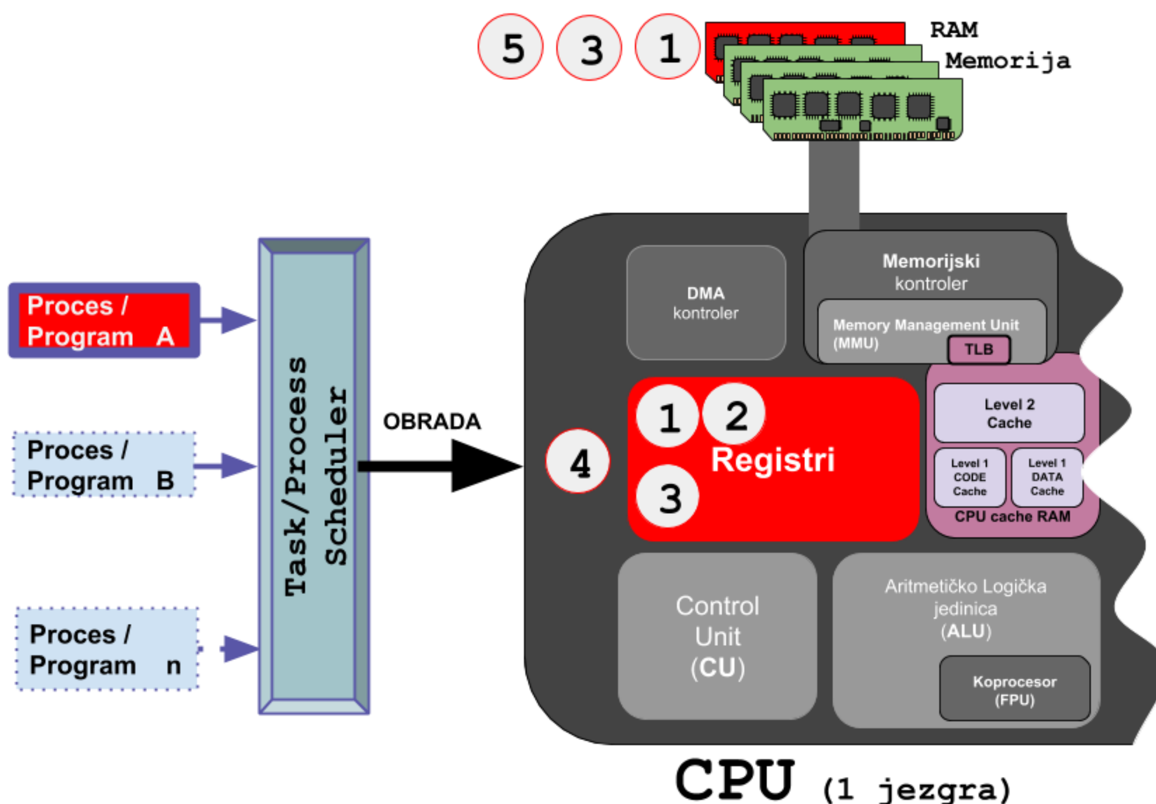
## 9.3.1. Context switching

Malo prije smo rekli kako je *task scheduler* poznat i kao *process scheduler* mehanizam zadužen za obradu odnosno baratanje s procesima. Zbog činjenice kako se na vrlo niskoj razini, na jednoj jezgri centralnog procesora (*CPU*) u pojedinom vremenskom okviru može odrađivati (procesirati) samo jedan proces odnosno program (aplikacija), ovaj mehanizam je vrlo važan za performanse cijelog operativnog sustava. Za ovaj mehanizam je u konačnici zadužen takozvani ***Context Switching***.
***Context switch*** je mehanizam u kojem se proces odnosno program ili njegove programske nîti koje se trebaju obraditi, u prvoj jedinici vremena prvo učitavaju u *CPU*, te obrađuju. Potom se u drugoj jedinici vremena pauziraju kako bi se njihovo stanje i međurezultati privremeno zapisali u RAM memoriju, a potom obrisali iz priručne memorije *CPU-a*. Sve se to događa zbog prebacivanja na izvršavanje novog procesa (programa) ili njegovih programskih nîti. Dakle sustav konstantno obrađuje sve programe/procese: jedan, po jedan, u vrlo malim vremenskim okvirima i tako u krug, od početka. To nam daje privid istovremenog izvršavanja više programa ili programskih nîti, ako govorimo o procesoru s jednom jezgrom.
Ako primjerice imamo procesor s četiri jezgre, na njemu se istovremeno mogu pokretati samo četiri procesa odnosno programa (u svakoj jedinici vremena). Mi ćemo analizirati sustav s jednom CPU jezgrom, zbog lakšeg razumijevanja problematike.

Prema definiciji *context switch* je proces pohranjivanja i ponovnog preuzimanja stanja (*contexta*) procesa u kojem njegova obrada može biti privremeno zaustavljena odnosno pauzirana, ali i ponovno pokrenuta u bilo kojem trenutku.

Naime ***Context switch*** mehanizam se konstantno ponavlja pošto svaki operativni sustav ima pokrenuto na desetke programa odnosno procesa kao i više nîti unutar svakog procesa. Dakle ovo prebacivanje između procesa ili nîti unutar procesa se događa na desetke, stotine ili tisuće puta u sekundi i to svake sekunde. Pogledajmo pojedini proces ili nît procesa na nižoj razini odnosno na razini njegovih logičkih cjelina ili funkcija koje se izvršavaju unutar njega, kako je vidljivo na slici 29.

**Slika 29. *Context switching*** koji se ponavlja za svaki pokrenuti program odnosno proces.




Zamislimo program koji radi određene matematičke operacije. Na početku naš program želi krenuti s izračunavanjem, pa *task scheduler* odrađuje prvo prebacivanje (*context switch*) s trenutnog programa koji je na obradi unutar *CPU-a*, na naš program:

- S obzirom kako je već neki drugi proces ili programska nît na obradi, prvo se svi registri CPU-a privremeno spremaju (snimaju) u RAM memoriju (**1**), te se potom brišu svi registri procesora (**2**), kako bi oslobodili mjesto za novi proces ili nît procesa odnosno novu cjelinu za procesorsku obradu.
- Zatim se od strane našeg (novog) programa učitavaju novi podaci u procesorske registre (**3**).
- Potom kreće obrada odnosno izvršavanje našeg programa, koji odrađuje prve matematičke operacije (**4**).
- Sve se odrađuje unutar vremenskog okvira koji smo dobili od task/proces schedulera, za naš proces (program).
- Rezultat prve matematičke operacije našeg programa se zapisuje u RAM memoriju (**5**).
- Nakon toga *task/process scheduler* prebacuje na obradu na drugi proces ili programsku nît odnosno ponavlja se prvi korak (**1**), ali za drugi program, pa potom slijedi drugi korak (**2**) te ponovno sve u krug od početka.



Za više detalja o registrima procesora (*CPUa)*, pogledajte poglavlje:
**10.7.1 CPU registri.**

Ovo je samo pojednostavljeni opis onog što se događa kod *context switchinga*; zapravo se tu mijenjaju:

- Registri.
- Pokazivači stôga (*stack pointeri*).
- Adresni prostor i drugi detalji.

Dakle *context switching* je osnovni mehanizam koji nam daje privid istovremenog izvršavanja više programa (procesa) unutar jedne jezgre CPU-a. Dakle *context switching* je osnovni mehanizam koji osigurava višezadaćnost koju nazivamo i *Multitasking*. *Context switch* mehanizam radi *proces* ili *thread switching* odnosno prebacivanje između procesa ili između programskih nîti unutar jednog programa odnosno procesa. Ovaj mehanizam (*context switch*) se često naziva i *task switch*.



Sada proučite poglavlja:
**12.3.4 Zbog čega toliko priče o *Page Table-u*** (TLB).
**12.3.2.3 Proces translacije adresa.**

Nakon što ste proučili gore navedena poglavlja možemo krenuti dalje. Naime svako novo prebacivanje između procesa odnosno *Context Switch*, prazni i ***TLB*** memoriju, te ju ponovno popunjava pretraživanjem glavne *Page* tablice tijekom prebacivanja obrade na novi proces. Ovo vrijedi i za programske nîti (engl. *Thread*). S obzirom kako se svake sekunde događa ogroman broj ovih prebacivanja između aktivnih procesa, jasno je koliko ovo može utjecati, a i utiče na performanse cijelog sustava. Međutim, kod novijih procesora, postoji podrška za ***tagged TLB*** metodu, kod koje se ne mora prazniti cijeli *TLB*, tijekom prebacivanja između procesa.
Naime kod ove metode, moguće je zasebno označavanje (engl. *Tag*) sâmo određenog dijela adresnog prostora, unutar *TLB* tablice, čime se prilično ubrzao proces, jer se tada više ne mora prazniti cijela *TLB* memorija.

Dodatno, svaki pojedini proces koji je trenutno na obradi, može u bilo kojem trenutku čekati na neki događaj (engl. *Event*) ili na neki uređaj; primjerice:

- Čekanje na pristup nekom hardverskom uređaju.
- Čekanje na zapisivanje na disk, jer je diskovni podsustav preopterećen drugim operacijama zapisivanja ili čitanja.
- Čekanje podataka s mreže, jer se još uvijek čitaju podaci iz mrežnih međuspremnika, a očekuju se novi "živi" podaci s mreže i slično.

U tom trenutku čekanja, aktivni proces ništa ne radi već je u stanju čekanja na neku drugu komponentu ili dio operativnog sustava. U takvim slučajevima je *moguće* da *task scheduler* oduzme vrijeme za obradu tom procesu, te se prebaci na obradu drugog procesa. Dakle on može napraviti novi *contex switch*, a potom kada prvi proces ponovno postane aktivan, jer ne mora čekati na prijem podataka od komponente zbog koje je čekao, da se prebaci na njega; i dalje sve iz početka (u krug).

Ovakav scenarij je moguć samo u slučajevima koji nisu prešli određene granice. Konkretan primjer bi bilo čekanje na zapisivanje prema diskovnom podsustavu, koje može biti asinkrono ili sinkrono. U oba slučaja, kada se popune sve međumemorije, naš proces koji je trebao nešto zapisati na disk, morat će ostati u stanju čekanja na dovršetak operacija snimanja na disk. Kod sinkronog zapisivanja ovo zagušenje će se kod velikog opterećenja prema diskovnom sustavu vrlo brzo dogoditi, a kod asinkronog nešto kasnije, jer se koriste dodatne međumemorije Linuxovog diskovnog podsustava.



Indikacija ovakvog stanja čekanja je veliko `I/O Wait` vrijeme procesora (*CPUa*). Za detalje pogledajte poglavlje: **14. Diskovni (I/O) podsustav**, i to poglavlje: **14.1.1. Sinkroni i Asinkroni I/O**.



Kako biste razumjeli hardverska ograničenja, koja utječu na *context switching*, pogledajte detaljnije poglavlje:
**10.7.1.3 CPU registri još detaljnije**.

Tek sada možemo uvrstiti i veći dio varijabli u izračun ukupnih troškova *context switchinga*:

- (1) Uslijed *context switcha* procesor mora kopirati međurezultate u RAM memoriju (cca. 60 **ns**).
- (2) Potom prazniti registre (nekoliko **ns**) te prazniti *TLB* memoriju (nekoliko **ns**). Uslijed promašaja *TLB*, dodatno:
  - 10 — 100 taktova *CPUa:* od nekoliko **ns** do nekoliko desetaka **ns**.
- (3) Te prazniti i/ili popunjavati/osvježavati *TLB cache* ili priručne: **L1**, **L2** ili **L3** memorije unutar procesora:
  - **L1** memorija: 1,2 — 2,1 **ns**.
  - **L2** memorija: 3 — 5,3 **ns**.
  - **L3** memorija: Ovisno o jezgri i/ili fizičkom procesoru
    - 12 — 21,4 **ns**: lokalno na istoj CPU jezgri.
    - 19,5 — 34,8**ns:** kod dijeljenje s drugom CPU jezgrom na istom fizičkom procesoru.
  - **L3 NUMA**: na ***NUMA*** arhitekturi s više fizičkih procesora, od kojih svaki ima više CPU jezgri:
    - 30 — 160,7 **ns** : CPU jezgra na prvom fizičkom procesoru - CPU jezgra na drugom fizičkom procesoru.
- (4) Zatim dohvatiti podatke od novog programa/procesa (1), popuniti registre (2) kao i popuniti *cache* međumemoriju (3).
- (5) Te konačno početi obrađivati (procesirati) podatke odnosno pokrenuti zadani program/proces.

U praksi, kada aplikacija koristi ***SMP*** sustav s više *CPU* jezgri, za ovakav sustav standardno nemamo kontrolu na kojoj će CPU jezgri u kojem trenutku proces biti prebačen dinamički, jer smo prepustili sustavu da se brine o tome.
Ovdje možemo očekivati sljedeća vremena pristupa (ovisno o CPU):

- CPU: Intel E5520 (2.27GHz): 4.500 ns (**4,5 µs**).
- CPU: Intel 5150 (2.67GHz): 4.300 ns (**4,3 µs**).
- CPU: Intel E5440 (2.83GHz): 3.600 ns (**3,6 µs**).
- CPU: Intel E5-2620 (2Ghz): 3.000 ns (**3 µs**).

Dakle vidimo vremena između 3.000 ns (**3 µs**) i 4.500 ns (**4,5 µs**) sâmo za prebacivanje s procesa na proces odnosno ***context switch***. Druga situacija je u slučajevima kada imamo ***SMP*** sustav, s više CPU jezgri, ali smo ručno „*zalijepili*" proces na točno definiranu CPU jezgru odnosno naložili sustavu da određeni proces (program) mora izvršavati sâmo određena CPU jezgra:

- CPU: Intel E5520: 1.400 ns (**1,4 µs**), CPU: Intel 5150: 1.900 ns (**1,9 µs**).
- CPU: Intel E5440: 1.300 ns (**1,3 µs**), CPU: Intel E5-2620: 1.600 ns (**1,6 µs**).

Sada vidimo kako su se vremena smanjila, na najmanje 1.300 ns (**1,3 µs**) odnosno najviše 1.900 ns (**1,9 µs**). Jasno je i kako sve ovisi o arhitekturi procesora i sâmog sustava. U oba primjera testa, radilo se sâmo prebacivanja s procesa na proces, bez ikakve obrade unutar sâmog procesa, dakle ovo su samo vremena potrebna kako bi se prebacilo s jednog procesa na drugi odnosno odradio jedan jedan *context switch*. Na novijim generacijama procesora ova vremena jesu i biti će sve manja, ali ne drastično.

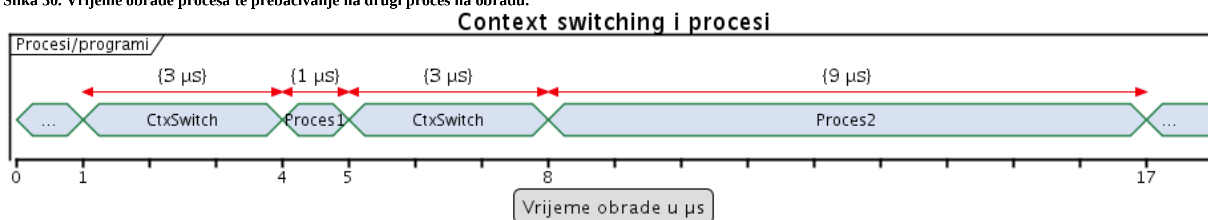
**Izvor informacija/testa: (294) i prateći dokumenti.**

# Vrijeme obrade

Pogledajmo sada primjer dijagrama vremena, s uključenim odnosno vidljivim vremenom za sâmo prebacivanje (*context switch)* kao i sâmu obradu procesa. U primjeru imamo dva procesa:

- *Proces1* koji se obrađuje 1 µs.
- *Proces2* koji se obrađuje 9 µs.
- Te sâm mehanizam *context switch* (**CtxSwitch**) za koji smo uzeli srednju vrijednost od **3µs**

**Slika 30. Vrijeme obrade procesa te prebacivanje na drugi proces na obradu.**




Na slici 30. je vidljivo kako se prije prebacivanja obrade s jednog procesa na drugi, odrađuju sve radnje unutar *context switch* mehanizma (gore navedene točke 1-3), a tek potom se kreće s obradom novog procesa i tako u krug. Vjerujemo kako je sada puno jasnije koliko „*košta*" svaki *context switch*, a bez kojeg ionako ne možemo (do neke granice).



Dodatno, pogledajte poglavlje:
**10.7.1.3 CPU registri još detaljnije** i to dio: „**I još malo detalja vezanih uz CPU, memoriju i brzinu obrade**".

Kako smo naveli u gore navedenom poglavlju o CPU registrima, u primjeru obrade mrežnih paketa, je vidljivo slijedeće:

| **Brzina mreže** | **Maksimalno vrijeme obrade** | **Broj taktova/ciklusa CPUa (2GHz) unutar koji se mora obraditi svaki mrežni paket** |
|---|---|---|
| 1 Gbps | 672 ns | 1.340 ciklusa |
| 10 Gbps | 67,2 ns | 134 ciklusa |
| 40 Gbps | 16,8 ns | 33,5 ciklusa |
| 100 Gbps | 6,7 ns | 13,4 ciklusa |

To znači kako *CPU* koji radi na *2GHz*, za obradu svakog pojedinog mrežnog paketa, na brzini od 1Gbps, ima vremena 1.340 CPU ciklusa, odnosno 672 *ns*. Dok već na brzini mreže od 10Gbps, isti *CPU* mora svaki mrežni paket obraditi za samo 134 CPU ciklusa, odnosno da ima samo 67,2 *ns* vremena za to.
U slučaju kada bi se procesirao svaki paket zasebno, što se može povremeno dogoditi, ali što srećom nije baš čest slučaj u primjerima dobro napisanih upravljačkih programa za mrežne kartice, kao i drugih komponenti mrežnog podsustava, to bi već postalo potencijalno problematično.
Naime dobro napisane/razvijene komponente mrežnog sustava trebaju imati dobar balans između grupiranja više mrežnih paketa koji se šalju na obradu i brzog odziva na obradu paketa. U konkretnom primjeru, to postaje veliki izazov (problem) za programere koji razvijaju ove sustave, jer se na sve većim brzinama mora paziti na svaki takt rada *CPUa* odnosno na svaku potrošenu *nano* sekundu (*ns*).
Naime standardno vrijeme obrade pojedinog mrežnog paketa je obično u rasponu od nekoliko stotina CPU ciklusa do čak nešto preko tisuću (1.000) CPU ciklusa. Sve ovisno o izvedbi svih slojeva mrežnog podsustava i u konačnici načina obrade odnosno procesiranja paketa odnosno procesa koji je zadužen za njihovu obradu, ali i *context switchinga*.
Iako izgleda kao teorija, stvarno su moguće i dostupne razne tehnike i tehnologije za tu svrhu, a jedan od primjera je upotreba sistemskih biblioteka, koje su visoko optimizirane za brzu obradu mrežnih paketa poput **DPDK**, koji unutar sâmo **80 CPU** ciklusa, modernih procesora i posebnih mrežnih kartica s pripadajućim upravljačkim programima, može primiti ili poslati pojedini mrežni paket. Sljedeći primjer je **netmap** *framework*, koji je također razvijen za sličnu namjenu, a koji omogućava programima mogućnost procesiranja i prosljeđivanja mrežnih paketa vrlo velikim brzinama (unutar **90 CPU** ciklusa), korištenjem *netmap API* poziva.

**Netmap** *Framework* se danas standardno koristi na **FreeBSD Unixu** (još od inačice **9.1**), ali sve više i na Linuxu, na kojem je integriran i sa **KVM**/**QEMU** sustavom za virtualizaciju. Jedan od primjera je i **PF RING** koji je razvijen za potrebe brzog dohvaćanja, filtriranja i analize mrežnih paketa, najviše za potrebe programa za analizu mrežnog prometa, poput programa ***ntop*** odnosno ***ntopng***. Drugi primjer je upotreba kombinacije novog mehanizma za dohvaćanje mrežnih paketa, unutar upravljačkog programa mrežne kartice, naziva **XDP**, te upotrebe pripadajućeg eBPF filtera.


Pogledajte poglavlje:
**26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP* + *eBPF*).**

U svim ovim slučajevima, zbog dobivanja drastično boljih performansi, uglavnom se zaobilazi *kernel* ili njegov mrežni stôg (*network stack)* i omogućava pristup fizičkoj (ili virtualnoj) mrežnoj kartici, preko *API* poziva koje nam daje neki od navedenih *frameworka*. Druga priča je razvoj visoko optimiziranih upravljačkih programa, a koji nisu dostupni za sve mrežne kartice, već obično sâmo one posebne poslužiteljske (*serverske)* i to od strane većih proizvođača mrežnih kartica (uređaja).

## Krenimo na primjere

Provjerimo koliko procesa se kreira svake sekunde te koliko *context switch*-eva je aktivno u svakoj sekundi, na razini cijelog operativnog sustava. Pratit ćemo ove statistike svake sekunde (`1`) i to četiri puta (`4`) za redom. Za ovu namjenu koristimo naredbu `sar` iz softverskog paketa `sysstat` koji smo već objasnili u prijašnjim poglavljima. Pokrenimo ovu naredbu:

```
sar -w 1 4

Linux 2.6.32-42-pve (server1)  11/11/2015      _x86_64_        (4 CPU)

21:20:38 PM    proc/s   cswch/s
21:20:39 PM      1.00  28053.00
21:20:40 PM      2.00  28518.00
21:20:41 PM     10.00  28349.00
21:20:42 PM      1.00  28313.00
Average:         3.50  28308.25
```

Opis rezultata (ispisa) slijedi:

- `proc/s` - ovo je broj procesa koji se pokreću svake sekunde, na razini cijelog sustava
- `cswch/s` - ovo je broj ukupnih *context switch*eva na cijelom sustavu.
  Ovdje je vidljivo kako se na cijelom sustavu odrađuje konstantno oko 28.000 *context switcheva* u sekundi.

Vidljivo je i kako se na našem Linuxu u prosjeku kreiraju 3,5 procesa u sekundi. Također je vidljivo da imamo prosječno oko 28.308 aktivnih *context switch-eva*. U primjeru je ispis sa prilično opterećenog poslužitelja s velikim brojem aktivnih procesa odnosno pokrenutih programa/aplikacija.

Provjerimo koliko naš (konkretan) proces ima aktivnih *context switching-a;* ručnom provjerom. Dakle nas zanima naš *Apache* (`httpd`) poslužiteljski proces. Prvo pronađimo njegov proces ID (***PID***), pomoću naredbe `pidof` na sljedeći način:

```
pidof httpd

2738847 6441
```

Dobili smo dva ***PID***-a, a mi ćemo gledati ***PID*** `6441`. Unutar datoteke `status` su vidljive razne statistike koje se osvježavaju u realnom vremenu. Između ostalih tu je i statistika o *context switchingu*. Kao što smo spominjali u poglavlju *Diskovni I/O sustav*, unutar direktorija `/proc/PID/` se nalazi i datoteka `status` iz koje tražimo željene statistike, koje ćemo filtrirati pomoću programa odnosno naredbe `grep`.
Važno je napomenuti kako se vrijednosti odnosno statistike koje se ovdje zapisuju, zbrajaju odnosno vrijednosti stalno rastu.

Pogledajmo ovu datoteku, za naš proces s ***PID*** brojem `6441`:

```
cat /proc/6441/status | grep ctxt

voluntary_ctxt_switches:        1109747
nonvoluntary_ctxt_switches:     13
```

Opis rezultata ispisa je:

- `voluntary_ctxt_switches` - ovo je broj aktivnih *context switch*eva (zbrojno: povećava/zbraja se).
- `nonvoluntary_ctxt_switches` - ovo je broj potencijalno problematičnih *context switch*eva.

Što znači *problematični*:
**Non-Voluntary** *context switch*ing se događa, kada operacijski sustav **oduzima** nît (*thread*) od *CPUa* bez da je zahtjev za oduzimanjem odnosno zatvaranjem tog *context switch*a (*thread*a) stvarno zahtjevan od strane procesa odnosno sâme aplikacije.
Ovo se može događati, ako su CPU resursi postali vrlo ograničeni te se počinju *nasilno* zatvarati programske nîti (*context switches/threads*). To često nije dobar pokazatelj i težimo što manjoj statistici na ovom parametru.
Provjerimo koliko naš proces ima aktivnih *context switching*a. Pomoću naredbe `pidstat` želimo statistike svake sekunde (`1`) i to četiri puta (`4`). Ako i dalje pratimo proces koji ima ***PID*** broj `6441` tada pokrenimo sljedeću naredbu:

```
pidstat -w -p 6441 1 4

02:01:11 PM       PID   cswch/s nvcswch/s  Command
02:01:12 PM      6441      1.00      0.00  httpd
02:01:13 PM      6441      1.00      0.00  httpd
02:01:14 PM      6441      1.00      0.00  httpd
02:01:15 PM      6441      1.00      0.00  httpd
Average:         6441      1.00      0.00  httpd
```

Pogledajmo opis ispisa:

- `cswch/s` - ovo je broj ***Voluntary*** *context switch-eva* u sekundi.
- `nvcswch/s` - je broj ***Non*** *voluntary context switch-eva* u sekundi. I ovdje se teži da je broj `nvcswch/s` što manji.

U slučaju kada želimo ispisati listu svih aktivnih procesa i njihov trenutni broj *context switching*a, potrebno je smo pozvati naredbu `pidstat` sa prekidačem `-w` na sljedeći način:

```
pidstat -w

01:39:16 PM       PID   cswch/s nvcswch/s  Command
01:39:16 PM         1      0.20      0.00  init
01:39:16 PM         2      0.00      0.00  kthreadd
01:39:16 PM         3      0.03      0.00  migration/0
01:39:16 PM         4      3.05      0.00  ksoftirqd/0
01:39:16 PM         5      0.00      0.00  stopper/0
01:39:16 PM         6      0.08      0.00  watchdog/0
```

Ovo (`pidstat -w`) je korisna naredba, ako želimo saznati koji od aktivnih (pokrenutih) procesa (programa/aplikacija) ima najveći broj *context switch-eva*.
Pogledajmo i primjer praćenja broja *context switch*eva na razini aplikacije, u realnom vremenu, ručnom metodom.
Naime ovu statistiku možemo pratiti i u stvarnom vremenu, koja se svake dvije sekunde osvježava, pomoću naredbe `watch`. Važno je napomenuti kako se ove vrijednosti također zbrajaju. Dakle ovdje ispisujemo sadržaj datoteke: `/proc/6441/status` te tražimo pojam: `ctxt` te osvježavamo ispis svake dvije sekunde pomoću naredbe: `watch`.

Pogledajmo kako ćemo to izvesti:

```
watch -d „cat /proc/6441/status | grep ctxt"

Every 2.0s: cat /proc/6441/status | grep ctxt    Mon Nov 11 22:46:30 2015

voluntary_ctxt_switches:        1110909
nonvoluntary_ctxt_switches:     13
```

Ovo praćenje se prekida s kombinacijom tipki: `CTRL` `C`

**Izvori informacija: (K-5), `man sar`, `man pidstat`, `man pidof`, `man 5 proc`.**

### 9.3.1.1. Još detaljniji rad procesa i nîti

Procesi su zapravo pokrenuti programi i kao takvi imaju svoj životni vijek: od trenutka pokretanja, rada i završetka rada odnosno izvršavanja. Osim toga oni mogu kreirati i novi pôd proces. Svaki proces ima pet osnovnih dijelova:

- Izvršni dio odnosno izvršni kôd (vidljiv kao „***text***“ dio).
- Dio s podacima (vidljiv kao „***data***“ dio) i stôg dio (engl. ***Stack***).
- Datotečni ulazno izlazni dio (*file I/O*) te dio s tablicom signala.

U ovom poglavlju proširit ćemo znanja o procesima i analizirat ćemo njihov rad znatno detaljnije, pa će neka pojednostavljenja koja smo do sada napravili, zvučati malo drugačije. Svrha procesa u kernelu je da djeluje kao entitet na koji su raspoređeni resursi sustava: CPU vrijeme, memorija, mrežni i diskovni pristup i drugo. Kada se pokrene novi pôd proces on je gotovo identičan roditeljskom procesu. On dobiva (logičku) kopiju roditeljskog adresnog prostora i izvršava isti programski kôd kao i roditelj, počevši od sljedeće instrukcije nakon poziva sustava. Iako procesi roditelj i dijete mogu dijeliti stranice memorije koje sadrže programski kôd („*tekst*“ dio), oni imaju zasebne kopije podataka („***stog***“ i „***heap***“ dijelovi), tako da su promjene od strane procesa djeteta na memorijske lokacije koje su u adresnom prostoru djeteta nevidljive procesu roditelju i obrnuto.

Pri tome svaki proces ima svoj jedinstveni ***PID*** broj, a proces dijete dobiva ***PPID*** broj od roditelja. Detaljnije, svaki proces je identificiran sa ***PID***, ***PPID***, ***PGID***, ***TID***, ***TGID*** brojevima. Pokrenimo naredbu `ps`, ali s dodatnim parametrima, s kojim ćemo moći vidjeti sve što nas zanima o našem pokrenutom procesu (***PID*** broja `4071`) i svim njegovim identifikatorima:

```
ps -mo pid,ppid,pgid,tid,tgid,fname,user,psr -p 4071

  PID  PPID  PGID   TID  TGID COMMAND  USER      PSR
 4071     1  4071     -  4071 kvm      root        -
    -     -     -  4071     - -        root        5
    -     -     -  4096     - -        root        3
```

Vidimo kako naš proces (***PID*** `4071`) ima više programskih nîti; pogledajte *Thread ID* (`TID`): `4071` i `4096` koje su trenutno aktivne.
Možda ih ima i više, ali trenutno nisu aktivne te kako se neke od njih pokreću na CPU jezgri (`PSR`) `5`, a neke na CPU jezgri `3`.
Ovdje vidimo i sljedeće:

- `PID` - je ***PID*** (*Process ID*) broj trenutnog (našeg) procesa. Poglavlja: **9.** i **9.6.** te **10.7.2.3.1.**
- `PPID` - je broj roditeljskog procesa koji je pokrenuo naš proces (a to je *INIT* ili *SYSTEMD* sa ***PID br.1***).
- `PGID` - je procesna grupa (*Process Group ID (**PGID**)*) kojoj naš proces pripada. Poglavlje: **9.6.**
- `TID` - je ***TID*** (*Thread ID*) odnosno ***ID*** od svake zasebne programske nîti našeg procesa.
  Pogledajte poglavlje: **10.7.2.1.2. Process/Thread Affinity i CPU Affinity.**
- `TGID` - ***TGID*** (*Thread Group ID*) je broj koji koristi ***PID*** broj procesa, ako se radi o njegovoj procesnoj nîti. Međutim ako se radi o račvanju programa (Engl. *Fork*) odnosno novoj instanci programa, tada nova instanca dobiva drugi ***PID*** i pripadajući (isti) ***TGID*** broj.

Što se tiče izvršavanja programa (procesa); njemu je potrebno više od binarnog (izvršnog) kôda koji govori računalu što treba učiniti. Program treba memoriju i razne resurse operativnog sustava kako bi se pokrenuo. *Proces* je ono što nazivamo programom koji je učitan u memoriju; zajedno sa svim resursima koji su mu potrebni za rad. Svaki proces se može izvršavati u više logičkih cjelina odnosno programskih nîti paralelno, samo, ako je tako napisan odnosno razvijen.

Kad se proces pokrene, dodjeljuje mu se memorija i svi ostali resursi. Svaka nît u tom procesu dijeli tu memoriju i ostale resurse sustava. U procesima s jednom nîti, proces logično sadrži jednu programsku nît što znači kako ima jedan logički nîz instrukcija (naredbi) koje izvršava.

M*ultithreaded* procesi odnosno procesi koji sadrže više od jedne nîti, rade tako da se unutar procesa istodobno odnosno paralelno može odrađivati više logičkih nîzova naredbi odnosno zadataka. Međutim, ako sve pogledamo malo detaljnije, postoje ipak neke razlike o kojim ćemo sada govoriti. Naime postoje dvije kategorije procesa:

- *Lagani procesi* (engl. ***Lightweight***).
- *Teški procesi* (engl. ***Heavyweight***).

Pod *teškim procesima* smatramo način rada u kojem se paralelno izvode višestruki procesi. Svaki proces u ovoj kategoriji ima svoj memorijski adresni prostor i resurse računala koje koristi. Komunikacija među ovakvim procesima je sporija jer procesi imaju različite memorijske adrese. Dakle ovakvi procesi ne dijele memoriju s drugim procesima.
Stoga komunikacija između ovih procesa uključuje standardne Unix/Linux komunikacijske mehanizme kao što su: *unix socketi* i *pipes mehanizam* (o njima nešto kasnije). Dakle ovo je normalan način rada i komunikacije između procesa koji koristi standardno ugrađene mehanizme koji se nalaze u *Unix* i *Linux* operativnim sustavima.

Preklapanje konteksta (engl. *Context switch*) odnosno prebacivanje na obradu (CPU) između takvih procesa je sâmim time „skuplje“ odnosno duže traje. Kod *laganih procesa* radi se o tome da oni mogu imati više programskih nîti unutar procesa.
Ove nîti odnosno logičke cjeline, koje možemo promatrati i kao logičke grane ili smjerove izvršavanja, a koriste se za međusobno dijeljenje zadataka ili funkcionalnosti unutar procesa, ali i raspodjelu opterećenja prema resursima računala: CPU, memorija, ili disk. *Programske nîti* unutar procesa koriste memoriju procesa kojem pripadaju.

Stoga komunikacija između nîti može biti brža od međusobne komunikacije odvojenih procesa jer nîti istog procesa dijele memoriju s procesom kojem pripadaju i tako mogu komunicirati. Rezultat komunikacije među nîtima je vrlo jednostavna i učinkovita komunikacija. Preklapanje konteksta izvršavanja (*Context switch*) između *nîti* istog procesa je *jeftinije* odnosno brže i s manje resursa u odnosu na prebacivanje između različitih (*teških*) procesa (*heavyweight*). To je zato što nîti unutar jednog procesa dijele memoriju s ostalim nîtima istog procesa te se ne moraju prazniti i popunjavati svi međuspremnici i registri procesora. Dodatno, postoje dvije vrste programskih nîti:

- Nîti na razini korisnika odnosno korisničkih programa (engl. ***User-level***).
- Nîti na razini kernela (engl. ***Kernel-level***) odnosno programa koji se izvršavaju na ovoj razini.

Nîti na razini korisnika izbjegavaju kernel i samostalno upravljaju radom. Nîti na razini korisnika imaju problem da jedna nît može monopolizirati vremenski okvir izvršavanja tako da se ostale nîti unutar procesa usporavaju odnosno čekaju na svoj red izvršavanja. *Nîti* na razini korisnika obično se izvršavaju logički iznad razine kernela u takozvanom korisničkom prostoru (*user-space*) i rade bez uplitanja kernela. Kernel ne zna ništa o *nîtima* na razini korisnika i upravlja s njima kao da su procesi s jednom *nîti*. Kao takve, *nîti* na razini korisnika su vrlo brze. Međutim *nîti* na razini kernela se često implementiraju zbog paralelnog izvršavanja dijelova programa. U tom slučaju, kernel raspoređuje svaku *nît* unutar vremenskog okvira izvršavanja svakog pojedinog procesa. Ovdje, budući da će mehanizam prebacivanja obrade procesa (*proces/thread scheduler*) odrediti vrijeme prebacivanja konteksta (*context switch*), manje je vjerojatno da će trenutna nît (zadatak) oduzeti vrijeme za obradu od drugih nîti unutar procesa. Nîtima na razini kernela izravno upravlja operativni sustav.
Odnos između nîti na razini korisnika i nîti na razini kernela nije potpuno neovisan, u stvari postoji interakcija između ove dvije razine. Općenito, nîti na razini korisnika mogu se implementirati pomoću jednog od četiri modela pristupa prema nîtima na razini kernela:

- Više na jedan (engl. M*any-to-one*).
- Jedan na jedan (engl. *One-to-one*).
- Mnogi na mnoge (engl. *Many-to-many*).
- Dvo razinski (engl. *Two-level*).

Dakle svi ovi modeli preslikavaju nîti na razini korisnika (*user-level-thread*) na nîti na razini kernela (*kernel-level-thread*) i sâmim time uzrokuju interakciju u različitim stupnjevima, između obje razine.

**Slika 31. Višenîtnost detaljnije.**

**Implementing threads**

**User-level threads**

All **code** and **data structures** for the library exist in user space.

Invoking a function in the API results in a **local function call** in user **space** and not a system call.

user mode

**API: application programming interface*

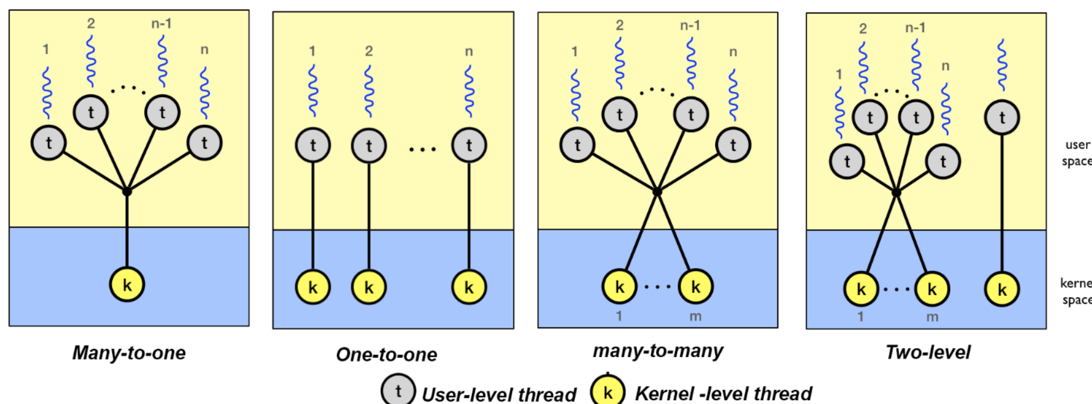
**Kernel-level threads**

All **code** and **data structures** for the library exists in **kernel space**.

Invoking a function in the API typically results in a **system call** to the kernel.

kernel mode

**User-level thread models**




Izvor fotografije: *Uppsala University — Information Technology:* http://www.it.uu.se/education/course/homepage/os/vt18/module-4/implementing-threads/

Pogledajmo i ove poveznice kako je vidljivo na slici **31**. Važno je razumjeti kako programske nîti na razini kernela pokreće sâm kernel. One su različite od nîti na razini korisnika u činjenici da *nîti* na razini kernela nemaju ograničeni adresni prostor.

Nîti na razini kernela „žive“ isključivo u prostoru kernela, a nikada se ne prebacuju u korisnički prostor, što je sa sigurnosne strane dobro. Nîti na razini kernela su u potpunosti prerasporedivi entiteti, s točke preraspodjeljivanja resursa (od strane *proces/thread scheduling* mehanizma).
Svrha programa i njihovih nîti na razini kernela je uglavnom za obavljanje dužnosti upravljanja i održavanja sustava odnosno odrađivanja sistemskih zadataka koji traže što direktniji pristup svim uređajima, ali i vrlo brz odziv.
Sâmo kernel može pokrenuti ili zaustaviti nîti kernela. S druge strane, proces na razini korisnika može se drugačije prerasporediti, a istodobno komunicirati s nîtima kernela na temelju raznih načina komunikacije; gore spomenutih, a vidljivih na slici 31. U svakom slučaju nîti na razini korisnika odnosno korisničkih (normalnih) aplikacija, moraju komunicirati s kernelom i nîtima kernela, kako bi na kraju koristili resurse sustava: CPU, memorija, disk, mreža. Možemo to reći i ovako: procesi i njihove nîti kernela odnosno procesi koji pokreću programski kôd unutar izoliranog adresnog prostora kernela ne ovise o nikakvom procesu odnosno programu koji se izvršava u drugom dijelu memorijskog prostora koji je rezerviran za korisničke programe ili aplikacije odnosno nîtima na razini korisnika. Dakle kernel programi i njihove nîti su prema logici rada poput *Unix*/*Linux* servisa, koji su stalno aktivni, ali na vrlo niskoj razini, te se nalaze u izoliranom prostoru kernela.
Naime kernel programi su obično programi zaduženi za rad s memorijom (memorijski menadžer), rad s procesima, za što je zadužen *proces/task scheduler*. On je zadužen i za druge sistemske zadatke, poput pristupa mreži, diskovnom sustavu i slično. Dakle nîti kernel programa su one koje se preko *proces/task/thread* menadžera šalju procesoru (CPU) na obradu.

Dodatno, sâm *proces/task/thread* menadžer je u mogućnosti razlikovati radi li se o kernel procesu s jednom nîti ili više nîti unutar jednog kernel procesa, kao i mijenjati prioritet odnosno vremenski okvir unutar kojeg se određena kernel nît može izvršavati, u slučajevima kada je to dinamički, u radu, potrebno promijeniti. S druge strane kada korisnički program (*user-space*) nešto treba odraditi, on preko sistemskih poziva, poziva kernel programe i/ili njegove programske nîti, da bi mogao koristiti resurse sustava (CPU, RAM, disk, mreža, …). Zapravo sâm kernel nije direktno svjestan programskih nîti unutar korisničkih programa (*user-space programa*) pa ih *task/proces/thread scheduler* obrađuje kao program s jednom programskom nîti.

Dakle ne dolazi do pozivanja nekog sistemskog poziva za aktivaciju ili pokretanja programske nîti unutar *user-space* programa. To se sve odrađuje korištenjem biblioteka koje se nalaze unutar programskog jezika u kojem je program razvijan, pa sve ovisi i o implementaciji unutar programskog jezika. Zbog toga se ovakvi programi s više programskih nîti od strane sustava izvršavaju kao da nemaju programske nîti (s točke kernela) pa kod dobrih implementacija nema (puno) prebacivanja konteksta ili je ono minimalno i brzo, naravno sve do isteka vremenskog okvira za izvršavanje tog konkretnog procesa (programa).

**Vratimo se nakratko na način razvoja i rada programa**
Programi se obično pohranjuju na disku u obliku koji se na našem računalu može izvršiti. Međutim, prije toga program se razvija u nekom programskom jeziku kao što su: **C**, **C++**, **JAVA**, **Fortran**, **Pascal** ili mnogi drugi, pomoću naredbi odnosno uputa koje uključuju logiku, manipulaciju podataka i uređaja, ponavljanje, grananje i interakciju s korisnikom. Krajnji rezultat je tekstualna datoteka s izvršnim kôdom u binarnom obliku (**1** i **0**), da bi se uopće pokrenuo na računalu.

Druga vrsta programa naziva se „tumačena“ ili *interpreterska*, a umjesto da se unaprijed napravi binarni izvršni programski kôd programa za pokretanje, tumači se u izvršnom kôdu programskog jezika u trenutku pokretanja.

Neki uobičajeni, *interpreterski* programski jezici su: **Python**, **PHP**, **JavaScript** i **Ruby**. Krajnji rezultat rada *interpreterskih* jezika je isti, međutim, kada se program pokrene, učitava se u memoriju u binarnom obliku.
Pošto procesor računala (CPU) razumije samo binarne upute, to je jedini oblik koji program treba imati u sâmom trenutku pokretanja. Unutar većine programski jezika implementirane su biblioteke za obradu i rad s nîtima (*threadovima*) koje sadrže programski kôd za stvaranje i uništavanje nîti, za prosljeđivanje poruka i podataka između nîti, za raspoređivanje izvršenja nîti te za spremanje i vraćanje konteksta (*context switching*) unutar programa (procesa).

Sâmo prebacivanje između nîti unutar procesa odrađuje se unutar procesa, upotrebom funkcionalnosti koje dobivamo specifično za svaki programski jezik, a unutar biblioteka za rad s nîtima, i naravno dobrom pisanju programskog kôda odnosno programa koji ih uopće i može koristiti.



Neki programski jezici imaju bolje, neki lošije, a neki gotovo nikakve mogućnosti izvršavanje više nîti unutar jednog programa (procesa). Neki od programskih jezika imaju i ograničenja u efektivnom broju nîti i njihovim efektivnim korištenjem, a sâmim time i indirektnim iskorištavanjem dostupnih CPU jezgri sustava. To znači kako se programi pisani u određenim programskim jezicima ne mogu dobro skalirati odnosno iskorištavati sve dostupne CPU jezgre, ali i druge resurse sustava.



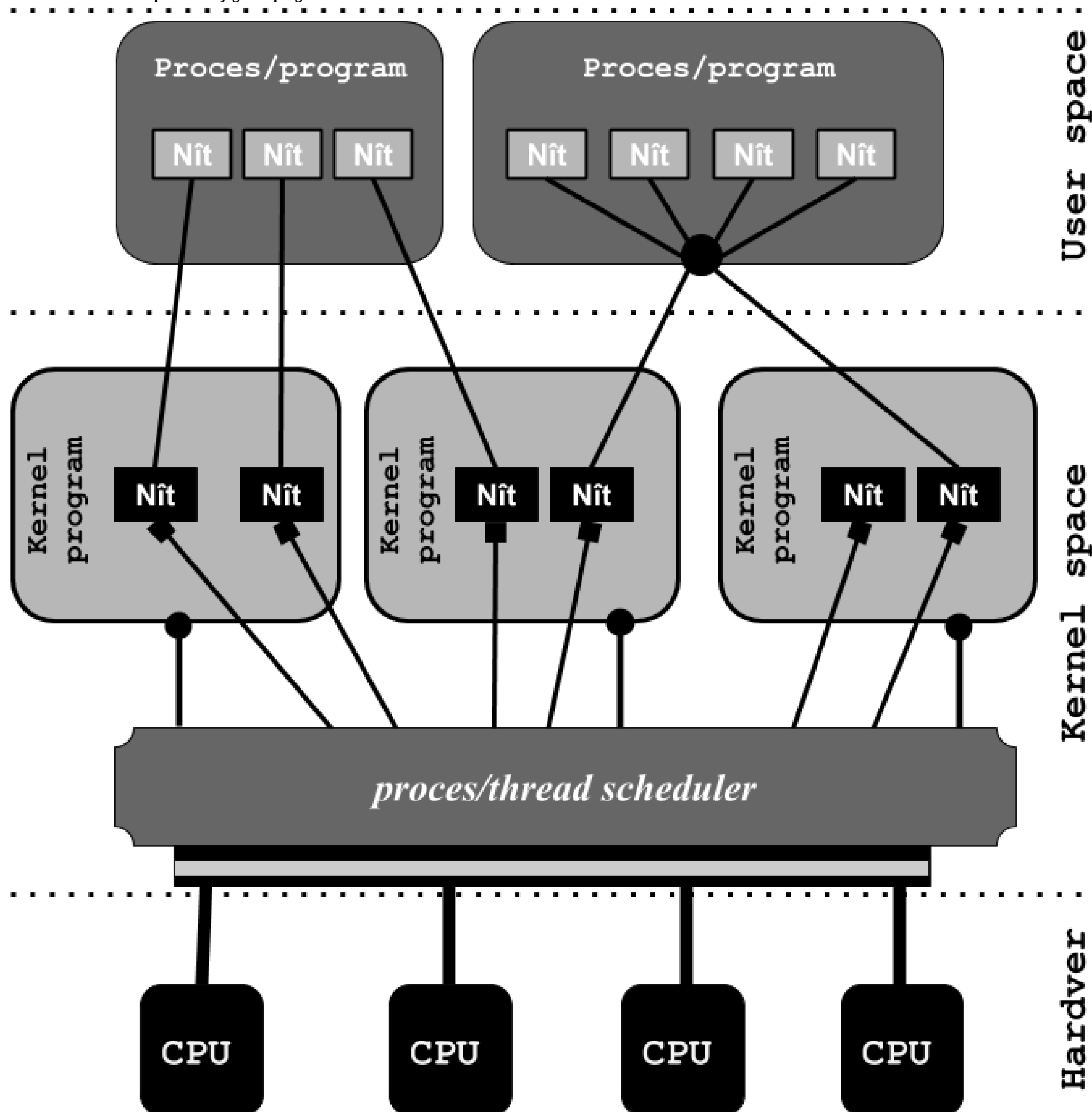
Vezano za rad programa odnosno procesa u memoriji pogledajte i poglavlje:
**12.4 Anatomija programa u memoriji**.

Naime u trenutku kada se *proces/task scheduler* prebaci na učitavanje i izvršavanje drugog procesa/programa, ponovno dolazi do prebacivanja konteksta (*context switch*). Problem nastupa kada jedna nît unutar programa zablokira ostale nîti zbog čekanja na neki događaj i/ili uređaj. Drugi problem korisničkih višenîtnih programa je taj što nemaju finu koordinaciju sa *proces/task/thread schedulerom* u slučajevima kada program ima puno nîti; jer će se on izvršavati kao da ih ima i vrlo malo.
Sljedeći slučaj je kada jedna nît treba više procesorskog vremena u odnosu na ostale nîti unutar procesa/programa.

U bilo kojem od tih slučajeva nema razlike; nîti će se uvijek izvršavati na isti način: bilo da se radi o programu sa 100 nîti ili sa sâmo jednom ili bez obzira ima li neka nît potrebu za povećavanjem prioriteta u odnosu na drugu.
Jedini način je rješavanje tog problema unutar sâmog programa/procesa odnosno programskog jezika u kojem je taj program napisan.

Pogledajmo potpuniji shematski prikaz komunikacije u kojoj korisnički program s lijeve strane ima tri programske nîti i spojen je na kernel nîti u modelu *jedan-na-jedan* (*One-to-one*). Dok su četiri nîti korisničkog programa s desne strane spojene na nîti kernel programa korištenjem modela: *više-na-više* (*Many-to-many*). Stoga pogledajmo sliku 32.

**Slika 32. Metode obrade procesa i njegovih programskih nîti.**




Vidljivo je i kako *proces/task scheduler* mehanizam u kernelu raspoređuje kernel nîti: (konektor ■) i/ili kernel programe (konektor •) na izvršavanje prema fizičkim (jezgrama) procesora (CPU).

Sada možemo sumirati prednost i mane obiju vrsta nîti (*threadova*):

**Nîti na razini kernela (kernel-space):**

- O nîtima na razini kernela u potpunosti se brine operativni sustav odnosno kernel komponenta zvana *proces/thread scheduler*. Prema tome sâm kernel se brine o cijelom životnom vijeku i radu nîti na ovoj razini.
- Postoji po jedan procesni kontrolni blok (engl. *Process control block* [**PCB**]) po kernel procesu, koji se brine o procesima.
- Postoji i po jedan nîtni kontrolni blok (engl. *Thread control block* [**TCB**]) po kernel nîti, koji se brine o nîtima.
- Na i s razine kernela (*kernel-space*) prosljeđuju se sistemski pozivi prema kojima komuniciraju nîti programa iz korisničke razine (*user-space*) odnosno *user-level nîti.*

***Prednosti:***

- Kernel ima potpunu kontrolu i nadzor nad svim nîtima kernela (i /ili kernel procesima).
- ***Task/proces scheduler*** može u bilo kojem trenutku dodijeliti više CPU vremena kernel procesu koji ima više nîti, a kojemu je potrebno više resursa odnosno smanjiti mu resurse, ako je to potrebno.
- Dobro je za aplikacije koje mogu često biti u stanju blokiranja odnosno čekanja ne neki događaj ili resurs jer *proces/thread scheduler* može prebaciti kontekst izvršavanja (*context switch*) na drugu nît unutar procesa i/ili na drugi proces.

***Mane:***

- Kernel ima potpunu kontrolu i nadzor nad svim kernel nîtima, što može bi i prednost i mana.
- Dodaje se još jedna razina kompleksnosti jer moraju postojati mehanizmi za rad s procesima (**PCB**), ali i mehanizmi za rad s nîtima procesa (**TCB**).
- Sporije izvršavanje jer se za komunikaciju moraju koristiti sistemski pozivi tijekom prebacivanja ili komunikacije između nîti.
- Zbog svega navedenog nîti na razini kernela su sporije i neučinkovitije od nîti na korisničkoj razini, čak do desetak ili stotinu puta.

Kako bi se izbjeglo često prebacivanje konteksta (*context switching*), a koje bi se često događalo u kernel procesima koji imaju više nîti, češće se pribjegava razvoju kernel programa koji imaju manje programskih nîti. Skaliranje se rješava na način da se za veliku većinu kernel programa, koji zahtijevaju veliku brzinu rada i što direktniji pristup hardveru, pokreće po jedan kernel program (proces), na jednoj jezgri procesora. Pa tako imamo obično onoliko kernel procesa za određenu namjenu, koliko imamo CPU jezgri.

Pogledajmo neke od njih uz kratki opis:

- `ksoftirq`***** je zadužen za sve softverske signale prekida. Primjerice između procesa/programa, ali i upravljačkih programa mrežnih kartica, disk kontrolera i drugih komponenti koje ih mogu koristiti za ubrzanje rada.
- `aio`***** je zadužen za pristup ulazno/izlaznom (I/O) pod sustavu preko asinkronih operacija.
- `migration`***** je zadužen za preraspodjeljivanje (distribuciju) opterećenja sustava (procesa/aplikacija) na više CPU jezgri.

Ako samo izlistamo navedene procese na računalu koje ima dvije CPU jezgre vidjet ćemo pokrenuta po dva (2) pokrenuta kernel procesa za svaku funkcionalnost, odnosno namjenu (fokusirani smo na gore navedena tri procesa*):

```
ps -ef | egrep „aio|ksoft|migration" | grep -v egrep

root          3      2  0 12:52 ?        00:00:00 [migration/0]
root          7      2  0 12:52 ?        00:00:00 [migration/1]
root          4      2  0 12:52 ?        00:00:00 [ksoftirqd/0]
root          9      2  0 12:52 ?        00:00:00 [ksoftirqd/1]
root         49      2  0 12:52 ?        00:00:00 [aio/0]
root         50      2  0 12:52 ?        00:00:00 [aio/1]
```

***Nîti* na razini korisnika (*user-space*)**

- Nîti unutar programa/procesa se odrađuju unutar procesa, u korisničkom prostoru (user-space) aplikacije, bez posredovanja kernela. To radi tako da se nîti unutar programa u potpunosti odrađuju upotrebom tzv. *Runtime biblioteka* unutar programskog jezika u kojem je program razvijan (pisan).
- U idealnim uvjetima operacije s nîtima su brze poput pozivanja funkcija unutar istog programa.
- Kernel nema nikakve spoznaje o tome ima li proces/program na ovoj razini (*user-space*) više nîti ili nema.

**Prednosti**:

- Moguće je implementirati nîti unutar programa/procesa i na operativnom sustavu koji nema podršku za višenîtno izvršavanje programa/procesa i samim time ne zahtjeva nikakve promjene na strani OS-a.
- Sve potrebne komponente programa poput: PC, registara, stôga (*stack*) i kontrole izvršavanja nîti, koriste se unutar korisničkog (*user-space*) adresnog prostora.
- Jednostavniji menadžment: od kreiranja, sinkroniziranja i prebacivanja između nîti se događa također unutar „*user-space*" prostora, ponovno bez intervencije kernela.
- Brzo i efikasnije prebacivanje između nîti programa.

**Mane**:

- Nije najbolje rješenje za sve slučajeve.
- Ne postoji koordinacija između nîti unutar programa na „*user-level*" razini s *proces/task/thread managerom* na razini kernela te stoga operativni sustav može donijeti loše odluke kod:
    - Prioriteta/procesorskog vremena za procese s *nîtima* koje su u stanju čekanja.
    - Cijeli proces će biti blokiran, ako je i jedna od *nîti* unutar procesa u stanju blokiranja (čekanja) čak i ako su druge *nîti* slobodne.
    - Za cijeli proces se dodjeljuje jedan vremenski okvir za izvršavanje, bez obzira koliko *nîti* taj proces ima.
- Ponovnim prebacivanjem na proces koji je trenutno u stanju pauziranja, kada se prebaciti kontekst izvršavanja (*context switch*) na novi proces, aktivira se mehanizam zaključavanja (engl. *Lock*).
- U navedenim, ali i drugim sličnim situacijama, potrebna je komunikacije između procesa i *kernel-level nîti*/procesa kako bi se navedeni problemi izbjegli, što uvodi dodatnu kompleksnost i sporiji rad.



Važno je znati kako na razini cijelog operativnog sustava Linux postoji definiran maksimalan broj programskih nîti koje se uopće mogu koristiti, a koji je definiran u **sysctl,** ali i u sistemskoj varijabli: `threads-max`.

Vrijednost navedene varijable koje definira maksimalan broj programskih niti na sustavu možemo vidjeti na sljedeći način:

```
cat /proc/sys/kernel/threads-max
449135
```

To znači kako je *proces/program/task scheduler* za sve pokrenute programe (procese) u mogućnosti koristiti sveukupno maksimalno toliko (`threads-max`) nîti. Ova varijabla je dostupna i kao ***sysctl*** varijabla: `kernel.threads-max`.

S naredbom `ps` možemo za željeni proces vidjeti i koliko programskih nîti koristi, pomoću prekidača: `-T`.
Pogledajmo koliko programskih nîti primjerice koristi program odnosno servis `rsyslog`, čiji trenutni ***PID*** broj je `2208`:

```
ps -af -T -p 2208

UID          PID  SPID   PPID  C STIME TTY          TIME CMD
root        2208  2208      1  0 Sep10 ?        00:00:00 /usr/sbin/rsyslogd -n
root        2208  2398      1  0 Sep10 ?        00:02:31 /usr/sbin/rsyslogd -n
root        2208  2399      1  0 Sep10 ?        00:00:14 /usr/sbin/rsyslogd -n
```

Vidimo kako ovaj program trenutno koristi tri (3) nîti, od kojih svaka ima svoj identifikator, vidljiv u stupcu (`SPID`).

Odnosno, isto možemo vidjeti i na slijedeći način, pomoću prekidača (`-L`):

```
ps -af -L -p 2208

UID          PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
root        2208     1  2208  0    3 Sep10 ?        00:00:00 /usr/sbin/rsyslogd -n
root        2208     1  2398  0    3 Sep10 ?        00:02:31 /usr/sbin/rsyslogd -n
root        2208     1  2399  0    3 Sep10 ?        00:00:14 /usr/sbin/rsyslogd -n
root       25400 35969 25400  0    1 08:31 pts/1    00:00:00 ps -af -L -p 2208
```

Ovdje je u stupcu (`LWP`) vidljiv identifikator svake nîti (*Thread ID*), a u stupcu (`NLWP`) vidljiv je broj nîti (*Number of Light Weight Processes*).

Ako sâmo želimo prebrojati programske nîti koje naš program koristi, to možemo napraviti sa:

```
ps -T -p 2208 | wc -l
```

I dobit ćemo njihov broj (primjerice): `3`. Sve postavke programskih nîti se pohranjuju u datoteke unutar strukture direktorija prema ***PID*** broju procesa.

Primjerice za ***PID*** broj `2208`, bi to bio vršni direktorij: `/proc/2208/task/` unutar kojeg se nalazi nova struktura pôddirektorija za svaki ***TID*** (*Thread ID*) odnosno ***ID*** od svake zasebne programske nîti našeg procesa. Poput primjerice: `/proc/2208/task/2308/`, `/proc/2208/task/2309/`, a unutar njih su sve datoteke s postavkama.



Veliki broj prebacivanja između programskih nîti unutar programa nije uvijek poželjno i najbolje rješenje jer se u određenim slučajevima previše resursa troši na prebacivanje između nîti unutar procesa (programa) nego na stvarnu obradu podataka koja bi se eventualno mogla odraditi unutar jedne (ili manjeg broja) programske nîti.



Vezano za mjerenja performansi programa uzrokovanih prebacivanjem s programske nîti na nît ili s procesa na proces, pogledajte poglavlje:

**15.1.1. perf stat** i to statistiku: `context-switches`, ali za vaš konkretan program.

**Izvori informacija: (355),(356),(357),(358),(359),(360),(361),(362),(K-5), `man 5 procfs`,`man 7 pthreads`, `man 7 signal`.**

# 9.4. Komunikacija između procesa

***Slijedi napredno poglavlje (9.4.x)!***
Komunikacija između programa odnosno procesa, znana je i pod nazivom *Interprocess Communication*, odnosno **IPC** komunikacija. Za komunikaciju između procesa postoje točno definirane metode komunikacije odnosno standardi, koji nam nude ovu vrlo važnu mogućnost svakog operativnog sustava. Naime još u vrijeme ranog zavoja *Unix* operativnih sustava početkom 1980-tih godina, definirani su neki od mehanizama komunikacije između procesa.

Već su u operativnom sustavu ***Unix System V*** uvedena tri ***IPC*** mehanizma, koji se glavnom potpomažu, a to su:

- ***Spremnik za poruke (engl. Message queues)*** - U ovom principu, svaki program/proces može poslati poruku u ovaj spremnik za poruke, a one se mogu dohvatiti na više načina. Svaki nîz poruka (spremnika) je jedinstveno identificiran s ***IPC*** identifikatorom i namijenjen je komunikaciji između definiranih programa odnosno procesa.
- ***Semafori (engl. Semaphores)*** – Služe za označavanje stanja pristupa (komunikacijskim) resursima. Naime oni se koriste za kontrolu pristupa dijeljenom resursu, između više programa odnosno procesa. Obično se koriste kao mehanizam privremenog zaključavanja resursa od strane onog programa/procesa koji trenutno radi neke operacije nad tim resursom kako neki drugi program/proces ne bi istovremeno radio promijene nad istim (komunikacijskim) resursom, što bi dovelo do nekonzistentnosti podataka.
- ***Dijeljena memorija (engl. Shared memory)*** odnosno sustav dijeljenja memorije između procesa koji komuniciraju. Kod ove metode određene regije memorije za komunikaciju se dijele između više programa odnosno procesa, kako bi se informacije direktno mogle zapisivati ili čitati iz tih regija memorije od strane više programa/procesa u komunikaciji. Dakle, svaki program/proces može za komunikaciju s drugim programima/procesima kreirati posebnu regiju memorije te zapisivati ili čitati iz nje, ili je na kraju i obrisati.

S druge strane, ***BSD Unix*** (*Berkeley Software Distribution*) je uveo takozvane ***Unix sockete*** kao osnovni mehanizam za razmjenu poruka između procesa/programa. Linux koristi i metode koje su nasljednice ***Unix System V*** i ***BSD Unix*a**.
Mada se u velikoj većini slučajeva koriste metode nasljednice iz ***BSD Unix*a**, poput:

- ***Unix*** *(domain)* ***socketa.***
- ***Unix pipe****a* odnosno cjevovoda.
- Dok se za mrežni dio komunikacije koriste ***network socketi.***

**Signali**
Dodatno, procesi mogu komunicirati među sobom i slanjem određenih sistemskih signala o kojima smo već govorili.
Međutim, ne može svaki program/proces, poslati bilo kojem drugom programu/procesu sve signale jer nema prava na to.
Samo kernel i *superuser* (*root* korisnik) imaju pravo svim programima odnosno procesima poslati apsolutne sve dostupne signale.
Dalje u tekstu upoznat ćemo se sa svim gore navedenim metodama komunikacije između programa odnosno procesa.

**Izvori informacija: (238),(239),(240),(241),(K-5), `man 7 pipe`.**

## 9.4.1. Unix Pipes

Vratimo se na komunikaciju između programa odnosno procesa. Jedna od metoda komunikacije između procesa je takozvana „*pipe*" funkcionalnost. Pod ovim pojmom se zapravo kriju dvije slične funkcionalnosti. Prva funkcionalnost je upotreba klasičnog *unix/linux pipea* odnosno cjevovoda koji simbolizira znak `|`. U načelu ova funkcionalnost nam daje mogućnost preusmjeravanja izlaza podataka iz jednog programa na ulaznu obradu drugom programu.
Drugi program je opet u mogućnosti, nakon što obradi podatke, proslijediti ih na ulaz na obradu trećem programu i tako dalje.
Prema logici rada ovdje imamo sljedeće stanjem kako je vidljivo na slici 33.

**Slika 33. Veza između procesa pomoću Unix pipe-a**






Prvo se podsjetite *pipea* ( | ) i redirekcije te pogledajte poglavlje:
**5.10 Preusmjeravanje (*redirekcija*) i Pipe.**

Već smo rekli kako se standardno za svaki program kreiraju tri osnovna *file deskriptora*:

- 0 — ovo je *file deskriptor* koji predstavlja standardni ulaz u program (***stdin***).
- 1 — ovo je *file deskriptor* koji predstavlja standardni izlaz iz programa (***stdout***).
- 2 — ovo je *file deskriptor* koji predstavlja standardnu grešku odnosno poruke o greškama iz programa (***stderror***).

Za svaki pokrenuti program odnosno svaki aktivni proces, tijekom njegovog pokretanja, sustav kreira upravo ova tri *file deskriptora*, a tek potom i ostale. U normalnom radu, sve što ulazi u program dolazi na *file deskriptor* **0**, kao i sve što izlazi iz njega, izlazi na *file deskriptor* **1**. U normalnoj situaciji to bi izgledalo ovako (ako gledamo s razine datoteka [*file deskriptora*]):

```
lr-x------ 1 root root 64 Feb 28 19:24 0 -> /dev/pts/0
l-wx------ 1 root root 64 Feb 28 19:24 1 -> /dev/pts/0
lrwx------ 1 root root 64 Feb 28 19:24 2 -> /dev/pts/0
```

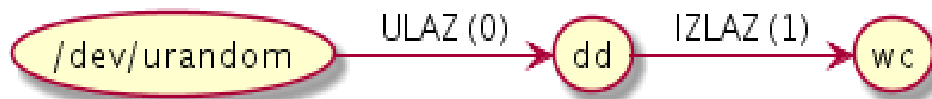
Dakle sva tri *file deskriptora* bi koristila virtualni terminal (pts), iz kojeg pišemo i pokrećemo sve naredbe i programe.
Ako pak koristimo *unix* odnosno *linux pipe*, to se malo mijenja, jer se tada poveznice na *file deskriptore* **0** i **1** mijenjaju.
Dakle logika nam govori kako se izlazni *file deskriptor* prvog programa spaja na ulazni *file deskriptor* drugog programa, što je u potpunosti točno.

Pogledajmo sljedeće dvije naredbe povezane s *linux pipeom* ( | ), poput naredbi dd i wc:

```
dd if=/dev/urandom | wc -l
```

Ove dvije naredbe povezane s linux *pipe*om ( | ), možemo promatrati i ovako (vidljivo na slici 34.).

**Slika 34. Unix pipe funkcionalnost detaljnije.**




Vidimo kako standardni izlaz naredbe dd preusmjeravamo u naredbu wc, odnosno na njen standardni ulaz.

## Dokaz teorije

Nakon što smo pokrenuli gornju naredbu, iz drugog terminala smo saznali kako je njen ***PID*** broj 21069, točnije napisali smo pidof dd, kako bismo dobili njen ***PID*** broj. Sada pogledajmo *file deskriptore* za taj program, kako bismo vidjeli poveznicu među njima.

Izlistajmo direktorij u kojem se vide *file deskriptori* za procese, po svakom pojedinom ***PID*** broju procesa:

```
ls -al /proc/21069/fd/

lr-x------ 1 root root 64 Feb 28 19:24 0 -> /dev/urandom
l-wx------ 1 root root 64 Feb 28 19:24 1 -> pipe:[47793]
lrwx------ 1 root root 64 Feb 28 19:24 2 -> /dev/pts/0
```

U ovom izlistanju vidimo slijedeće:

- *File deskriptor* **0** (standardni ulaz) u naredbu dd je postala datoteka /dev/urandom (koja generira slučajne brojeve/slova) jer smo tako i naložili naredbi dd, pomoću prekidača if (koji označava *input file* tj. ulaz u dd).
- Za standardni izlaz smo sada dobili novi *unix FIFO pipe*, ali njega ćemo objasnit samo kratko, jer ćemo o njemu govoriti ubrzo u drugoj cjelini. Pronađimo na što se taj *unix FIFO pipe* koji je vezan za naš *file deskriptor* **1**, veže. Poveznica je konkretno *inode* broj 47793.

Provjeru ove poveznice ćemo napraviti pomoću naredbe lsof na sljedeći način:

```
lsof -n | grep 47793
```

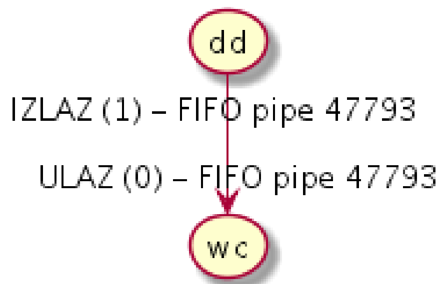
| COMMAND | PID | TID | USER | FD | TYPE | DEVICE | SIZE/OFF | NODE | NAME |
|---|---|---|---|---|---|---|---|---|---|
| dd | 21309 | | root | 1w | FIFO | 0,8 | 0t0 | 47793 | pipe |
| wc | 21310 | | root | 0r | FIFO | 0,8 | 0t0 | 47793 | pipe |

Vidimo kako je *unix FIFO pipe* poveznica prema drugoj naredbi u nizu: wc, prema kojoj smo koristili *linux pipe* (|).

Ako gledamo standardni izlaz (1) naredbe dd ovdje vidimo 1w što znači kako je *file descriptor* 1 u stanju „*write*" *(zapisivanje)* jer on zapisuje i šalje podatke.

Za drugu stranu, odnosno program `wc` (u drugom redu ispisa) vidimo kako je spojen na *unix FIFO pipe* (iz prve naredbe `dd`) i to na svoj *file descriptor* `0` koji je njen standardni ulaz, s kojeg ona čita podatke pa vidimo: `0r`, gdje `r` označava kako je on u stanju čitanja odnosno u stanju „*read*“. To možemo prikazati i ovako, vidljivo na slici 35.

**Slika 35. FIFO pipe funkcionalnost.**




**Drugi *pipe* je zapravo *unix FIFO pipe* funkcionalnost**

S *unix FIFO pipe* smo se već upoznali, jer smo vidjeli kako i *linux pipe* (simbol `|`) isto u pozadini koristi *unix FIFO pipe* i preusmjeravanje standardnih ulaza (`0`) i izlaza (`1`) pa pogledajmo neke detalje o njemu. Ova funkcionalnost radi tako da se u slučaju povezivanja dva programa, pomoću *pipe*a, kreiraju dvije *file* strukture podataka, koje zapravo pokazuju na isti *inode* unos na razini virtualnog datotečnog sustava (***VFS***). Oni u konačnici pokazuju na istu regiju memorije, odnosno istu stranicu memorije (*memory page*).
To funkcionira tako da prvi program u svojoj *file* strukturi *pipea* sadrži pokazivač na rutinu za zapisivanje podataka (*write*), koje šalje drugom programu, a drugi program koji preko *pipe*a preuzima izlazne podatke prvog programa kao svoje ulazne, ima pokazivač na rutinu za čitanje tih podataka (*read*). Dakle prvi program zapisuje u *pipe*, a drugi čita iz njega.
Ove datotečne (*file)* strukture su zapravo posebni *file desktriptori* odnosno posebne vrste datoteka, koje se kreiraju u virtualnom datotečnom sustavu (***VFS***).



Za detalje o ***VFS***-u pogledajte poglavlje:
**14.1. Diskovni ulazno/izlazni sustav (*I/O*).**

Ono što se događa je zapravo kreiranje *shared memory* regije odnosno bloka dijeljene memorije između ova dva programa, u koju prvi program upisuje podatke, a drugi program iz nje čita ono što je prvi program upisao. Ovdje se koristi mehanizam semafora, jer kada prvi program započne proces snimanja podataka u *file deskriptor* koji predstavlja dijeljenu regiju memorije između dva procesa, on uključuje semafor s kojim se stavlja oznaka, kako je sada ta regija memorije zaključana od procesa koji kreće s kopiranjem podataka u nju. Dakle podaci se prvo kopiraju iz adresnog prostora programa koji radi kopiranje, u adresni prostor te regije dijeljene memorije odnosno u konkretni *file descriptor*.
S druge strane program koji treba čitati podatke iz tog *file deskriptora*, čeka da se proces zapisivanja završi, te tada može krenuti s čitanjem podataka. Proces koji čita, prije nego krene s čitanjem, prvo zaključava operacije snimanja, sve dok ne pročita cijeli blok podataka, te ih po završetku otključava, kako bi po potrebi prvi proces mogao nastaviti s novim nîzom zapisivanja, s već opisanom procedurom.
Ova vrsta *file deskriptora* se zove ***unix FIFO pipe file deskriptor***, a vidljiv je na sljedeći način (u primjerima dolje).

Iako ovi mehanizmi izgledaju kompleksni i usporeni, cijela ova metoda je vrlo pouzdana te nevjerojatno brza i optimizirana.
Svaki je proces s pripadajućim ***PID*** brojem, uz sve resurse koje koristi, vidljiv unutar posebnog */proc/* direktorija, točnije */proc/**PID**/*, gdje je *PID*, ***PID*** broj konkretnog procesa.
Nas zanimaju *file deskriptori* našeg procesa/programa, a oni se nalaze u strukturi direktorija: */proc/**PID**/fd/*.

Pogledajmo kako to izgleda za jedan klasičan program, koji prihvaća ulaz i izlaz podataka. Radi se o programu `mc` (*Midnight Commander*), čiji trenutni ***PID*** broj je `25632`. Pogledajmo i što vidimo ručnim ispisom datoteka unutar zadanog direktorija:

```
ls -al /proc/25632/fd/

lrwx------ 1 root root 64 Feb 20 12:09 0 -> /dev/pts/0
lrwx------ 1 root root 64 Feb 20 12:09 1 -> /dev/pts/0
lrwx------ 1 root root 64 Feb 20 12:09 2 -> /dev/pts/0
lrwx------ 1 root root 64 Feb 20 12:09 3 -> /dev/tty
lrwx------ 1 root root 64 Feb 20 12:09 4 -> /dev/ptmx
lrwx------ 1 root root 64 Feb 20 12:09 5 -> /dev/pts/1
lr-x------ 1 root root 64 Feb 20 12:09 6 -> pipe:[187719912]
l-wx------ 1 root root 64 Feb 20 12:09 7 -> pipe:[187719912]
```

Na kraju ispisa vidimo dva *file deskriptora*, koji predstavljaju *unix FIFO pipe* (`pipe:[187719912]`) *i* (`pipe:[187719912]`).

Što još ovdje vidimo:

- ***File deskriptor*** broj 6:
  - Ima *READ* ovlasti (lijevi dio ovlasti (*r*)). Dakle on je za čitanje podataka.
  - U nazivu linka na ime datoteke (6) stoji *pipe:[187719912]*, što znači kako je on definitivno *pipe file deskriptor* te kako je njegov *inode* broj u virtualnom datotečnom sustavu (***VFS***): *187719912*
- ***File deskriptor*** broj 7:
  - Ima *WRITE* ovlasti (lijevi dio ovlasti (*w*)). Dakle on je zadužen za zapisivanje podataka.
  - U nazivu linka na ime datoteke (7) stoji *pipe:[187719912]*, što znači kako je on definitivno *unix FIFO pipe file deskriptor* te kako je njegov *inode* broj u virtualnom datotečnom sustavu (***VFS***): *187719912*, koji je identičan sa prvim *pipe file deskriptorom*, što indicira kako su povezani.

Kako bismo saznali s kojim je još procesom povezan ovaj *pipe file deskriptor*, moramo pronaći koji još program/proces koristi isti *pipe file deskriptor*, prema *inode* broju: 187719912. Pogledajmo kako ćemo to saznati pomoću naredbe lsof:

```
lsof | grep 187719912

mc          25632         root     6r      FIFO          0,8          0t0   187719912 pipe
mc          25632         root     7w      FIFO          0,8          0t0   187719912 pipe
bash        25634         root     7w      FIFO          0,8          0t0   187719912 pipe
```

Vidimo kako je osim našeg programa mc, s njime povezana i bash *ljuska*, kroz isti *pipe file deskriptor*. To znači kako od *bash ljuske* možemo primati podatke, jer je *bash* u stanju zapisivanja (*write), vidljivo kao* 7w, odnosno u stanju je komunicirati s njime (s *bash ljuskom*). Istovremeno vidimo kako imamo i jednu mc instancu koja je isto u stanju zapisivanja (*write)* (7w).
Navedena instanca je zapravo funkcija programa koja može slati poruke nekoj drugoj funkciji istog programa, a to je ona koja čita (*read*) odnosno vidljiva je kao 6r.

*Ovdje smo naučili kako programi/procesi među sobom mogu komunicirati pomoću pipe file deskriptora.*

Vratimo se samo korak nazad, da bismo vidjeli kako nam izgleda ispis i s naredbom lsof, za naš proces ***PID***: 25632:

```
lsof -p 25632 | egrep -vi „REG|DIR|CHR"

COMMAND    PID  USER    FD   TYPE  DEVICE  SIZE/OFF        NODE        NAME
mc       25632  root     6r  FIFO     0,8         0t0      187719912   pipe
mc       25632  root     7w  FIFO     0,8         0t0      187719912   pipe
```

S naredbom egrep smo filtrirali sve nepotrebno, pa vidimo isto, kao s ručnom metodom, samo malo drugačije formatirano:

- FD broj 6 je u stanju čitanja (*r*), njegov tip je *FIFO*, vidimo *inode* broj, pod stupcem NODE i ime (NAME) je *pipe* što označava kako je on *unix pipe file deskriptor*.
- FD broj 7 je u stanju zapisivanja (*w*), njegov tip je *FIFO*, vidimo *inode* broj, pod stupcem NODE i ime (NAME) je *pipe* što označava kako je on *unix pipe file deskriptor*.

*Pipe file deskriptori* se koriste za komunikaciju između programa/procesa, ali i za komunikaciju između *roditeljskog* i procesa *djeteta* odnosno onog procesa kojeg pokreće *roditeljski* proces.

Pogledajmo još jedan jednostavniji primjer upotrebe *unix FIFO file descriptora* preko unix/linux *pipea* ( | ). Pokrenut ćemo naredbu top i reći joj da konstantno ispisuje statistike na standardni izlaz (1) odnosno na ekran, te ćemo taj izlaz preusmjeriti u drugu naredbu, pomoću *linux/unix pipea* (|).
Druga naredba će biti grep, s kojom ćemo filtrirati ispis i tražiti u njemu sâmo jednu ključnu riječ: *java*.

U prvom terminalu pokrećemo:

```
top -b -n 20 | grep java
```

Dok se ova naredba izvršava, brzo u drugom terminalu potražimo njen ***PID*** broj:

```
pidof top

15026
```

Saznali smo da je ***PID*** broj naredbe iz prvog terminala broj 15026.

Sada pogledajmo za taj ***PID*** broj (od traženog programa), koje sve *file deskriptore* on ima otvorene:

```
ls -al /proc/15026/fd

lrwx------ 1 root root 64 Mar  1 07:57 0 -> /dev/pts/1
l-wx------ 1 root root 64 Mar  1 07:57 1 -> pipe:[117898256]
l-wx------ 1 root root 64 Mar  1 07:57 2 -> /dev/null
lrwx------ 1 root root 64 Mar  1 07:57 3 -> /dev/pts/1
lr-x------ 1 root root 64 Mar  1 07:57 4 -> /proc/uptime
lr-x------ 1 root root 64 Mar  1 07:57 5 -> /proc/meminfo
lr-x------ 1 root root 64 Mar  1 07:57 6 -> /proc/loadavg
lr-x------ 1 root root 64 Mar  1 07:57 7 -> /proc/stat
```

U prethodnom ispisu vidimo kako je *file deskriptor* `1` (standardni izlaz), naredbe `top` (***PID***:`15026`) sada postao *unix FIFO pipe*, broja: `117898256`. To znači kako se sav izlaz naše naredbe `top` (***PID***:`15026`), šalje na *unix FIFO pipe* broja *`117898256`*.

Sada ćemo saznati s kim on komunicira i kako:

```
lsof | grep 117898256

COMMAND     PID   TID    USER    FD     TYPE     DEVICE  SIZE/OFF  NODE         NAME
top         15026        root    1w     FIFO      0,8        0t0   117898256    pipe
grep        15027        root    0r     FIFO      0,8        0t0   117898256    pipe
```

Ovdje je sve jasno:

- Naredba `top` snima i šalje podatke: njen *FD* broj `1` (standardni izlaz): je u stanju *WRITE* `1w` (zapisuje).
- Naredba `grep` prima podatke: njen *FD* broj `0` (standardni ulaz): je u stanju *READ* `0r` (čita).

I vidljivo je kako komunikacija između ove dvije naredbe ide preko *unix FIFO pipe file deskriptora*, koji identificira *inode* broj: *`117898256`*.

## Što se događa kod preusmjeravanja (*redirekcije*)?

U slučajevima kada radimo preusmjeravanje na razini *file deskriptora* se također događa zamjena *file deskriptora*.
Ponovimo prijašnji primjer, ali malo modificiran. Naime, sada ćemo izlaz iz naredbe: `top -b -n 20` preusmjeriti u datoteku, odnosno napravit ćemo *redirekciju* u datoteku: `/root/test/file.txt`.

Preusmjeravanje odnosno *redirekcija* se u Linuxu radi sa posebnim znakom `>`
Sada napravimo preusmjeravanje ispisa naredbe `top` u datoteku: `/root/test/file.txt`

```
top -b -n 20 > /root/test/file.txt
```

I ponovno ćemo u drugom terminalu doznati koji ***PID*** broj je naša `top` naredba dobila. U našem slučaju je to: `11339`.

Sada pogledajmo što se dogodio s *file deskriptorima* (skratili smo ispis samo na one *file deskriptore* koji nas zanimaju):

```
ls -al /proc/11339/fd

lrwx------ 1 root root 64 Mar  1 10:29 0 -> /dev/pts/1
l-wx------ 1 root root 64 Mar  1 10:29 1 -> /root/test/file.txt
l-wx------ 1 root root 64 Mar  1 10:29 2 -> /dev/null
```

## Što se ovdje dogodilo?

Dogodilo se to da je *file deskriptor* broj `1`, koji predstavlja standardni izlaz, sada preusmjeren u datoteku: `/root/test/file.txt`. To znači kako će sve što naredba: `top -b -n 20` ispiše, biti zapisano u datoteku: `/root/test/file.txt`.
Upravo to smo i htjeli.

Dakle preusmjeravanjem iz neke naredbe/programa, se mijenja *file deskriptor* broj `1`, što predstavlja standardni izlaz te naredbe, koji tada završava tamo gdje smo mu i naznačili, konkretno u navedenu datoteku što je vidljivo kao:
**`1`** `-> /root/test/file.txt`.



Za druge primjere upotrebe naredbe `lsof`, pogledajte i poglavlja:
**4.5.5.1. File deskriptori i naredba *lsof*.**
**9.4.3. Slijedeći koraci: *Unix* i *network socketi*.**
**9.4.3.1. Unix socketi.**
**25.7.6. Naredba *list open files* (*lsof*).**

**Izvor informacija: (676), (K-5),** **`man bash`**, **`man lsof`**, **`man 7 pipe`**, **`man 7 inode`**.

## 9.4.2. Named pipe

Prije nego krenemo dalje, upoznat ćemo se i s jednom posebnom vrstom datoteke, koja se zove ***named pipe***.

*Named pipe* ponaša se slično kao i obični Unix/Linux *pipe* uz nekoliko razlika:

- *Named pipe* postoji kao posebni uređaj (device), odnosno posebna datoteka u datotečnom sustavu. Naime, iako je *named pipe* dio datotečnog sustava, on se ponaša kao kao komunikacijski kanal.
- Procesi različitih roditelja mogu dijeliti podatke kroz *named pipe* odnosno *named pipe* omogućava *unidirekcionalni* tok podataka između procesa. To znači da podaci idu u jednom smeru - od jednog procesa do drugog.
- Kada su sve ulazno/izlazne (I/O) operacije završile kroz *named pipe*, on i dalje ostaje na datotečnom sustavu kao datoteka za dalju odnosno ponovnu upotrebu prema potrebi.

Nemojte pomiješati standardnu ***pipe*** funkcionalnost (|) o kojoj smo prije govorili i ***named pipe***, jer se radi o prilično različitim funkcionalnostima. Kroz primjere će vam upotreba *named pipe-a* biti jasnija.
Za kreiranje *name pipe* datoteke se koristi naredba `mkfifo`.

**Primjeri**
Kreirajmo *named pipe* datoteke imena `my_pipe` u našem trenutnom direktoriju: `/root/` pomoću naredbe `mkfifo`:

```
mkfifo my_pipe
```

Datoteka *`my_pipe`* se kreira u trenutnom direktoriju (mapi) iz koje ste pozvali naredbu: `mkfifo my_pipe`.

Pogledajmo ovu datoteku, pri tom smo skratili ispis samo na nju, pomoću naredbe:

```
ls -al | grep my_pipe

prw-r—r--   1 root root        0 May 29 10:53 my_pipe
```

U njenim ovlastima vidimo kako je prvo slovo p, što znači kako se radi o *named pipe* datoteci. Nakon što smo kreirali *named pipe* datoteku, napravit ćemo dvostruko preusmjeravanje (< i >) s kojim ćemo komprimirati sve što uđe u naš *named pipe*: `my_pipe` i stvoriti komprimiranu datoteku imena: `komprimirana.datoteka.gz`.
Na kraju ovog nîza naredbi koje se preusmjeravaju jedna u drugu smo ostavili znak **&**, koji znači kako će ovaj nîz naredbi nastaviti raditi u pozadini.

Dakle ovdje smo naredbi `gzip` rekli da radi komprimiranje podataka, a naš *named pipe*: `my_pipe` prosljeđuje podatke s jedne strane od `gzip`–a, a s druge strane prema datoteci koja se kreira: `komprimirana.datoteka.gz`.

Pogledajmo kako smo to izveli:

```
gzip -9 -c < my_pipe > komprimirana.datoteka.gz &
```

Sada ćemo poslati neku svoju datoteku; primjerice: `datoteka.txt` na komprimiranje, upotrebom *named pipe* datoteke:

```
cat datoteka.txt > my_pipe
```

Ovdje smo napravili sljedeće:

- Otvaramo datoteku imena: `datoteka.txt` s naredbom `cat` u našu *named pipe* datoteku: `my_pipe`.
- Naša *named pipe* datoteka: `my_pipe` sadržaj naše datoteke: `datoteka.txt` s jedne strane prosljeđuje naredbi: `gzip -9 -c`, koji ju komprimira, a onda `my_pipe` rezultat obrade `gzip`-a prosljeđuje u novu (konačnu) datoteku imena: `komprimirana.datoteka.gz`

Brisanje *named pipe* datoteke je isto kao i za bilo koju drugu datoteku. Pogledajmo kako ju obrisati pomoću naredbe `rm`:

```
rm my_pipe
```

Kao što smo spominjali i prije, sada postaje jasnije kako izjava "*Everything is a file*" ima sve više smisla.

**Izvori informacija: (104), (104.1),(K-5), `man mkfifo`,`man 7 fifo`.**

### 9.4.3. Slijedeći koraci: *Unix* i *network socketi*

Sljedeće dvije metode, također se koriste za komunikaciju, ali malo drugačiju. U ovoj cjelini ćemo govorit o:

- ***Unix socketima*** — koriste se za komunikaciju između programa/procesa (***IPC***).
- ***Network socketima*** — koriste se za mrežnu komunikaciju: između programa, preko mreže.

#### 9.4.3.1. Unix socketi

*Unix socketi* koriste se za komunikaciju između programa odnosno procesa na istom računalu.
Poznati su i pod nazivom ***Unix domain socketi***. Komunikacija pomoću njih može biti pomoću:

- ***DATAGRAM*** *socketa*: oni su manje pouzdani; takozvani „*connection-less*" odnosno nisu orijentirani na konekciju, ali podržavaju primanje poruka izvan slijeda kojim su poslane, što je korisno za neke primjene.
- ***STREAM*** *socketa*: oni su pak pouzdani; takozvani „*connection based*" odnosno orijentirani na konekciju, podržavaju primanje poruka u nîzu odnosno u istom redoslijedu s kojim su poslane.
- ***Sequential packet socket*** odnosno ***SOCK_SEQPACKET*** su slični prethodnom, uz dodatne mehanizme provjere.

Naime u komunikaciji između procesa odnosno programa, jedan proces odnosno program otvara neki od *unix socketa*, kao poslužitelj, a drugi proces odnosno program se spaja na njega, i to kao klijent.
Sve otvorene *unix sockete* možemo vidjeti s nekoliko naredbi:

- `lsof` - koristi se za ispis svih otvorenih *file deskriptora*.
- `netstat` - koristi se za ispis mrežnih konekcija te mrežnih, ali i *unix socketa*.
- `ss` - koristi se za ispis mrežnih, ali i *unix socketa*.

Dodatno sve je moguće vidjeti i unutar */proc/**PID**/fd/* direktorija od željenog procesa (s pripadajućim ***PID*** brojem).

Krenimo s listom svih *unix socketa*, pomoću naredbe `netstat` na sljedeći način (ispis smo skratili):

```
netstat --protocol=unix -p

Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags   Type      State        I-Node     PID/Program name    Path
unix  2      [ ]     DGRAM                  14342      1/systemd          run/systemd/notify
unix  3      [ ]     STREAM    CONNECTED    24869      1166/bash
unix  2      [ ]     DGRAM                  183158636  46329/sshd: root@pt
unix  2      [ ]     DGRAM                  191542883  37001/sshd: root@pt
unix  2      [ ]     DGRAM                  183460941  40783/sshd: root@pt
unix  3      [ ]     STREAM    CONNECTED    9210       2205/sshd
unix  3      [ ]     STREAM    CONNECTED    8809       1144/irqbalance
```

Što je ovdje vidljivo (prema stupcima):

- `PROTO` : `unix` označava *unix socket.*
- `TYPE` označava konkretnu vrstu *unix socketa*, koja može biti:
  - `STREAM` označava *stream unix socket*; ova vrsta *socketa* podržava dvosmjernu komunikaciju s mehanizmima transmisije podataka, koji se šalju i primaju u slijedu (redom), što znači kako je slanje i primanje podatak pouzdano i ne duplicirano. Kada se ovaj *socket* koristi za mrežnu komunikaciju, koristi se za ***TCP*** transportni protokol. On se označava sa `u_str` ili `STREAM`.
  - `SOCK_SEQPACKET` označava *stream unix socket*, podvrstu *sequential packet socket* koji je još pouzdaniji od prethodnog, po logici rada je poput ***SCTP*** transportnog protokola na mreži.
  - `DGRAM` označava *datagram unix socket* — ova vrsta *socketa* se koristi za kratke poruke jer je kod njega moguće primati poruke izvan slijeda kojim su poslane. Njegova mana je nedostatak mehanizma provjere, je li primatelj primio poruku. Ova vrsta *socketa* podržava dvosmjernu komunikaciju. Kada se ovaj tip *socketa* koristi na mreži, preko njega se koristi ***UDP*** mrežni transportni protokol. On se označava sa `u_dgr` ili `DGRAM`.
- `STATE` stupac označava stanje spajanja na taj *unix socket, može biti:* `CONNECTED` ili bez stanja (ako nema konekcije).
- `I-Node` - označava *inode* identifikacijski broj *unix socketa* (*inode* je identifikator na razini datotečnog sustava).
- `PID/Program name` označava ***PID*** programa/procesa koji je otvorio navedeni *unix socket* te ime sâmog *daemona*/servisa/programa/procesa koji ga koristi.

Istu statistiku možemo vidjeti i s novijom naredbom `ss` na sljedeći način:

```
ss -a --unix -p
```

**U primjeru koji slijedi analizirat ćemo *SSH* servis odnosno proces imena `sshd` i njegovu povezanost sa sustavom.**
Pogledajmo sada za ovaj *sshd* proces, ***PID***: `2205`, za koji smo vidjeli da koristi *inode:* *`9210`*, što nam daje naredba `lsof`:

```
lsof -p 2205 | grep unix
sshd    2205 root    1u  unix 0xffff8810385ae000   0t0     9210 socket
sshd    2205 root    2u  unix 0xffff8810385ae000   0t0     9210 socket
```

Te pogledajmo i što vidimo u /proc/ direktoriju za ovaj *PID* broj 2205:

```
ls -l /proc/2205/fd/

lr-x------ 1 root root 64 Jan  9 10:53 0 -> /dev/null
lrwx------ 1 root root 64 Jan  9 10:53 1 -> socket:[9210]
lrwx------ 1 root root 64 Jan  9 10:53 2 -> socket:[9210]
lrwx------ 1 root root 64 Jan  9 10:53 3 -> socket:[52440]
lrwx------ 1 root root 64 Jan  9 10:53 4 -> socket:[52442]
```

Ovdje je vidljivo da imamo dva *socketa* koji imaju isti *inode* broj 9210. Ovdje nije vidljivo kako se radi o *unix socketima,* ali nam je to jasno iz prijašnje naredbe kao i iz slijedeće naredbe s kojom ćemo ispisati sve *file deskriptore* i filtrirati sâmo 9210

```
lsof -n | grep 9210

sshd        2205      root     1u      unix 0xffff8810385ae000       0t0         9210 socket
sshd        2205      root     2u      unix 0xffff8810385ae000       0t0         9210 socket
```

**Zbog čega ovdje vidimo dva *unix socketa* (9210 socket *te ponovno* 9210 socket)?**
U načelu, zbog toga što svaki *socket* može komunicirati i s nekim drugim procesom odnosno programom.
Ali kako bismo jednostavno saznali s kojim, moramo imati noviji kernel (**3.10.x**) te pripadajući paket iproute-3.**xx** ili noviji, kako bi se jednostavno (s jednom naredbom) mogle povezati dvije točke u komunikaciji s *unix socketima*.
Ono što možemo odmah zaključiti iz gornjeg ispisa je slijedeće:

- Prvi *socket* se spaja na ***PID*** 2205 (što je naš *SSHD* servis) i to na *file deskriptor* broj **1** (to je standardni izlaz).
- Drugi *socket* se spaja isto na ***PID*** 2205 (što je naš isti *SSHD* servis) i to na *file deskriptor* broj **2** (standardna greška).

To znači kako se isti *unix socket*, u ovom slučaju, koristi i za standardni izlaz (**FD 1**) i ispis standardne greške (**FD 2**) prema nekom drugom procesu što znači da se i standardni ispis i ispis grešaka šalju prema drugom procesu (programu).

Sada još moramo saznati na što su oni spojeni, preko *inode* broja: **9210**, koji predstavlja ovaj konkretni *unix socket*.

```
ss -a -x --unix -p |  grep 9210

u_str  ESTAB 0 0 * 9210  * 24923  users(„sshd",pid=2205,fd=2),(„sshd",pid=2205,fd=1))
u_str  ESTAB 0 0 /run/systemd/journal/stdout 24923 * 9210 users(„systemd-journal",pid=671, fd=24))
```

U ovom ispisu vidimo komunikaciju između naša dva programa odnosno procesa (prvi i drugi red ispisa naredbe ss):

- U prvom retku vidimo da ovaj *unix socket* (**9210**) trenutno koristi naš *SSH servis* (***PID*** 2205) **FD** (*file deskriptor*) 2 koji komunicira i sa (**FD** 1) te kako im je drugi kraj komunikacije spojen na *unix socket* broj 24923.
- U drugom retku vidimo kako je taj isti *unix socket* (**9210**) spojen na program systemd odnosno na njegov journal proces ***PID***: 671 i to na njegov *unix socket* broj 24923 i to prema našem *unix socketu* broj **9210**.

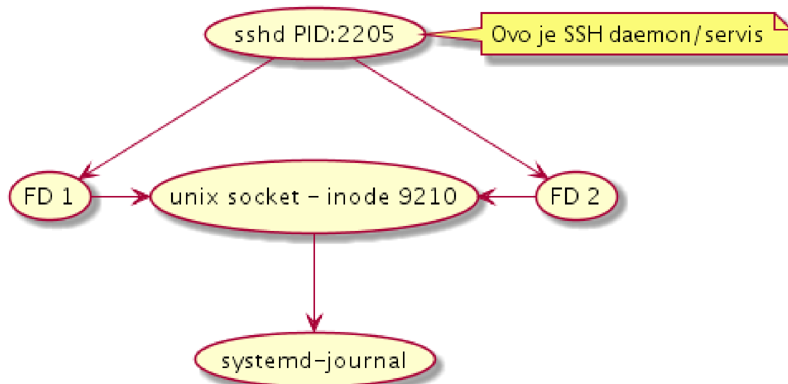
Naime u ovom prikazu vidimo obje točke komunikacije, kako jedna vidi drugu odnosno kako su međusobno spojene, stoga se vide dva *unix socketa* koji koriste isti *inode* broj 9210, što se ne vidi s ispisom prijašnje naredbe lsof -n | grep 9210.



Važno je znati kako se svi trenutno otvoreni *unix socketi* zapisuju u datoteku: /proc/net/unix
**Za brže filtriranje samo i isključivo *unix socketa*, možemo koristiti naredbu `lsof` s prekidačem `-U`.**

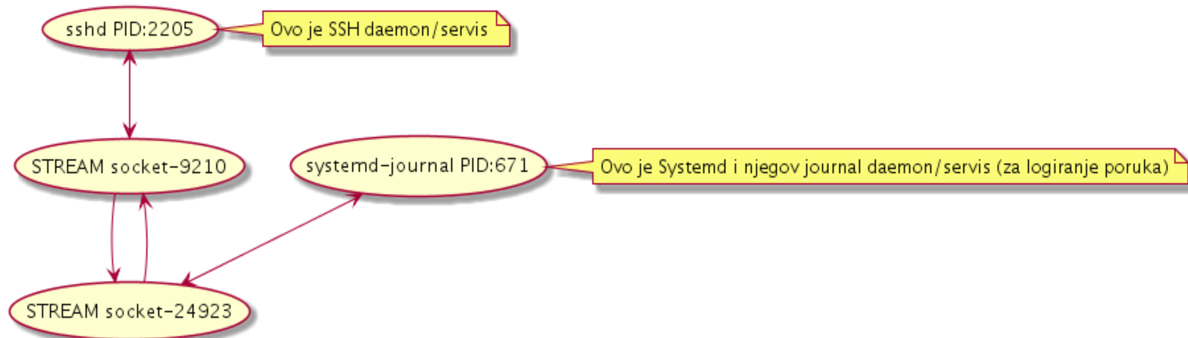
**Sumirajmo sve do sada naučeno**
U prvom primjeru smo vidjeli kako su *file deskriptori* (**FD**): **1** i **2** povezani, što znači kako se i standardni izlaz i standardna greška šalju negdje. To znači kako će se sve normalne izlazne poruke programa, koje predstavlja **FD 1**, kao i sve poruke o greškama, koje predstavlja **FD 2** šalju negdje. U drugom koraku vidimo i gdje se šalju, dakle šalju se na program za logiranje poruka (systemd-journal), pa će on tako zaprimiti i normalne poruke (u ovom slučaju, kad se netko spoji na SSH servis), ali i poruke o nekim greškama (bilo koja greška u spajanju na konkretni *SSH* servis). Sve to logički izgleda ovako (slika 36.):

Slika 36. Pogled na sshd proces i njegove *file deskriptore*.



U našem primjeru od prije, ako samo gledamo *STREAM unix sockete* i programe/procese koji ih otvaraju, to je logički vidljivo na slici 37.:

**Slika 37. Pogled na procese i unix stream sockete.**




(**1**) Ako pogledamo zadnja dva *socketa* iz naredbe `ls` od prije, ali sada s naredbom `lsof` i filtriranjem broja: `52440`:

```
lsof -n | grep 52440
sshd 2205    root     3u    Ipv4    52440   0t0       TCP    *:ssh     (LISTEN)
```

(**2**) Odnosno s filtriranjem broja: `52442`

```
lsof -n | grep 52442
sshd 2205    root     4u    Ipv6    52442   0t0       TCP    *:ssh     (LISTEN)
```

Vidimo da se ovdje radi o sljedećem:

- (**1**) U prvom ispisu (prvi *socket*) je *network socket*, za *SSH* servis, koji sluša na TCP portu 22, za IPv4. To je stoga jer gledamo vršni *ssh* servis (*daemon)* tj. onaj koji "sluša" na TCP portu 22 i koji kada se ostvari SSH konekcija (vidljivo kao `:ssh`), konekciju prosljeđuje novoj instanci SSH servisa odnosno novom ***sshd*** procesu, sa novim ***PID****om*.
- (**2**) U drugom ispisu (drugi *socket)* je isto *network socket*, za *SSH* servis, koji sluša na TCP portu 22, ali za IPv6. To je stoga jer gledamo vršni *ssh* servis odnosno onaj koji "sluša" na TCP portu 22 i koji kada se ostvari SSH konekcija (vidljivo kao `:ssh`), konekciju prosljeđuje novoj instanci SSH servisa (isto novom ***sshd*** procesu, sa novim ***PID****om*).

Naime u slučaju kada se prvi *SSH* klijent spoji na *SSH servis (*na ***sshd*** proces*)*, pokreće se nova instanca ***sshd*** servisa. Ako gledamo ***PID*** broj ove nove instance (procesa) *SSH* servisa (u našem slučaju ***PID*** je `40783`), vidjeli bi sljedeće(skratili smo ispis):

```
ls -l /proc/40783/fd/

lrwx------ 1 root root 64 Feb 19 12:32 0 -> /dev/null
lrwx------ 1 root root 64 Feb 19 12:32 1 -> /dev/null
lrwx------ 1 root root 64 Feb 19 12:32 2 -> /dev/null
lrwx------ 1 root root 64 Feb 19 12:32 3 -> socket:[183462222]
lrwx------ 1 root root 64 Feb 19 12:32 4 -> socket:[183460941]
lr-x------ 1 root root 64 Feb 19 12:32 5 -> pipe:[183460944]
l-wx------ 1 root root 64 Feb 20 12:20 6 -> /run/systemd/sessions/185016.ref
l-wx------ 1 root root 64 Feb 20 12:20 7 -> pipe:[183460944]
lrwx------ 1 root root 64 Feb 20 12:20 8 -> /dev/ptmx
```

**Ovdje imamo vidljiva samo dva *socketa*:**
Prvi *socket* je vidljiv sa sljedećom naredbom:

```
lsof -n | grep 183462222
sshd 40783  root 3u Ipv4 183462222 0t0  TCP 192.168.1.1:ssh->192.168.1.110:49364 (ESTABLISHED)
```

On je *network socket* (*vidljivo iz* `Ipv4` *i* `TCP`), a kroz njega teče *SSH* komunikacija (`:ssh`). Pri tome je IP adresa poslužitelja: 192.168.1.1, a klijentska IP adresa je: 192.168.1.110. Dodatno se vidi kako je veza uspostavljena (status je: `ESTABLISHED`).

Drugi *socket* je vidljiv sa sljedećom naredbom:

```
lsof -n | grep 183460941
sshd       40783     root     4u     unix 0xffff880dcae04400     0t0   183460941  socket
```

Dakle ovaj *socket* je *unix socket* (vidljivo iz oznake `unix`), a preko kojeg naša instanca *SSH* servisa može komunicirati sa vršnim (roditeljskim) ***SSH*** servisom.

## Drugi scenarij upotrebe Unix socketa

Sada ćemo pogledati malo drugačiji primjer u kojem ćemo ručno kreirati *unix socket* datoteku i povezati ju s našim programom. Naime inače bilo koji program kada ima potrebu komunicirati s drugim programima na istom računalu, može u bilo kojem trenutku preko sistemskih poziva zatražiti od sustava da mu kreira *Unix socket*. Taj (kao i svi drugi) *unix socket* će biti vidljiv u stablu `/proc/PID/fd/` (pri čemu je `PID`, ***PID*** broj procesa). U svakom slučaju od trenutka kada aplikaciji sustav kreira *Unix socket*, on je dostupan za rad odnosno komunikaciju s drugim programima. Ovo je uobičajeni način rada u kojem se *Unix socket* dodatno ne povezuje s nekom datotekom nego mu drugi programi direktno pristupaju i šalju podatke, te tako komuniciraju. Međutim moguće je kreirati *Unix socket* koji se može povezati s datotekom u datotečnom sustavu.

Slanjem podataka na tu datoteku, podaci se direktno prosljeđuju na taj *Unix socket* i u konačnici programu koji ga je kreirao. Neki programi rade upravo ovako, kako bi *Unix socket* bio lako vidljiv kao obična datoteka koju lako možemo pronaći i s našim programom pomoću nje komunicirati s krajnjim poslužiteljskim programom/aplikacijom. Ti programi obično ovu datoteku nazivaju s ekstenzijom `.sock`.

Za potrebe ovog primjera moramo instalirati jednu novu naredbu naziva ***net cat***.
Za ***RedHat/CentOS*** 6.x je to naredba `nc` (*net cat*) pa ju onda ovako i instalirajmo:

```
yum -y install nc
```

Za ***RedHat/CentOS*** 7.x je to novija naredba `ncat` koja dolazi u softverskom paketu *nmap-ncat* pa ćemo ju instalirati ovako:

```
yum -y install nmap-ncat
```

Sada ćemo na jednoj strani (u jednom terminalu/*shellu*) pokrenuti program `nc` odnosno `ncat` (ovisno koristimo li ***RedHat/CentOS*** 6.x ili 7+) i to kao poslužitelj. Naime ovdje pokrećemo poslužiteljsku stranu programa, kojem ćemo naložiti da kreira *Unix socket* datoteku i da se veže na nju odnosno da prima sve podatke koji se pošalju na nju. Primjere ćemo dati za ***RedHat/CentOS*** 7+ (za stariju inačicu *RedHat/CentOS* umjesto naredbe `ncat` koristite naredbu `nc`).

Za poslužiteljsku stranu odnosno prvi terminal/*shell* (na istom računalu), pokrenimo naredbu `ncat` na sljedeći način:

```
ncat -l -U /tmp/test.sock-2
```

S ovim će biti kreirana *Unix socket* datoteka: `/tmp/test.sock-2`

Kako bismo bili sigurni da je ovo stvarno standardna *Unix domain socket* datoteka, pogledajmo ju iz druge ljuske (terminala):

```
ls -alhi /tmp/test.sock-2
85069839 srwxr-xr-x 1 root root 0 Mar 27 12:30 /tmp/test.sock-2
```

Vidimo kako ova datoteka ima ovlasti `srwxr-xr-x` odnosno ovlast (`s`) na početku, što označava *unix socket datoteku.*

Sada u drugom terminalu (*shellu*), ali na istom računalu pronađimo ovaj poslužiteljski `ncat` proces, s naredbom `pidof`:

```
pidof ncat
42483
```

Vidimo kako mu je ***PID*** broj `42483`. A sada pogledajmo koje sve *unix sockete* ima otvorene ovaj proces (***PID*** `42483`):

```
lsof -p 42483 | grep -i unix

ncat    42483 root    3u  unix 0xffff9f41bd74ec00      0t0 627897517 /tmp/test.sock-2
```

To isto možemo vidjeti i sa drugom naredbom:

```
ss -a -x --unix -p | grep -i /tmp/test.sock-2

u_str  LISTEN     0      10     /tmp/test.sock-2 627897517 * 0  users:((„ncat",pid=42483,fd=3))
```

Vidimo kako ovaj *unix socket* koristi proces s ***PID*** brojem `42483` i da mu je to *file deskriptor* broj tri (`fd=3`).

Sada s druge strane (s drugog terminala odnosno *shella*) pošaljimo probni tekst i to direktno na ovu *Unix socket* datoteku:

```
echo „TEST poruka" | ncat -U /tmp/test.sock-2
```

A na poslužiteljskoj strani gdje smo pokrenuli `ncat` koji je kreirao *unix socket* datoteku: `/tmp/test.sock-2` i povezao se na njega, dobit ćemo poruku odnosno tekst koji smo poslali:

```
TEST poruka
```

Kada `ncat` primi ovu poruku, zaustavit će se. Međutim *UNIX socket* datoteka: `/tmp/test.sock-2` će ostati jer `ncat` naredba tako radi za razliku od drugih programa, koji kada završe s radom uredno i obrišu *unix socket* datoteku; ako su ju i kreirali tijekom pokretanja programa. Mi ju možemo kasnije obrisati sa sljedećom naredbom:

```
rm -f /tmp/test.sock-2
```

Sve ovo znači da komunikacija programa preko *unix socketa* uredno radi.
Dakle slanjem poruke/podataka na *unix socket* datoteku, koja je vezana za program koji ju je i kreirao, program povezan s *unix socket* datotekom prima te podatke. Pogledajmo primjer iz Linux sustava, u kojemu se također koristi ovakva vrsta *Unix socketa*, koji se veže za datoteku na datotečnom sustavu. U ovom primjeru se radi o posebnoj datoteci `/dev/log` koja predstavlja uređaj koji se koristi za *logiranje* poruka sustava.

Ovaj uređaj (posebnu datoteku) kreira `systemd` servis/*daemon* (za ***RedHat/CentOS*** 7.x ili noviji). Sada pogledajmo o čemu se radi odnosno kako izgleda ova posebna datoteka:

```
ls -al /dev/log
srw-rw-rw- 1 root root 0 Dec 27 14:30 /dev/log
```

Vidimo kako je ova datoteka *unix socket* (`unix`). Sada potvrdimo i koji program se veže za nju, preko *unix socketa*:

```
netstat --protocol=unix -p |grep /dev/log

unix  14      [ ]         DGRAM              12530    1/systemd              /dev/log
```

Nadalje vidimo da je ovo definitivno *unix socket* (`unix`) i kako je on vrste `DGRAM` (*datagram*) te da ga koristi `systemd` servis. Probajmo poslati neki tekst na njega, što bi trebalo rezultirati zapisivanjem tog teksta u sistemsku log datoteku: `/var/log/messages`. Sada pošaljimo probni tekst preko naredbe `ncat`(jer ne možemo direktno slati podatke na ***unix datagram socket*** datoteku)

```
echo „Test" | ncat -uU /dev/log
```

Ovdje smo koristili `ncat -uU` jer je *unix socket* na koji šaljemo podatke, vrste *datagram*.
Naime da smo koristili samo prekidač `-U` dobili bi grešku: `Ncat: Protocol wrong type for socket.`

Pogledajmo sada sistemsku log datoteku: `/var/log/messages` u kojoj je naša poruka morala završiti:

```
tail /var/log/messages

Mar 28 12:28:01 svr102 systemd: Starting Session 412352 of user root.
Mar 28 12:29:21 svr102 journal: Test
```

Na kraju vidimo poruku koju smo i poslali, dakle to je dokaz da ova komunikacija radi.



Za druge primjere upotrebe naredbe `lsof`, pogledajte i poglavlja:
**4.5.5.1. File deskriptori i naredba *lsof*.**
**25.7.6. Naredba *list open files* (*lsof*).**

**Izvori informacija:** (602),(603),(604),(605), **`man lsof`**,**`man ncat`**,**`man nc`**,**`man netstat`**,**`man ss`**, **`man 7 socket`**,**`man 5 proc`**.

### 9.4.3.2. Network *socketi*

Mrežne spojne točke, ***mrežne utičnice*** *ili* ***network sockete*** smo već letimično vidjeli u prijašnjim primjerima, a sada ćemo na njih malo više obratiti pažnju. Pojam mrežne spojne točke (*network socket*) inače se odnosi na jednu stranu komunikacije između aplikacije i operativnog sustava. Dok njena druga strana završava na mrežnom portu koji se veže za određenu IP adresu.
To možemo promatrati i kao komunikacijski kanal: ***aplikacija - socket - (OS) - IP adresa:port***.
Pri tome je lokalni ***socket*** interna točka komunikacije (**aplikacija – *socket* - OS**), a ***port*** s pripadajućom IP adresom na koju je vezan, kao vanjska točka komunikacije (**OS - *IP:port***).



Svaka pojedina mrežna utičnica (**network socket**) nosi jedinstveni identifikator (***INODE*** broj), a veže se za pet parametara veze:

- Protokol; primjerice TCP ili UDP.
- Pâr: izvorišna IP adresa i njen pripadajući port te odredišna IP adresa i njen pripadajući port.

Dakle mrežna utičnica (**network socket**) jedinstveni je identifikator svake pojedine otvorene veze.

Ako gledamo s razine aplikacije, *socket* se svakako promatra kao lokalna točka u komunikaciji. Međutim pod pojmom *network **socket***, podrazumijeva se i cijeli komunikacijski kanal: ***aplikacija - socket - (OS) - IP adresa:port***, o kojem ćemo ovdje i govoriti. To možemo reći i ovako: *network socket* identificira cijeli komunikacijski kanal: ***aplikacija - socket - (OS) - IP adresa:port*** ili i dodatno, odredišne: **IP adresa** i njen **port.** pa se često skraćuje naziv. Umjesto ***network socket*** u samo ***socket***. U konačnici, širi pojam *network socketa* definiraju: protokol + lokalna **IP** i njen port + udaljena **IP** i njen port.

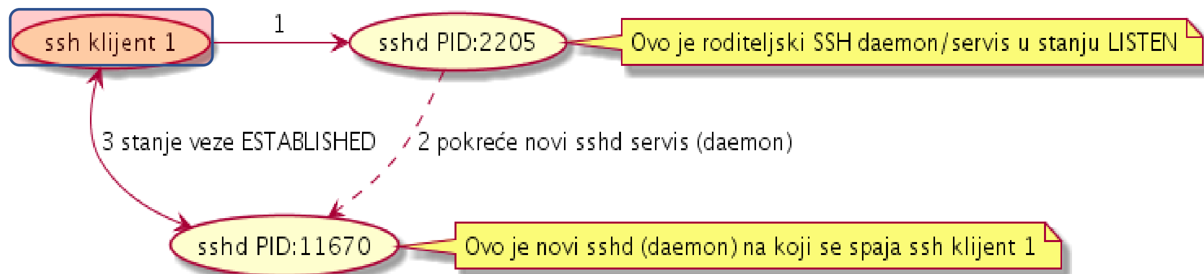
Vratimo se prvo našem vršnom (roditeljskom) *SSH* servisu, sa slike 37. od nekoliko stranice prije, čiji zadatak je slušanje na *TCP* portu *22*. To znači kako on mora imati otvoren *TCP* port *22*, kako bi zaprimio inicijalne *SSH* klijente. Naime kada se *SSH* klijent inicijalno spoji na njega (na *SSH* servis), on potom pokreće novu instancu (proces) *ssh* servisa, na koji se korisnik preusmjerava, a koji je zapravo novi *sshd* servis koji je zadužen za dalju komunikaciju s tim (jednim) *ssh* klijentom.



Za svakog novog ***ssh*** klijenta se pokreće novi ***ssh*** servis (proces), koji je aktivan sve dok se ***ssh*** klijent ne odspoji.

Ako gledamo sve vezane procese SSH servisa, to logički možemo promatrati kao što je vidljivo na slici 38.:

**Slika 38. Veza SSH servisa i njegovih pod procesa**




Sada pogledajmo stanje svih *socketa*, za naš roditeljski (vršni) *sshd* servis čiji ***PID*** broj je `2205`:

```
ls -l /proc/2205/fd/

lr-x------ 1 root root 64 Jan  9 10:53 0 -> /dev/null
lrwx------ 1 root root 64 Jan  9 10:53 1 -> socket:[9210]
lrwx------ 1 root root 64 Jan  9 10:53 2 -> socket:[9210]
lrwx------ 1 root root 64 Jan  9 10:53 3 -> socket:[52440]
lrwx------ 1 root root 64 Jan  9 10:53 4 -> socket:[52442]
```

U prijašnjem poglavlju smo vidjeli kako su *file deskriptori* **1** (*socket:[9210]*) i **2** (*socket:[9210]*) zapravo *unix socketi* za *IPC* komunikaciju. Sada ćemo pomoću naredbe `lsof` analizirati zadnja dva *file deskriptora*: **3** (*socket:[52440]*) i **4** (*socket:[52442]*):

```
lsof -p 2205 | egrep -i „52440|52442"

COMMAND  PID  USER   FD   TYPE          DEVICE  SIZE/OFF    NODE NAME
sshd     2205 root   3u   Ipv4           52440       0t0     TCP *:ssh (LISTEN)
sshd     2205 root   4u   Ipv6           52442       0t0     TCP *:ssh (LISTEN)
```

Ovdje vidimo da su *file deskriptori (FD)* **3** i **4** poveznice na ***Inode*** *broj*: `52440` ***i*** `52442`, a koji su zapravo identifikatori ***network socketa***. Ako pogledamo njihova polja: `TYPE` koja su označena kao: `Ipv4` i `Ipv6` ona govore da je:

- *Inode broj* (`DEVICE`) **52440** povezan na ***FD*** **3**, koji je `IPv4` ***network socket*** i kako je njegov transportni mrežni protokol *TCP* (polje *NODE*) u stanju `LISTEN`*;* dakle sluša na svim mrežnim sučeljima. Sva mrežna sučelja se označavaju sa `*:` i to na TCP portu koji je rezerviran za **`ssh`** protokol, dakle portu broj 22.
- *Inode broj* (`DEVICE`) **52442** povezan na ***FD*** **4**, koji je `IPv6` ***network socket*** i kako je njegov transportni mrežni protokol *TCP* (polje *NODE*) u stanju `LISTEN`*;* dakle sluša na svim mrežnim sučeljima. Sva mrežna sučelja se označavaju sa `*:` i to na TCP portu koji je rezerviran za **`ssh`** protokol, dakle portu broj 22.

To sve nam govori ono što smo već rekli: za IPv4 je aktivan jedan *mrežni socket*, koji sluša na *TCP* portu *22* (*ssh*) i po jedan za IPv6 protokol. Pogledajmo kako se to vidi s naredbom `netstat`:

```
netstat -tunap | grep sshd | grep LISTEN

Proto Recv-Q Send-Q Local Address     Foreign Address     State      PID/Program name
tcp        0      0 0.0.0.0:22            0.0.0.0:*       LISTEN       2205/sshd
tcp6       0      0 :::22                 :::*            LISTEN       2205/sshd
```

Spajanjem novog *ssh* klijenta, na naš roditeljski *sshd* servis (***PID*** ***broja*** *`2205`*), roditeljski *ssh* servis, pokreće novi *sshd* servis, samo za ovog *ssh* klijenta, kao i za svakog novog.

U našem slučaju ovaj novi *sshd* servis, dobiva ***PID*** broj `11670` i *ssh* konekcija se ostvaruje preko njega, kako ćemo sada i vidjeti:

```
netstat -tunap | grep 11670

Proto Recv-Q Send-Q Local Address      Foreign Address       State       PID/Program   name
tcp      0      0   192.168.1.254:22   192.168.1.100:59039   ESTABLISHED 11670/sshd:   root@pt
```

Dakle naš *sshd* server ima IP adresu `192.168.1.254`, a *ssh* klijent `192.168.1.100`.

Pogledajmo sve sa strane mrežnih *socketa* za ovaj novi *sshd* proces/program/servis, s ***PID*** brojem `11670`:

```
lsof -n -p 11670 | grep -i Ipv4

sshd 11670 root 3u Ipv4 196069454 0t0 TCP   192.168.1.254:ssh-> 192.168.1.100:59039
(ESTABLISHED)
```

Zatim pogledajmo sve *sockete* za navedeni ***PID*** broj `11670`:

```
ls -l /proc/11670/fd/

lrwx------ 1 root root 64 Feb 22 10:29 0  -> /dev/null
lrwx------ 1 root root 64 Feb 22 10:29 1  -> /dev/null
lrwx------ 1 root root 64 Feb 22 10:29 2  -> /dev/null
lrwx------ 1 root root 64 Feb 22 10:29 3  -> socket:[196069454]
lrwx------ 1 root root 64 Feb 22 10:29 4  -> socket:[196101230]
lr-x------ 1 root root 64 Feb 22 10:29 5  -> pipe:[196101233]
l-wx------ 1 root root 64 Feb 22 11:49 6  -> /run/systemd/sessions/198122.ref
l-wx------ 1 root root 64 Feb 22 11:49 7  -> pipe:[196101233]
lrwx------ 1 root root 64 Feb 22 11:49 8  -> /dev/ptmx
```

Na ispisu vidimo dva *socketa* (`socket:`); pri tome je ***prvi socket*** vidljiv na slijedeći način (pomoću `196069454`):

```
lsof -n -p 11670 | grep -i 196069454

sshd 11670 root 3u Ipv4 196069454 0t0 TCP 192.168.1.254:ssh->192.168.1.100:59039 (ESTABLISHED)
```

Dakle to je mrežni *socket* o kojem smo pričali. Dok je ***drugi socket*** vidljiv na slijedeći način (pomoću `196101230`):

```
lsof -n -p 11670 | grep -i 196101230

sshd    11670 root    4u  unix 0xffff880d242be000      0t0 196101230 socket
```

Ovaj drugi *socket* je *unix socket* koji smo već objasnili u prethodnoj cjelini.



Vezano za dodatne primjere i detalje upotrebe ***network socketa*** pogledajte poglavlja:
**24. Transportni protokoli (OSI sloj 4).**
**25.1. Linux mrežni model.**

## Napomena vezana za *inode* broj koji stalno spominjemo

U diskovnom podsustavu Linuxa vršna komponenta kojoj pristupaju svi programi/procesi/aplikacije je virtualni datotečni sustav (*virtual file system*) odnosno **VFS**. On se logički nalazi iznad svih datotečnih sustava, s kojima može biti formatirana bilo koja od particija diskova. Jedna od namjena ***VFS*a** je omogućiti i posebnim datotečnim sustavima poput ***procfs***, ***sysfs*** i drugih, koji se ne nalaze na nekoj od particija diskova, već u RAM memoriji, normalan i nesmetan rad.



Vezano za ***VFS***, pogledajte poglavlje:
**14. Diskovni (I/O) podsustav.**

Naime posebni datotečni sustavi poput `/proc` ili `/sys`, koji predstavljaju unose u kernelu, se nikada i ne trebaju naći na nekom fizičkom disku, odnosno u datotečnom sustavu na nekoj od particija fizičkih diskova. S druge strane, standardne datoteke koje se nalaze na normalno formatiranoj particiji nekog od fizičkih diskova, moraju se stvarno i nalaziti na tim particijama diskova, odnosno na njihovim datotečnim sustavima. Mi u ovom poglavlju govorimo o ovoj prvoj kategoriji *virtualnih* datoteka, mada i jedne i druge imaju svoju strukturu podataka: od sâmog sadržaja, ovlasti i slično, a koje se zapisuju u određeni *inode* identifikator svake pojedine datoteke. Dakle svaki objekt u virtualnom datotečnom sustavu odnosno **VFS**-u ima svoj *inode* unos. Pod objektom ugrubo podrazumijevamo standardne datoteke, ali i posebne objekte poput *network socketa* i *pipe*-a, koji nemaju klasično ime datoteke koju predstavljaju jer su oni zapravo *file deskriptori*.
Stoga se oni identificiraju sâmo i isključivo pomoću svog *inode* broja. Naredba poput `lsof` ispisuje *inode* brojeve za sve *file deskriptore* koje vidi na sustavu i prikazuje ih u polju *NODE*. S druge strane naredba `netstat` također ispisuje *inode* brojeve, ali samo za *unix sockete*, koji su prave datoteke koje se nalaze na diskovnom sustavu.



Vezano za *inode*-ove, podsjetite se poglavlja:
**4.5. Datotečni sustav detaljnije.**



Za primjere o mrežnim konekcijama, *file deskriptorima* i njihovoj poveznici, pogledajte i poglavlja:
**4.5.5.1. File deskriptori i naredba *lsof*.**
**25.7.6 Naredba list open files (*lsof*).**
**25.7.7. Naredba *socket statistics* (*ss*)** - *vezano za network sockete, pogledajte primjer broj* ***6.*** *i objašnjenja koja slijede.*

**Izvori informacija:** **(1032),(1033),(1034),** `man lsof`,`man netstat`,`man 7 socket`.

### 9.4.4. Linux IPC naredbe

Standardno u većini distribucija linuxa dobivamo dvije naredbe s kojima možemo vidjeti statistike vezane za starije *IPC* mehanizme, koji su naslijeđeni iz ***Unix System V***, a i danas se još uvijek, doduše minimalno, koriste. Međutim neki specijalizirani programi, poput određenih baza podataka ih i dalje koriste. Dakle sada više nećemo govoriti o *IPC* mehanizmima odnosno mehanizmima za komunikaciju između procesa koje smo već spominjali (*unix* i network *socketi*, unix/linux *pipes* i *named pipes*), već o drugoj metodi komunikacije među procesima, koja je naslijeđena iz ***Unix System V***. Programi/naredbe koji mogu prikazati ovu vrstu *IPC*-a su: `ipcs` i `lsipc`.
Prva naredba `ipcs`, daje nam statuse svih **Unix *System V*** *IPC* mehanizama. Pogledajmo koje sve prekidače ona ima:

- `-q` — ispisuje samo *message queue IPC.*
- `-s` — ispisuje samo *IPC semafore.*
- `-m` — ispisuje samo *IPC shared memory.*
- `-l` — ispisuje ograničenja (limite) sustava, prema određenom IPC mehanizmu.
- `-q` — specifičan upit (za neki od gornja tri prekidača).
    - `-i` — upit za gornji prekidač — prema *msqid/shmid/semid*

Pogledajmo primjer, u kojem ćemo ispisati sve Unix *System V IPC* mehanizme i to sva tri gore navedena (ispis je skraćen):

```
ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0xffffffff 0          root       600        0            0
0x000004d2 65537      root       666        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0xf902fc14 0          root       600        222        1

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x00000000 0          hpsmh      600        1
0x00000000 458753     hpsmh      600        1
0x00000000 425995     hpsmh      600        1
0x56565656 491532     root       666        3
0x00000000 524301     hpsmh      600        1
```

Ispis smo skratili ali vidljivo je kako svaka od metoda *Unix System V IPC*a, za svaki *IPC* ima ključ (*key*) te jedinstveni identifikator, koji ovisi o vrsti *IPCa*:

- Za *Message Queues* je to `msqid`.
- Za *Shared Memory Segment* je to `shmid`.
- Za *Semaphore Array* je to `semid`.

Osim toga, za svaki od njih vidimo i koji korisnik je njegov vlasnik kao i ovlasti pojedinog *Unix System V IPC*a, na osnovu oktalne vrijednosti polja: *`perms`*. Sada pogledajmo ograničenja sustava, po pitanju ***Unix System V IPC*a**:

```
ipcs -l

------ Messages Limits --------
max queues system wide = 32768
max size of message (bytes) = 8192
default max size of queue (bytes) = 16384
------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 18014398509465599
max total shared memory (kbytes) = 18014398442373116
min seg size (bytes) = 1
------ Semaphore Limits --------
max number of arrays = 128
max semaphores per array = 250
max semaphores system wide = 32000
max ops per semop call = 32
semaphore max value = 32767
```

Pogledajmo i detalja za pojedine *Unix System V IPC*ove. Primjerice zanima nas: ***Message Queue***, ***msqid***, broj: `65537`:

```
ipcs -q -i 65537

Message Queue msqid=65537
uid=0   gid=0   cuid=0  cgid=0  mode=0666
cbytes=0        qbytes=16384    qnum=0  lspid=3834      lrpid=4335
send_time=Wed Feb 21 10:56:15 2018
rcv_time=Wed Feb 21 10:56:23 2018
change_time=Tue Jan  9 10:53:02 2018
```

Pošto se radi o ***Unix System V*** *IPC*ovima, koji se minimalno i rijetko koriste u Linuxu, o njima nećemo više govoriti.

**Izvori informacija: (K-4), `man ipcs`, `man lsipc`, `man 7 svipc`.**

## 9.4.5. Izolirani prostori Linuxa (*Linux namespaces*)

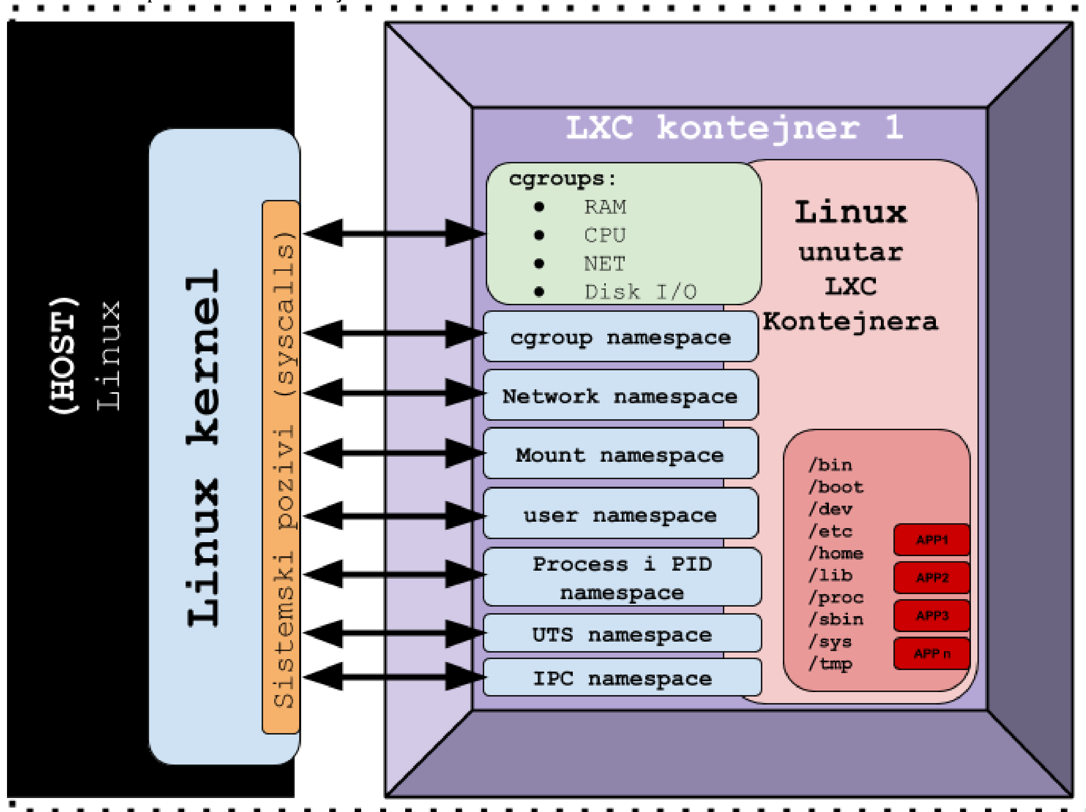
U ovom naprednom poglavlju govorit ćemo o posebnoj tehnologiji izoliranih prostora Linuxa ili takozvanih Linux *namespace*-ova. *Namespace* nam daje funkcionalnost s kojom Linux kernel može razdvojiti i raspodijeliti resurse sustava poput: CPU, RAM, mreže, diskovnog sustava, korisnika i korisničkih grupa, sve tako da se resursi koji se nalaze u jednoj logičkoj cjelini ili particiji nalaze u potpuno izoliranom prostoru, a istovremeno se izvršavaju i rade poput bilo kojih drugih Linux procesa odnosno programa. Dodatno unutar izoliranog prostora izolirani su i korisnici i korisničke grupe, mrežna komunikacija, pristup diskovnom sustavu, sustavu komunikacije između procesa (***IPC***), naziv računala i domene te drugo. Dakle logično je za zaključiti kako se ove metode koriste za izolaciju poput izolacije u kojoj se nalaze virtualna računala unutar jednog fizičkog poslužitelja, ali na znatno jednostavniji, učinkovitiji i brži način. Ova tehnologija se koristi za razne Linux kontejnerske tehnologije, poput ***LXC*** **Linux kontejnera**, ***Dockera*** ili ***Podmana*** koje su naizgled slične virtualizaciji, ali bez potrebe za virtualiziranjem svih komponenti računala. Ove tehnologije rade tako da izoliraju pokrenute programe u posebni prostor za izolaciju sistemskih resursa, unutar već postojećeg Linuxa odnosno operativnog sustava. Za ovu potrebu, počevši od Linux kernela v.3.8. do danas, kreirane su posebne kategorije za izolaciju sistemskih resursa. Dakle zasebnih *name space-ova,* od kojih je svaki zadužen za jedno područje izolacije. Trenutno je u kernelima 4.15.x dostupno njih sedam (7+1).
Pri tome svaki od navedenih *name spaceova* ima odgovarajuću *oznaku* u Linuxovom virtualnom datotečnom sustavu (***VFS***).
U Linuxu trenutno imamo dostupne sljedeće metode izolacije:
***Cgroups*** (`cgroup`) - ovo zapravo nije *namespace* sâm po sebi već se koristi za kontrolne grupe odnosno za izolaciju pripadnosti kontrolnih grupa pokrenutih procesa koji koriste neki *namespace*.

Kontrolne grupe su zadužene za dodjeljivanje, ograničavanje i izolaciju resursa računala, poput: CPU, RAM, disk (I/O) i mreže.

1. ***Cgroup namespace*** (`cgroup`) – za skrivanje identiteta kontrolne grupe (*cgroup*) čiji je proces član.
2. ***IPC namespace*** (`ipc`) – za procese i njihove mehanizme komunikacije (*IPC — Interprocess Communication*).
3. ***PID namespace*** (`pid`) – za izolaciju procesa i ***PID***-ova (***PID***=*Process ID)* pokrenutih procesa.
4. ***Network namespace*** (`net`) – za izolaciju mreže i mrežnih sustava.
5. ***User namespace*** (`user`) – za izolaciju korisnika i njihovog rada (prema: ***UID*** i ***GID***).
6. ***Mount namespace*** (`mnt`) – zaduženih za *montiranje (mount)* unutar izoliranog prostora; primjerice unutar ***LXC*** linux kontejnera.
7. ***UTS namespace*** (`uts`) – omogućava promjene *hostname* i *domainname* imena unutar izolacije, za različite procese.

Pogledajmo i dva scenarija upotrebe Linux kontejnera: slika **39** prikazuje standardni punokrvni Linux kontejner, a ispod nje je vidljiv mikro kontejner – slika **40.**

**Slika 39. Linux namespaces i klasični Linux kontejner.**

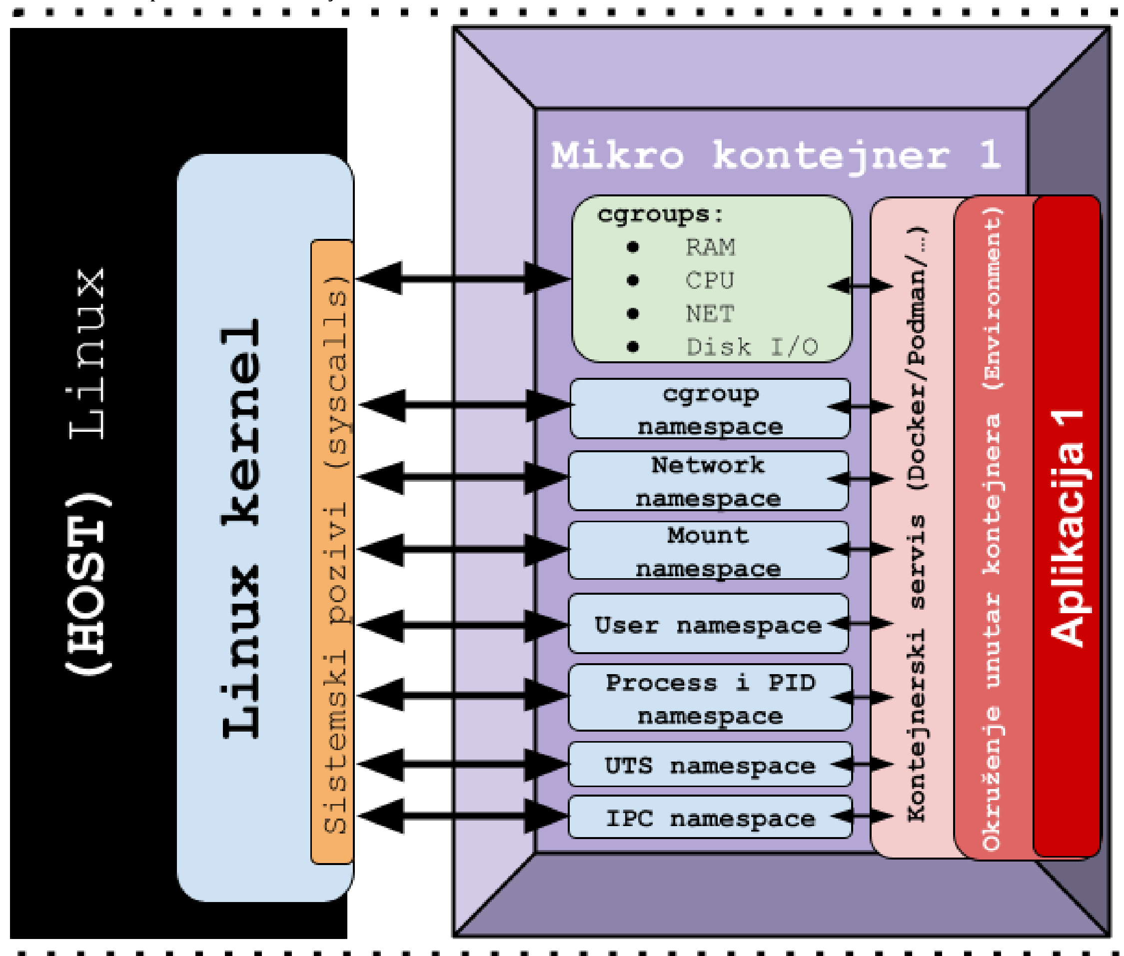



Na slici **39.** je vidljiva namjena Linux kontejnera: jedan kontejner - više aplikacija (**APP1, APP2, APP3, … APP n**).
Ispis odnosno listu *namespace-ova* koji su kompilirani u vaš kernel, možete vidjeti s naredbom:

```
egrep '^CONFIG_NAMESPACES|_NS=' /boot/config-$(uname -r) | grep -v CONN
```

Na donjoj slici 40. vidljiva je drugačija namjena Linux kontejnera: jedan mikro kontejner – jedna aplikacija tj. mikro servis (**APP1**). Ovo je primjer upotrebe Linux kontejnera (pr. ***Docker***) za mikro servise, kako bi bio portabilan i lako upotrebljiv.

**Slika 40. Linux namespaces i Linux mikro kontejner.**




Kao što je vidljivo na slici 39. mi imamo pokrenut jedan Linux **LXC** kontejner, koji je pokrenuo proces imena (`lxc monitor`) koji je potom u izoliranom *linux namespaceu* pokrenuo novi linux kontejner sa svojim `init` procesom.

Sve *linux name spaceove* možemo izlistati s naredbom `lsns` na sljedeći način:

```
lsns

        NS TYPE   NPROCS   PID USER             COMMAND
4026533038 mnt         1 19418 root             [lxc monitor] /var/lib/lxc 136
4026533047 mnt         7 19490 root             /sbin/init
4026533048 uts         7 19490 root             /sbin/init
4026533049 ipc         7 19490 root             /sbin/init
4026533050 pid         7 19490 root             /sbin/init
4026533052 net         7 19490 root             /sbin/init
4026533122 cgroup      7 19490 root             /sbin/init
```

U prvom stupcu vidimo identifikator *name spacea* (`NS`), a u drugom vrstu (`TYPE`) koju smo naveli gore. Potom vidimo ***PID*** broj procesa (`PID`) i korisnika koji je pokrenuo taj proces (`USER`) te u krajnjem desnom stupcu ime naredbe ili programa koji je pokrenut (`COMMAND`).

Moguće je filtrirati i samo željene imenične prostora Linuxa (uz druge detalje [**`--output-all`**]), primjerice samo mrežne:

```
lsns --output-all --type=net
  NS       TYPE   PATH              NPROCS  PID  PPID COMMAND      UID USER NETNSID
NSFS
4026533052 net   /proc/19490/ns/net    7       1   0   /sbin/init  0 root unassigned  0
```

Svaki od procesa koji se eventualno izvršavaju u nekom *imeničnom prostoru*, ako primjerice imamo pokrenut ***LXC*** ili ***Docker kontejner*** u svom ***PID*** direktoriju (`/proc/PID/`) ima i pôd direktorij imena: `ns`.
Unutar njega su vidljivi svi ovi identifikatori i njihova poveznica.

Tako je primjerice za prethodno vidljivi `init` proces s ***PID*** brojem `19490`, vidljivo sljedeće (ispis smo skratili):

```
ls -ali /proc/19490/ns/

39203391 lrwxrwxrwx 1 root root 0 Sep 14 13:00 cgroup -> cgroup:[4026533122]
37147868 lrwxrwxrwx 1 root root 0 Sep 13 13:52 ipc -> ipc:[4026533049]
37147865 lrwxrwxrwx 1 root root 0 Sep 13 13:52 mnt -> mnt:[4026533047]
37147869 lrwxrwxrwx 1 root root 0 Sep 13 13:52 net -> net:[4026533052]
37147866 lrwxrwxrwx 1 root root 0 Sep 13 13:52 pid -> pid:[4026533050]
39215229 lrwxrwxrwx 1 root root 0 Sep 14 13:10 pid_for_children -> pid:[4026533050]
39203390 lrwxrwxrwx 1 root root 0 Sep 14 13:00 user -> user:[4026531837]
37147867 lrwxrwxrwx 1 root root 0 Sep 13 13:52 uts -> uts:[4026533048]
```

Dakle kao i u bilo kojoj komunikaciji između procesa u Linuxu, ovdje vidimo poveznice (izrezali smo prvi i zadnji stupac).
Dakle veza je: **INODE ←→ INODE** te sada to vidimo ovako:

```
39203391 ← cgroup            → cgroup:[4026533122]
37147868 ← ipc               → ipc:[4026533049]
37147865 ← mnt               → mnt:[4026533047]
37147869 ← net               → net:[4026533052]
37147866 ← pid               → pid:[4026533050]
39215229 ← pid_for_children  → pid:[4026533050]
39203390 ← user              → user:[4026531837]
37147867 ← uts               → uts:[4026533048]
```

U prvom stupcu, lijevi broj je lokalni ***INODE*** broj, pa slijedi vrsta *name spacea* (pr. `net` koji smo označili) i na kraju je poveznica na koji je identifikator (***INODE***) spojen, a to je desni broj u uglatoj zagradi, primjerice za `net` je: `[4026533052]`.

Dakle za network *namespace* za ovaj proces (u označenom četvrtom retku) imamo sljedeću poveznicu:

```
37147869 → net → 4026533052
```

Ako pogledamo desni ***INODE*** broj (`4026533052`) kao identifikator, pronađimo s kime je spojen, pomoću naredbe `lsof`:

```
lsof | grep 4026533052

COMMAND      PID     USER   FD      TYPE         DEVICE SIZE/OFF NODE        NAME
lxc-start 19418      root    9r      REG            0,3        0 4026533052  net
lxc-start 19418      root   18r      REG            0,3        0 4026533052  net
```

Poveznica ovdje je (I)**NODE** s desne strane (`4026533052`) i ***PID*** broj procesa na koji je povezan (s lijeve strane); konkretno ***PID***: `19418`.
Istu poveznicu smo mogli dobiti i pomoću naredbe `ps`, na sljedeći način:

```
ps -eo netns,pid,ppid,user,args --sort netns | grep 4026533052
```

Sada znamo kako je ovaj `init` proces unutar *namespacea*, a to je `init` proces unutar ***LXC*** kontejnera, s ***PID*** brojem (`19490`) pokrenut od strane naredbe: `lxc-start`, koja ima ***PID***: `19418`.
Ovaj proces `lxc-start` je u konkretnom slučaju proces koji je pokrenuo naš `init` proces unutar Linux ***LXC*** kontejnera.

On s njim komunicira preko standardnih *file deskriptora* na slijedeći način (skratili smo ispis):

```
ls -al /proc/19418/fd/

lrwx------ 1 root root 64 Sep 13 13:52 0  -> /dev/null
lrwx------ 1 root root 64 Sep 13 13:52 1  -> /dev/null
lrwx------ 1 root root 64 Sep 14 13:16 13 -> /var/lib/lxc/136/rootfs/.lxc-keep
lr-x------ 1 root root 64 Sep 14 13:16 14 -> mnt:[4026533047]
lr-x------ 1 root root 64 Sep 14 13:16 15 -> pid:[4026533050]
lr-x------ 1 root root 64 Sep 14 13:16 16 -> uts:[4026533048]
lr-x------ 1 root root 64 Sep 14 13:16 17 -> ipc:[4026533049]
lr-x------ 1 root root 64 Sep 14 13:16 18 -> net:[4026533052]
```

Dakle ***PID***: `19418` preko *file deskriptora* broj `18` -> komunicira s -> `net:[4026533052]` koji je ***INODE*** poveznica preko *network namespacea* na naš izvorni `init` proces sa ***PID*** brojem: `19490` i to za mrežni (`net`work) dio komunikacije između ova dva procesa.

Pogledajte i druge korisne naredbe za izolirane prostore linuxa:

- Naredbe za mrežne izolirane prostore: `ip netns` (`add`|`del`|`list`|`exec`).
    - Kao i druge dijelove `ip` naredbe.
- Ali i naredbe za druge izolirane prostore (*name/pid/user/mount/*...): `lsns`, `lsenter`, `unshare` i druge.

Ako želite ručno raditi sa *cgroup* mehanizmima[746], instalirajte si softverski paket koji sadrži potrebne programe za rad s njima:

```
yum install -y libcgroup-tools
```

S njim ćete dobiti i slijedeće naredbe (navest ćemo ih samo nekoliko):`cgcreate`, `cgexec`, `cgdelete`, `cgset`, `cgclear`.



Od **Red Hat 7.7.** paket **`libcgroup-tools`** se više ne koristi jer je sada ***systemd*** jedini zadužen za ***cgroup*** mehanizme!



Vezano za **cgroup** pogledajte poglavlje:
**10.7.2.5. Ograničavanje CPU resursa i cgroup mehanizam.**

Za primjere o mrežnim izoliranim prostorima Linuxa (*Linux network namespaces*), pogledajte poglavlja:
**26.8.1. Network Namespaces.**
**20.6.4. VETH - posebno mrežno sučelje.**
**27.2. Linux kontejneri.**
**27.2.2. Podman Linux kontejneri.**

## Pogledajmo još neke primjere upotrebe izoliranih imeničnih prostora Linuxa!

Još jedna od korisnih naredbi za izolirane prostore (*namespaces*) Linuxa je naredba `unshare` koja omogućuje pokretanje programa unutar izoliranog prostora Linuxa koji nije dijeljen s njegovim roditeljem odnosno Linuxom (*unshare* = engl. nedijeljeni).

***UTS namespace*** (`uts`)
*UTS namespace* kontrolira naziv računala (*hostname*) i **NIS** domenu.
Kreirajmo novi **UTS *namespace*** unutar kojega će se pokrenuti *bash* ljuska:

```
unshare --uts /usr/bin/bash
```

Važno je razumjeti da se nakon pokretanje ove naredbe više ne nalazimo u izvornoj instanci Linuxa već unutar izoliranog *UTS namespace*-a. Pogledajmo sve *UTS namespace-ove*.

```
lsns -t uts

NS         TYPE  NPROCS   PID  USER   COMMAND
4026531838 uts      129     1  root   /usr/lib/systemd/systemd --switched-root …
4026532191 uts        2  1838  root   /usr/bin/bash ← Ovo je naš UTS
```

Vidim da se pojavio novi *UTS namespace; TYPE:* `uts`. Te da je unutar njega pokrenuta *bash* ljuska.

A sada s naredbom `nsenter`, unutar našeg *UTS namespace-a* koji ima *PID* broj `1838` možemo promijeniti ime računala u NS1:

```
nsenter -t 1838  -u hostname NS1
```

Međutim ovakav izolirani prostor Linuxa (*UTS namespace*) nam nije nužno previše upotrebljiv. Stoga idemo dalje.

***UTS namespace*** (`uts`) i ***Network namespace*** (`net`)
Sada ćemo kreirati novi UTS izolirani prostor Linuxa koji će unutar sebe imati i mrežni izolirani sustav (*Network namespace*).

```
unshare --net --uts /usr/bin/bash
```



Važno je razumjeti da čim zatvorimo inicijalni terminal iz kojega smo kreirali ove **UTS** i **Network** *namespace*-ove, oni će se zavvoriti. Stoga, ako ih želimo da nasave raditi kao servisi (u pozadini), možemo ih pokrenuti ovako:

```
unshare --net --uts /usr/bin/bash &
```

Sada pogledajmo što smo sve dobili:

```
lsns -t net -t uts
        NS TYPE NPROCS   PID  USER     NETNSID NSFS   COMMAND
4026531838 uts     126     1  root                    /usr/lib/systemd/systemd …
4026531840 net     126     1  root   unassigned       /usr/lib/systemd/systemd …
4026532191 uts       2  2334  root                    /usr/bin/bash ← Naš UTS NS
4026532192 net       2  2334  root   unassigned       /usr/bin/bash ← Naš UTS NET
```

Vidimo da su se pojavili: novi *UTS namespace; TYPE:* `uts` i *Network namespace; TYPE:* `net` te da je unutar njih pokrenuta *bash* ljuska.
Također vidimo isti *PID* broj *bash* ljuske za oba *namespace-a*, to jest *PID* broj `2334`.
Sada ćemo unutar ovog izoliranog prostora koji čine *UTS namespace* i *Net namespace*, promijeniti ime računala:

```
nsenter -t 2334 -u hostname netns1
```

Potom na izvornom Linuxu kreirajmo par **VETH** mrežnih sučelja i to:

- Prvi dio *VETH* pâra (povezanih sučelja) s imenom **`veth1a`** će se vezati za inicijalnu instancu Linuxa.
- Drugi dio *VETH* pâra (povezanih sučelja) s imenom **`veth1b`** će se vezati za *Network namespace*.

Krenimo s kreiranjem *VETH* pâra sučelja:

```
ip link add veth1a type veth peer name veth1b
```

I potom aktivirajmo sučelje **`veth1a`**:

```
ip link set veth1a up
```

A zatim sučelje **`veth1b`** ubacimo u izolirani mrežni prostor (*Network namespace*), prema *PID 2334* broju:

```
ip link set veth1b netns 2334
```

Sada uđimo u mrežni izolirani prostor:

```
nsenter -t 2334 -n
```

I pogledajmo sva mrežna sučelja koja u njemu imamo:

```
ip -br link show
lo              DOWN  00:00:00:00:00:00 <LOOPBACK>
veth1b@if6      DOWN 26:c2:03:73:eb:5c <BROADCAST,MULTICAST,DOWN,LOWER_UP>
```

Ovdje vidimo naše *VETH* sučelje **`veth1b`**, međutim moramo pokrenuti sva mreža sučelja vidljiva ovdje:

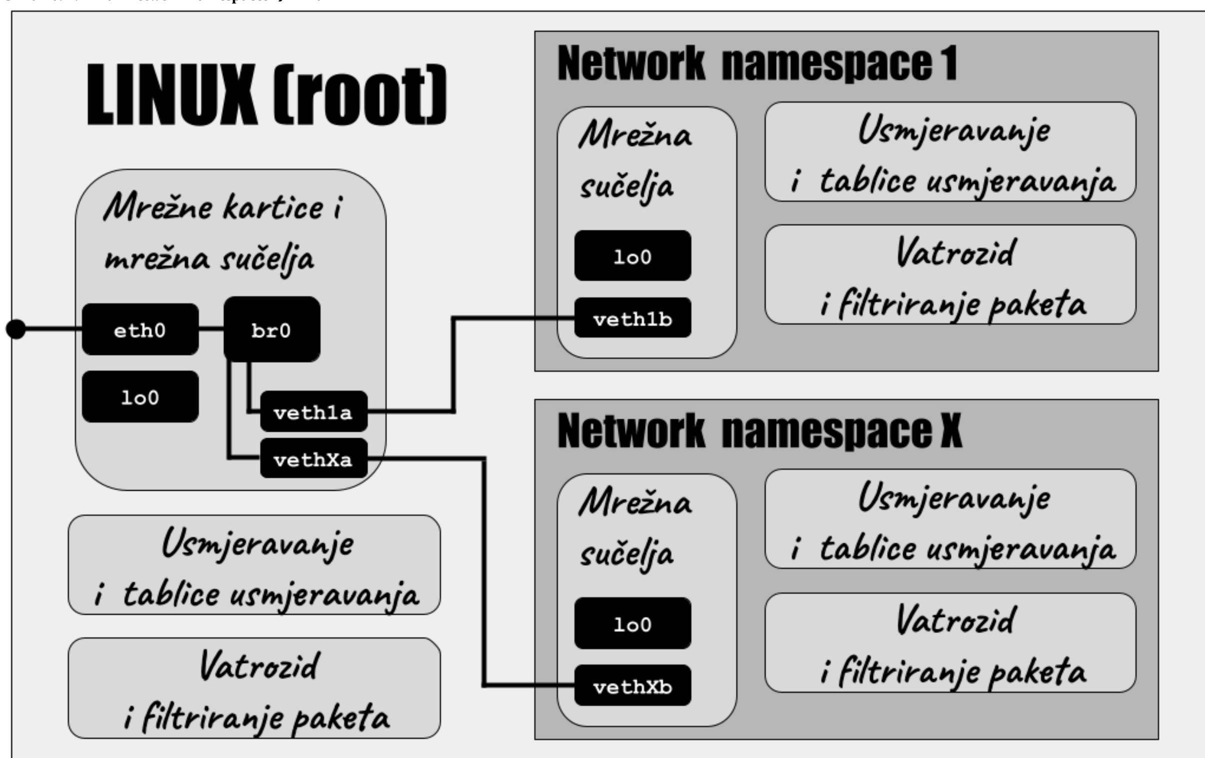
```
ip link set veth1b up
ip link set lo up
```

I sada su oba sučelja aktivna:

```
ip -br link show
lo               UNKNOWN         00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
veth1b@if6       UP              26:c2:03:73:eb:5c <BROADCAST,MULTICAST,UP,LOWER_UP>
```

Sada možemo dodijeliti IP adresu tom sučelju, međutim moramo biti svjesni da su ovo sučelje, kao i njegova buduća IP adresa i dalje izolirani unutar ovog *Network namespace-a*. Da bi to riješili, na razini vršnog (roditeljskog) Linuxa, potrebno je povezati drugi dio *VETH* para sučelja (**`veth1a`**) s našom mrežnom karticom (ako želimo komunikaciju od OSI sloja dva), i to preko *Bridge* mrežnog sučelja. Ili napraviti neku drugu vrstu veze preko koje će teći komunikacija, jer se nalazimo u izoliranom mrežnom prostoru. Ovakva logička shema s upotrebom ***Bridge*** i ***VETH*** mrežnih sučelja je vidljiva na slici **40.A.**

**Slika 40.A. Linux network namespaces → Linux**






Za druge primjere o mrežnim izoliranim prostorima Linuxa pogledajte poglavlje:
**20.6.4. VETH - posebno mrežno sučelje.**

**Vezano za mrežni izolirani prostor Linuxa, postoji i još jedna varijacija u izvedbi!**

Osim primjera koje smo naveli s naredbom `unshare`, moguće je koristiti i naredbu `ip`, koja dolazi u softverskom paketu `iproute`. Pomoću nje je također moguće kreirati mrežne izolirane prostore Linuxa, ali na malo drugačiji način (iako se u pozadini pokreću isti mehanizmi). Kreirajmo jedan mrežni izolirani prostor Linuxa iz vršnog (roditeljskog [*root*]) Linuxa:

```
ip netns add netnstest
```

Međutim, ovako kreirani mrežni izolirani prostor Linuxa nije vidljiv preko `lsns` naredbe:

```
lsns -t net
```

On je kreiran na malo drugačiji način, tako što će se za njega unutar direktorija: `/var/run/netns/`, pojaviti nova datoteka čije ime će biti identično ovom mrežnom izoliranom prostoru Linuxa (koji smo upravo kreirali). Pogledajmo ovu datoteku:

```
ls -alhi /var/run/netns/netnstest
4026532250 -r--r--r-- 1 root root 0 Apr 21 11:05 /var/run/netns/netnstest
```

U prvom stupcu vidimo njen *INODE* broj (`4026532250`).

Nakon što je mrežni izolirani prostor Linuxa kreiran, možemo ući u njega pokretanjem *bash* ljuske u njemu:

```
nsenter --net=/var/run/netns/netnstest /usr/bin/bash
```

ili sve operacije unutar njega možemo raditi pomoću naredbe `ip`.

***U našem slučaju ne pokrećite prethodnu naredbu!***

Sada ćemo u vršnom (roditeljskom [*root*]) Linuxu kreirati *VETH* pâr sučelja:

```
ip link add veth1a type veth peer name veth1b
```

I potom aktivirajmo sučelje (**`veth1a`**):

```
ip link set veth1a up
```

A zatim drugi dio pâra *VETH* sučelja (**`veth1b`**) dodajmo u novi mrežni izolirani prostor Linuxa (**`netnstest`**):

```
ip link set veth1b netns netnstest
```

Prema potrebi možemo ući u mrežni izolirani prostor Linuxa i u njemu pokrenuti *bash* ljusku (to nećemo napraviti):

```
ip netns exec netnstest bash
```

Ili možemo iz vršnog (roditeljskog) Linuxa pokretati željene naredbe unutar mrežnog izoliranog prostora Linuxa, što ćemo i učiniti. Dakle u ovom koraku ćemo aktivirati mrežna sučelja unutar mrežnog izoliranog prostora Linuxa (**`netnstest`**):

```
ip netns exec netnstest ip link set veth1b up
ip netns exec netnstest ip link set lo up
```

I sada pogledajmo sva mrežna sučelja unutar novog mrežnog izoliranog prostora Linuxa (**`netnstest`**):

```
ip netns exec netnstest ip -br link show
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
veth1b@if6       UP             06:3e:6c:25:bb:62 <BROADCAST,MULTICAST,UP,LOWER_UP>
```

Vidimo *loopback* sučelje (`lo`) te *VETH* sučelje (`veth1b`), koja su oba aktivna. A sada ćemo *VETH* sučelju dodijeliti IP adresu

```
ip netns exec netnstest ip addr add 10.10.1.2/24 dev veth1b
```

A iz istog opsega IP adresa ćemo u vršnom (roditeljskom [*root*]) Linuxu, na njegovom drugom kraju pâra *VETH* sučelja (`veth1a`) također dodijeliti IP adresu:

```
ip addr add 10.10.1.1/24 dev veth1a
```

I sada iz mrežnog izoliranog prostora Linuxa (**`netnstest`**) probajmo *ping* prema vršnom (roditeljskom [*root*]) Linuxu i *VETH* sučelju (`veth1a`):

```
ip netns exec netnstest ping -c3 10.10.1.1

PING 10.10.1.1 (10.10.1.1) 56(84) bytes of data.
64 bytes from 10.10.1.1: icmp_seq=1 ttl=64 time=0.061 ms
64 bytes from 10.10.1.1: icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from 10.10.1.1: icmp_seq=3 ttl=64 time=0.033 ms

--- 10.10.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2063ms
rtt min/avg/max/mdev = 0.033/0.044/0.061/0.014 ms
```

← Vidimo da je sve u redu (`3 packets transmitted`, `3 received`, `0% packet loss`).
Sada još unutar mrežnog izoliranog prostora Linuxa možemo dodati zadani usmjerivač (*default gateway*):

```
ip netns exec netnstest ip route add default via 10.10.1.0
```

Nakon ovog koraka, za komunikaciju s vanjskim svijetom treba riješiti ***Bridge*** sučelje, eventualno **NAT** te pravila vatrozida.

**PID namespace (`pid`)**

Upotrebom izoliranog *PID* prostora Linuxa postalo je moguće napraviti ugniježđena stabla procesa. To omogućuje procesima koji nisu ***systemd*** ili ***init*** (koji imaju PID=1) da sebe percipiraju kao korijenski proces pomicanjem na vrh svog podstabla, čime oni dobivaju PID=1 u tom podstablu. Nadalje Svi procesi u istom podstablu također će dobiti *PID* brojeve u odnosu na svoju poziciju unutar imeničnog (izoliranog) *PID* prostora. To također znači da neki procesi mogu imati višestruke ID-ove (*PID* brojeve) ovisno o izoliranom *PID* prostoru procesa u kojem se nalaze. Ipak, u svakom izoliranom *PID* prostoru procesa najviše jedan proces može imati određeni *PID* broj jer je on jedinstveni identifikator svakog procesa u stablu procesa. Time odnosi između procesa u izoliranom *PID* prostoru ostaju netaknuti.


Pogledajte i poglavlje: **27.2.2. Podman Linux kontejneri →** dio s **PID** i **GID** izolacijama.

Odnosno proces u novom *PID* izoliranom prostoru još uvijek je pripojen svom roditeljskom (unutar vršnog [*root*]) Linuxa), stoga je dio svog nadređenog *PID* imenskog prostora. Ovi odnosi između svih procesa mogu se vidjeti u prostoru imena korijenskog (roditeljskog) procesa, ali u *PID* izoliranom prostoru ugniježđenog procesa nisu vidljivi. Definicija opsega podređenih *PID* i *GID* brojeva za imenske prostore definiraju se u datotekama: `/etc/subuid` i `/etc/subgid`, a inicijalno u `/etc/login.defs`. Pogledajmo kako kreirati izolirani *PID* prostor Linuxa i u njemu pokrenuti *bash* ljusku:

```
unshare --pid --fork  /usr/bin/bash
```

Pogledajmo PID brojeve procesa unutar ovog *PID* izoliranog prostora Linuxa (s naredbom `ps`):

```
ps
    PID TTY          TIME CMD
   1058 pts/0    00:00:00 bash
   2195 pts/0    00:00:00 unshare
   2196 pts/0    00:00:00 bash
   2225 pts/0    00:00:00 ps
exit
```

Ovdje i dalje vidimo *PID* brojeve koji su zapravo iz prostoru imena korijenskog (roditeljskog) procesa vršnog [*root*]) Linuxa. To je stoga jer program/naredba `ps` koristi virtualni datotečni sustav *procfs* za dobivanje informacija o trenutnim procesima u sustavu, a ovaj datotečni sustav je montiran u `/proc` direktoriju. Međutim, u novom *PID* prostoru Linuxa ova točka montiranja i dalje opisuje procese iz korijenskog prostora imena *PID* brojeva.

**Mount namespace (`mnt`)**

Izolirani prostor za montiranje kontrolira točke montiranja. Tijekom stvaranja izoliranog prostora za montiranje, montiranje iz trenutnog prostora (obično roditeljskog odnosno vršnog [*root*]) Linuxa) kopiraju se u novi izolirani prostor (tzv. *shared* način rada), ali točke montiranja stvorene nakon toga ne šire se između izoliranih prostora. S obzirom na stanje iz prethodnog primjera, morat ćemo uvesti mehanizam montiranja zasebnog *procfs* datotečnog sustava, što ćemo postići sa:

```
unshare --fork --pid --mount-proc  /bin/bash
```

Pogledajmo stanje sada, ponovno s naredbom `ps`:

```
ps
    PID TTY          TIME CMD
      1 pts/0    00:00:00 bash
     27 pts/0    00:00:00 ps
```

← Sada konačno vidimo samo izolirano stablo *PID* brojeva unutar našeg izoliranog *PID* prostora.
Ako istovremeno otvorimo novi terminal unutar vršnog [*root*]) Linuxa te pokrenemo:

```
ps -ef | grep unshare

root   2283   1058  0 12:17 pts/0  00:00:00 unshare --fork --pid --mount-proc /bin/bash
```

← Vidimo da je stvarni *PID* broj `2283`, koji se očito mapira u *PID* `1` unutar našeg izoliranog PID prostora.

Vezano za *mount namespace*, on nam dodatno omogućuje i stvaranje privatnog izoliranog privremenog datotečnog sustava koji čak ni drugi procesi u vlasništvu vršnog Linuxa ne mogu vidjeti ili pregledavati.

*Proučite i takozvani* ***bind mount*** [`man mount`]



Pogledajte poglavlje: **13.7. Kako se montiraju particije u direktorije i zašto.**

**User namespaces (`user`)**

Svi pokrenuti procesi u Linux i Unix svijetu imaju svog vlasnika (kao i datoteke i direktoriji). Postoje privilegirani i neprivilegirani procesi ovisno o njihovom efektivnom korisničkom ID-u (tzv. **UID**). Ovisno o ovom UID-u, procesi imaju različite privilegije u odnosu na operativni sustav. U izoliranim prostorima Linuxa moguće je izolirati i korisničke ID oznake odnosno ***UID*** (i oznake grupe [**GID**]). Pokrenimo jednu takvu izolirano okolinu:

```
unshare --fork --pid --mount-proc --mount  --user /bin/bash
```

Ako sada u ovom izoliranom prostoru s naredbom `id` pogledamo naše **UID** i **GID** oznake, vidjet ćemo:

```
id
uid=65534(nobody) gid=65534(nobody) groups=65534(nobody)
```

← Vidimo da je naš **UID** 65534 (nepostojeći korisnik) uz nepostojeću grupu (GID=65534), što je za sada normalno.

Sada automatski definirajmo mapiranje (stvarni **UID**→ izolirani **UID** te stvarni **GID**→ izolirani **GID**):

```
unshare --fork --pid --mount-proc --mount  --user --map-root-user /bin/bash
```

Ako sada u ovom izoliranom prostoru s naredbom `id` pogledamo naše **UID i GID** oznake će biti:

```
id
uid=0(root) gid=0(root) groups=0(root)
```

← Dakle *UID* i *GID* oznake to jest brojevi su uredno skrivene unutar izolacije. Stvarne *UID* i *GID* oznake koje se mapiraju s onima u izoliranom prostoru su one s kojima je inicijalizirana naredba `unshare` odnosno izolirani prostor Linuxa.

Naredba `unshare` dolazi u softverskom (programskom) paketu imena: `util-linux`.

**Izvori informacija: (316),(317),(318),(319),(745),(746),(1473),(1474),(1475),(K-4),(K-5),** **`man lsns`,`man subuid`** **`man nsenter`,`man 7 namespaces`,`man unshare`,`man ip`,`man ip-netns`,`man ps`,`man mount`,** **`man mount_namespaces`,`man pid_namespaces`,`man network_namespaces`,`man user_namespaces`.**

### 9.4.6. Secure Computing Mode

*Secure Computing Mode* (**seccomp**) značajka je Linux kernela koja omogućuje ograničavanje sistemskih poziva sustava koje aplikacije mogu koristiti. S ovim mehanizmom ograničavamo aplikacije (procese) na ograničen skup sistemskih poziva, a ako proces pokuša pozvati bilo koji drugi sistemski poziv koji mu nije odobren/dozvoljen, kernel će ga prekinuti koristeći signale `SIGKILL` ili `SIGSYS`.
Zbog svojih mogućnosti ***seccomp*** se obično koristi u izoliranim okruženjima poput Linux kontejnera kao što su ***Docker*** i ***Podman*** i ***LXC***, ali i unutar **systemd** servisa. Osim toga može se koristiti upotrebom ***kubernetes*** orkestratora pa čak i unutar internetskih preglednika (kao što su ***Google Chrome*** i ***Mozilla Firefox***) kao dodatni sloj sigurnosti. Na taj način *seccomp* može spriječiti zlonamjerne aplikacije u iskorištavanju ranjivosti sustava ili dobivanja neovlaštenog pristupa resursima. Također, može se koristiti za ograničavanje mogućnosti aplikacije da pristupi određenim resursima sustava, primjerice mreže ili datotečnog sustava.
Od Linux kernela 3.17 uveden je novi sistemski poziv `seccomp(2)`, a može se koristiti i `prctl(2)` s kojima se barata s ovim mehanizmima na najnižoj razini. Od kernela 4.14 također postoji "`/proc`" sučelje za *seccomp* koje se nalazi unutar vršnog direktorija: `/proc/sys/kernel/seccomp/`. Tamo možemo pronaći datoteke:

- `actions_avail` – s popisom samo za čitanje, povratnih akcija *seccomp* filtra koje podržava kernel.
- `actions_logged` – s popisom za čitanje i pisanje radnji *seccomp* filtra koje je dopušteno bilježiti.

Nadalje, `seccomp-bpf` je ekstenzija za *seccomp* mehanizam koja dopušta filtriranje sistemskih poziva korištenjem konfigurabilne politike implementirane pomoću pravila *Berkeley Packet Filter* sustava (**eBPF**). Nju primjerice koriste *OpenSSH* (SSH servis) i *vsftpd* (FTP servis), kao i neki web preglednici.



Vezano za **sistemske signale** i pozive pogledajte poglavlja:
**9.2. Procesi i signali koje im možemo poslati.**
**9.2.1. Napredno o procesima i signalima.**

## Seccomp i systemd

*Seccomp* filtere je moguće integrirati unutar željenih *Systemd* servisa omogućujući jednostavno ograničavanje sistemskih poziva za željene servise definiranjem pravila (filtera) u konfiguracijskim datotekama **systemd**-a (tzv. *unit files*).

**Uređivanje konfiguracijske datoteke željenog *systemd* servisa (*unit file*)**
Pronađimo konfiguracijsku datoteku jedinice za servis koju želimo ograničiti.
Ove datoteke se obično nalaze u direktorijma: `/etc/systemd/system/` ili `/usr/lib/systemd/system/`.
Dodavanje *seccomp* filtera (pravila) radi se dodavanjem neke od sljedećih direktiva unutar sekcije `[Service]`:

- `SystemCallFilter` – ova opcija omogućava filtriranje sistemskih poziva koje proces može ili ne može izvršavati. Možete definirati dozvoljene ili zabranjene sistemske pozive kako biste smanjili sigurnosne rizike. Sintaksa može biti jednostavna, kao lista poziva, ili složenija s prefiksima poput `~` za zabranu određenih grupa poziva. Primjerice:
  ```
  SystemCallFilter=~@network
  ```
  ← Ova postavka zabranjuje sve sistemske pozive vezane za mrežu odnosno za korištenje mreže.
  Pogledajmo listu nekih od lista (grupa) poziva koje ovdje možemo koristiti:
  - `@basic-io` – sistemskih pozivi za osnovne ulazno-izlazne operacije (čitanje, pisanje, otvaranje datoteka, itd.).
  - `@file-system` – sistemskih pozivi vezani za datotečni sustav (obuhvaća pozive poput: `mkdir`, `rmdir`, `link`, `unlink`, `rename`, `symlink`, …).
  - `@network-io` – sistemskih pozivi za određene grupe mrežnih operacija (obuhvaća pozive poput: `socket`, `bind`, `connect`, `listen`, `accept`, `sendto`, `recvfrom`).
  - `@mount` – sistemskih pozivi vezani uz montiranje i demontiranje datotečnih sustava. . . .
- `SystemCallErrorNumber` – ova opcija omogućava postavljanje specifičnog broja greške (`errno`) koji će biti vraćen kada se pokuša izvršiti zabranjeni sistemski poziv. Koristi se u kombinaciji sa `SystemCallFilter` za definiranje što će se dogoditi kada proces pokuša izvršiti zabranjeni sistemski poziv. Zadana vrijednost je `EPERM` (*Operation not permitted*). Primjerice:
  ```
  SystemCallErrorNumber=EPERM
  ```
  ← Ova postavka definira da, ako se pokuša izvršiti zabranjeni sistemski poziv, bit će vraćena greška EPERM.
- `SystemCallArchitectures` – ova opcija određuje koje arhitekture sistemskih poziva su dozvoljene za proces. Korisno je za ograničavanje izvršavanja sistemskih poziva na određene arhitekture, čime se dodatno poboljšava sigurnost sustava. Primjerice:
  ```
  SystemCallArchitectures=native
  ```
  ← Ova postavka ograničava izvršavanje sistemskih poziva samo na arhitekturu na kojoj je sustav pokrenut.



Vezano za ***Systemd*** i ***Unit*** datoteke, kao i druge detalje o **systemd**, pogledajte poglavlja:
**7.3.2.** *Systemd* i sistemski servisi (*daemoni*)**.**
**7.3.2.1. *Unit* datoteke.**
**7.3.2.4. Kreiranje vlastite Systemd servisne skripte (sistemskog servisa).**
**7.3.2.6. Ograničenja resursa Systemd servisa.**

## Seccomp i Docker, Podman i LXC

*Seccomp* filtere je moguće koristiti i za pokretanje željnih Linux kontejnera poput *Docker* i *Podman* ili *LXC*.

**Podman**
Za *Podman* kontejnere za upotrebu *seccomp* značajki potrebno je kreirati seccomp profil. *Seccomp* profil je JSON datoteka koja definira pravila za sistemske pozive. Jednostavni primjer ovakvog *seccomp* profila jest:

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "syscalls": [
    {
      "names": ["exit", "exit_group", "read", "write"],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

Ovakav *seccomp* profil potrebno je spremiti u JSON datoteku, pr. imena `seccomp.json`.
Nakon toga je potrebno pokrenuti željeni *Podman* kontejner; primjerice ovako:

```
podman run --rm --security-opt seccomp=/putanja/seccomp.json -it alpine sh
```



Vezano za *Podman* kontejnere pogledajte poglavlje:
**27.2.2. Podman Linux kontejneri.**

**Docker kontejneri**
Za *Docker* kontejner se također uobičajeno koristi JSON definicija odnosno *seccomp* profil koji je potrebno postaviti u zasebnu datoteku. Nakon toga se *Docker* kontejner poziva primjerice ovako:

```
docker run --security-opt seccomp=/path/to/custom/profile.json <container>
```

**LXC kontejneri**
Za *LXC kontejnere* se uobičajeno ne koristi JSON definicija odnosno *seccomp* profil već je unutar konfiguracijske datoteke LXC kontejnera potrebno dodati novi redak sa direktivom (`lxc.seccomp.profile`), poput:

```
lxc.seccomp.profile = JSON-profil
```

Konkretno, primjerice poput ove:

```
lxc.seccomp.profile = /etc/lxc/seccomp/seccomp-profile.json
```

I nakon toga je potrebno pokrenuti LXC kontejner.



Vezano za *LXC* kontejnere pogledajte poglavlje:
**27.2.1. Konfiguracija i upotreba Linux kontejnera.**

## Upotreba seccomp prilikom pisanja vlastitog programskog koda

Ako sami pišete svoj programski kod *seccomp* značajke mogu se koristiti upotrebom programske biblioteke `libseccomp`.
Programsku biblioteku `libseccomp` je moguće instalirati na sljedeći način:

```
yum install libseccomp libseccomp-devel
```

**Izvori informacija: (1662),(1663),(1664),(1665),(1666),(1667),** **`man 2 seccomp`**,**`man 2 prctl`**.

# 9.5. Linux poslovi (*jobs*)

U normalnom radu na *Unix* ili *Linux* operativnim sustavima u terminalu (ljusci/*shell*u), obično kao korisnici pokrećemo jednu po jednu naredbu, odnosno imamo obično pokrenut sâmo jedan program (proces). Dakle svakodnevne radnje u terminalu poput pokretanja naredbi za izlistanje direktorija: `ls -al` uređivanje datoteka; primjerice s programom `vi` i slično, rade na način u kojem program odnosno naredba koju pozivamo preuzima kontrolu nad našim terminalom te vraća kontrolu ljusci (*shell*u), tek kada se program završi. Na nižoj razini, radi se o preuzimanju kontrole nad standardnim ulazom (*stdin*) i standardnim izlazom (*stdout*), što znači preuzimanje kontrole nad pisanjem ili čitanjem na terminal, a po završetku prepuštanju istih (*stdin* i *stdout*) *shell*u odnosno naredbenoj ljusci. U određenim situacijama, ipak imamo potrebu, pokrenuti neki program, ali i istovremeno započeti s radom na drugom programu u istom terminalu.

Stanje u kojem pokrenuti program prebacujemo na rad u pozadini, u kojoj on nije u mogućnosti primati ulaz iz našeg terminala i naravno tipkovnice, naziva se rad u pozadini tj. Engleski ***Background***. Dok program u kojem radimo u terminalu, i koji prima ulaz s terminala (pr. unos s tipkovnice) radi u tzv. ***Foreground*** načinu rada. Unix odnosno Linux poslovi (engl. *Jobs*) omogućavaju nam upravo ove mogućnosti; programe možemo pokretati u našoj ljusci (*Shell*u): to su *foreground* poslovi, a možemo ih, u bilo kojem trenutku privremeno zaustaviti, prebaciti na rad u pozadini (*background*) ili one privremeno zaustavljene možemo ponovno pokrenuti.

Kontrola poslova odnosno „*jobova*" je definirana kao mogućnost selektivnog zaustavljanja (engl. *Suspend*) izvršavanja željenog procesa uz mogućnost njegovog kasnijeg nastavljanja s izvršavanjem (engl. *Resume*). Dakle možemo reći kako za bilo koji pokrenuti:

- Interaktivni program.
- *Shell* skriptu.
- Ili nîz naredbi koje smo povezali sa *pipe*om ( | ) možemo promijeniti stanje rada (*job control*), kako smo naveli.

Provjeru stanja Linux poslova (*job*ova) možemo napraviti korištenjem naredbe `jobs`.
U ovakvom radu, poslovi (*jobs*) mogu biti u sljedećim stanjima:

- ***Foreground*** - ovo je standardno stanje izvršavanja programa s kojim smo u interakciji [označava se kao `fg` ].
- ***Background*** - pokretanje u pozadini ili ako se pokreće sa znakom **&** na kraju [označava se kao `bg` ].
- ***Stopped*** - zaustavljen, ali se može ponovno pokrenuti u *Foreground* ili *Background* stanju.
- ***Terminated*** - zaustavljen trajno odnosno terminiran (ubijen).

**Malo detaljnije o Unix/Linux poslovima (*job*ovima)**
Za standardno razumijevanje linux poslova (*job*ova) potrebno je promatrati ih kao pojedine pokrenute programe ili naredbe odnosno procese. Ali pošto svaki Unix i Linux podržava povezivanje naredbi preko *pipe*-a ( | ) sada dobivamo malo drugačiju situaciju. Naime povezivanjem naredbi preko *pipe*-a, povezujemo više naredbi ili programa, koji komuniciraju međusobno.
U takvom radu se rezultat prve naredbe prosljeđuje kao ulaz drugoj, a njen rezultat (izlaz) se prosljeđuje trećoj naredbi, kao ulaz i tako dalje. Dakle u jednoj liniji ili prolazu, pokreće se više zasebnih programa ili naredbi (procesa), a s kojima *job control* mehanizam nekako mora znati raditi. Pogledajmo slijedeći primjer ovakvog povezivanja naredbi pomoću *pipe-a*:

```
cat imena-studenata.txt | grep pero
```

Ovdje smo s naredbom `cat` ispisali sadržaj datoteke: `imena-studenata.txt` u kojoj imamo popis studenata koji su se prijavili na predavanje, a potom pomoću *pipe*-a, ispis prosljeđujemo naredbi `grep`, koja zatim traži ključnu riječ odnosno pojam: `pero`.
U ovom primjeru imat ćemo pokrenuta dva programa/naredbe odnosno procesa; ako gledamo s razine operativnog sustava:

1. Prvi proces je naredba `cat` koja otvara tekstualnu datoteku imena: `imena-studenata.txt`.
2. Drugi proces je naredba `grep` koja pretražuje ispis prve naredbe i traži pojam: `pero`.

Međutim oba se procesa pokreću iz jedne (iste) naredbene linije i logički su jedna cjelina, pa bi se trebali i ponašati kao jedan entitet, odnosno u ovoj priči, kao jedan Linux posao (engl. *Linux Job*).

**Još detaljnije**
Prema **POSIX** standardu, kojeg se pridržava i Linux, svaka nova ljuska preuzima kontrolu nad tzv. sesijom (engl. *Session***)** i procesnom grupom (engl. *Process group***)**, te preuzima kontrolu nad *procesnom grupom* sâmog **terminala** na kojem radimo. Naime ljuska kreira novu *procesnu grupu* unutar svoje *sesije*, za svaki posao (*job)* koji se pokreće. U slučajevima kada se primjerice koristi *pipe* mogućnost, te se u jednom nîzu izvršava više programa ili naredbi, ljuska se brine o tome kako bi se kod inicijalizacije, dogodilo sljedeće:

1. Kreirala se nova *procesna grupa*.
2. Sve naredbe ili programi koji se izvršavaju u *pipe* nîzu, se moraju ubaciti u tu istu *procesnu grupu*.

Ovo je osnovni mehanizam, kako bi uopće bio moguć rad Linux poslova (*job*ova). U gornjem primjeru, kada se kreira novi nîz naredbi povezanih *pipe*om odnosno cjevovodom (*pipeline*), kreira se i nova *procesna grupa* u kojoj se izvršavaju konkretne dvije naredbe (dva različita procesa) i to: `cat` i `grep`, a koje se kontroliraju iz jedne jedine *procesne grupe*, koju kontrolira *job control* mehanizam.



U slučajevima kada jedan program odnosno proces pokreće neki novi program odnosno proces, tada novi program/proces pripada istoj procesnoj grupi kao proces koji ga je pokrenuo (kao roditeljski proces).

U primjerima koji slijede bit će vam jasno što su Linux poslovi i kako se barata s njima.

**Primjeri:**
Kreirajmo datoteku koja će se popunjavati slučajnim znakovima, pomoću naredbe dd na sljedeći način:

```
dd if=/dev/urandom of=/root/slucajni.znakovi.txt
```

Dakle naredba dd koristi kao ulaz (prekidač: if=) generator slučajnih znakova (slova, brojeva, itd.) koji predstavlja posebna datoteka (/dev/urandom). S druge strane naredba dd te sve slučajne znakove kao izlaz (prekidač: of=) zapisuje u datoteku imena: /root/slucajni.znakovi.txt.

Zaustavimo ovaj posao (*job*) kreiranja datoteke sa kombinacijom tipki: CTRL z pa ćemo dobiti sljedeći ispis:

```
CTRL z
[1]+ Stopped dd if=/dev/urandom of=/root/slucajni.znakovi.txt
```

Sada smo dobili poruku da je ovaj posao (zadatak) zaustavljen (Stopped).
Vidljivo je kako ovaj posao im identifikacijski broj odnosno ***ID***: [**1**].
Ovaj broj [1] se odnosni na identifikacijski broj Linux posla (zadatka)!

Pošto smo ga zaustavili, možemo ga u bilo kojem trenutku ponovno pokrenuti u pozadini (engl. *Background*), s naredbom bg, iza koje slijedi identifikacijski broj posla (*joba*). U našem slučaju je to broj **1**.
Sada ga ponovno pokrenimo u pozadini:

```
bg 1
```

I na kraju, ponovno, u bilo kojem trenutku, možemo ga ponovno prebaciti u normalan rad, kao da ga nismo niti zaustavljali i ponovno pokretali (*foreground*). Za ovu namjenu koristimo naredbu: fg iza koje mora biti upisan identifikacijski broj posla (*joba*). To ćemo napraviti na sljedeći način:

```
fg 1
```

Sada ćemo ga trajno zaustaviti sa kombinacijom tipki:

```
CTRL c
```

**Pokretanje naredbi u pozadini**
Istu naredbu (dd) smo mogli odmah pokrenuti u pozadini, te nastaviti raditi u našoj ljusci. Pošto bi ova naredba nastavila raditi u pozadini beskonačno dugo i kreirala datoteku koja bi postajala sve veća te bi u konačnici prepunila disk, ipak ćemo ju malo ograničiti.
Naredbi dd ćemo stoga dodati par prekidača koji će ograničiti veličinu datoteke koja će biti kreirana.
Dodat ćemo joj prekidač bs= koji nam govori kolika će biti veličina blokova podataka u kojim će se snimati podaci u datoteku.
Mi ćemo definirati ovu veličinu kao 1MB.
Dodatno ćemo definirati koliko puta po toj jediničnoj veličini (bs) će se ponavljati zapisivanje.

Za to koristimo: count= i reći ćemo **1**. Dakle 1MB (blok [bs=]) x 1 (broj ponavljanja [count=]) = 1MB ukupno.

```
dd if=/dev/urandom of=/root/slucajni.znakovi.txt bs=1M count=1 &
```

Znak & na kraju znači kako smo naredbu (ili nîz naredbi) pokrenuli u pozadini, praktično kao servis.



Pogledajte poglavlje vezano za meta znakove (posebno obratite pažnju na meta znak **&**) :
**5.11. Meta znakovi.**



Pogledajte i sljedeća poglavlja (u njima posebno obratite pažnju na upotrebu meta znaka **&**):
**6.7. Nizanje i ulančavanje naredbi i statusni kodovi.**
**9.2. Procesi i signali koje im možemo poslati.**
**9.4.2. Named pipe.**

**Izvori informacija: (K-12) te ugrađene naredbe u *bash* ljusku: `man jobs`, `man bg`, `man fg`.**

# 9.6. Procesne grupe

***Slijedi napredno poglavlje!***
Svaki proces je član procesne grupe odnosno skupine, koja čini zbirku jednog ili više procesa koji su općenito povezani jedni s drugima u svrhu kontrole poslova (*jobs*). Primarni atribut procesne skupine jest taj da se signali mogu slati svim procesima u skupini odnosno grupi: jedna radnja može prekinuti, zaustaviti ili nastaviti sve procese u istoj skupini odnosno grupi procesa. Svaka procesna grupa identificirana je *ID*-om procesne grupe (`pgid`) i ima voditelja procesne grupe. Identifikator (*ID)* procesne skupine jednak je ***PID*** broju voditelja procesne skupine.

Procesne skupine postoje sve dok u njoj postoji minimalno još jedan član. Čak i ako voditelj procesne skupine završi s radom, procesna skupina nastavlja postojati. Kada se novi korisnik prijavi na računalo, postupak prijave stvara novu sesiju koja se sastoji od jednog procesa i korisničke ljuske za prijavu.

Administrator za prijavu procesne skupine funkcionira kao voditelj cijele sesije. *ID* voditelja sesije koristi se kao *ID* sesije (procesne skupine). Sesija je pri tome zbirka jedne ili više procesnih skupina. Sesije organiziraju aktivnosti prijavljenih korisnika i povezuju tog korisnika s kontrolnim terminalom, koji predstavlja specifični `tty` uređaj, a koji obrađuje korisnikov ulaz odnosno izlaz na terminal. Slijedom toga, sesije su uglavnom nîzovi ili skupine procesa.



Prema ***POSIX*** standardu, a koji prati i Linux, procesna skupina ili grupa označava zbirku jednog ili više procesa. Između ostalog, procesna se skupina koristi za kontrolu distribucije signala sustava: kada je signal usmjeren na procesnu skupinu, signal se dostavlja svakom procesu koji je član procesne skupine.

Naime kada neki program/proces kreira svoj pôd proces, događa se sljedeće:

1. Roditeljski proces ima svoj ***PID broj*** u trenutku kada se pokrene te postavlja i ***PGID*** broj jednak svom ***PID*** broju (***PID=PGID***)
2. Pôd proces dobiva svoj novi ***PID*** broj.
3. Pôd proces dobiva i ***PPID*** broj (*Parrent PID*/roditeljski *PID*) od procesa koji ga je pokrenuo (od roditeljskog procesa).
4. Pôd proces, kao i svi pôd, pôd procesi dobivaju isti zajednički ***PGID*** broj koji je postavio roditeljski (vršni) proces koji ih je pokrenuo, ako se logički nalaze unutar iste grupe procesa, dok u drugim slučajevima to ne mora biti slučaj.

**Kako vidjeti PGID broj procesa?**
***PGID*** broj se može vidjeti i pomoću naredbe `ps`.

Pogledajmo: ***PID***, ***PPID*** i ***PGID*** brojeve za sâmo nekoliko pokrenutih Linux procesa (skratili smo ispis):
**`ps -ae -o pid,ppid,pgid,command`**

```
  PID  PPID  PGID COMMAND
    1     0     1 /sbin/init
    2     0     0 [kthreadd]
    3     2     0 [migration/0]
    4     2     0 [ksoftirqd/0]
  414     1   414 /sbin/udevd -d
  723   414   414 /sbin/udevd -d
  732   414   414 /sbin/udevd -d
 1206     1  1206 /usr/sbin/sshd -D -o Ciphers=aes256… … …
 1255     1  1255 /sbin/mingetty /dev/tty1
 1290  1206  1290 sshd: root@pts/1
 1292  1290  1292 -bash
 1363  1292  1363 ps -ae -o pid,ppid,pgid,command
```

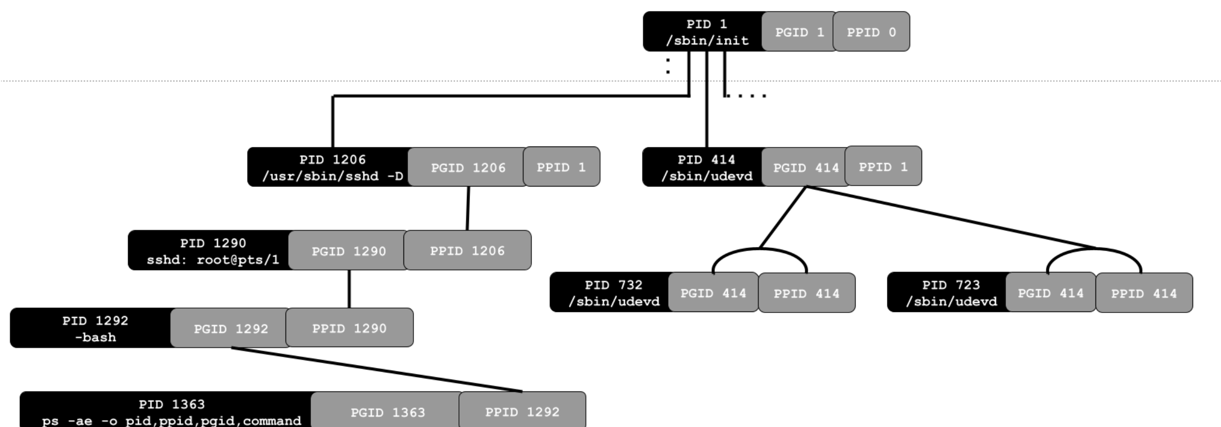
Ovdje je vidljivo kako je primjerice `init` proces sâm u svojoj procesnoj grupi (`PGID 1`) što znači kako su svi ostali procesi koje on pokreće u zasebnim grupama i ne ovise o njemu.

S druge strane imamo proces `kthreadd` koji je posebni kernel proces/*thread* a koji pokreće pôd procese: `migration/0,ksoftirqd/0` i druge, ali on je njima svima postavio ***PGID*** grupu `0`, kojoj je on voditelj. Isto tako imamo i slučaj sa `udevd` daemonom/servisom kojega je pokrenuo `init` proces.

To je stoga jer vidimo kako je prvom `udevd` procesu koji ima ***PID***=`414`, njegov ***PPID***=`1`; što znači kako mu je roditelj ***PID***=`1`, a to je `init` proces. Ovdje vidimo i kako svi ostali `udevd` servisi imaju kao roditeljski ***PID*** (*PPID*=`414`); dakle prvi `udevd` koji i ima ***PID***=`414`, a svima njima je grupni ***PID*** (*PGID*=`414`).
To opet znači kako im je voditelj procesne grupe onaj proces čiji ***PID*** broj jest `414`, a to je naš prvi `udevd` servis.

Pogledajmo i logičko stablo pokrenutih procesa, s naglaskom na **PID**, **PGID** i **PPID**, kako je vidljivo na slici **40.B.**

**Slika 40.B. Linux stablo procesa: pogled na PID, PGID i PPID.**




Na slici je jasno vidljiva veza između roditeljskih procesa i njihovih pod procesa, kao i njihovih procesnih (**PID**), grupnih (**PGID**) i roditeljskih identifikatora (**PPID**).

Osim navedene naredbe `ps`, s korištenim prekidačima, možemo koristiti istu naredbu kako bismo vidjeli stablo procesa, ali s dodatkom ***PGID*** brojeva, dakle sa sljedećom naredbom:

```
ps -ajxf
```



Moguće je i zaustavljati ili ubijati cijelu grupu procesa, korištenjem grupnog ***PID*** broja odnosno ***PGID***-a, slanjem nekog od sistemskih signala: *TERM* / *INT* / *QUIT* / *KILL* /…, kao da se radi o običnom ***PID*** broju.
Dakle moguće je slanje sistemskih signala na cijelu ***PGID*** (grupu) i sâmim time na sve njene članove.



Postoji i ***TGID*** broj (engl. *Thread Group ID*) koji koristi ***PID*** broj procesa, ako se radi o njegovoj procesnoj nîti, ali ako se radi o računju programa, tada nova instanca dobiva drugi ***PID*** i ***TGID=PID*** broj.



Pogledajte i poglavlje:
**10.7.2.1.2. Process/Thread Affinity i CPU Affinity.**



Za više detalja o procesima pogledajte i poglavlja:
**9. Proces menadžment.**
**7.3.3. Sistemski servisi (i procesi).**

**Izvori informacija: (295),(296),(297),(298),(299), (K-4), `man ps`.**

# 10. Upoznajmo se s našim računalom

Linux pristupa hardveru na vrlo niskoj razini te je za razumijevanje njegovog, ali i rada bilo kojeg drugog operativnog sustava, potrebno i razumijevanje rada sâmog računala. U poglavljima koja slijede upoznat ćemo se s osnovama arhitekture i rada računala.

## 10.1. Od čega se sastoji računalo

Svako računalo se sastoji od nekoliko osnovnih komponenti:

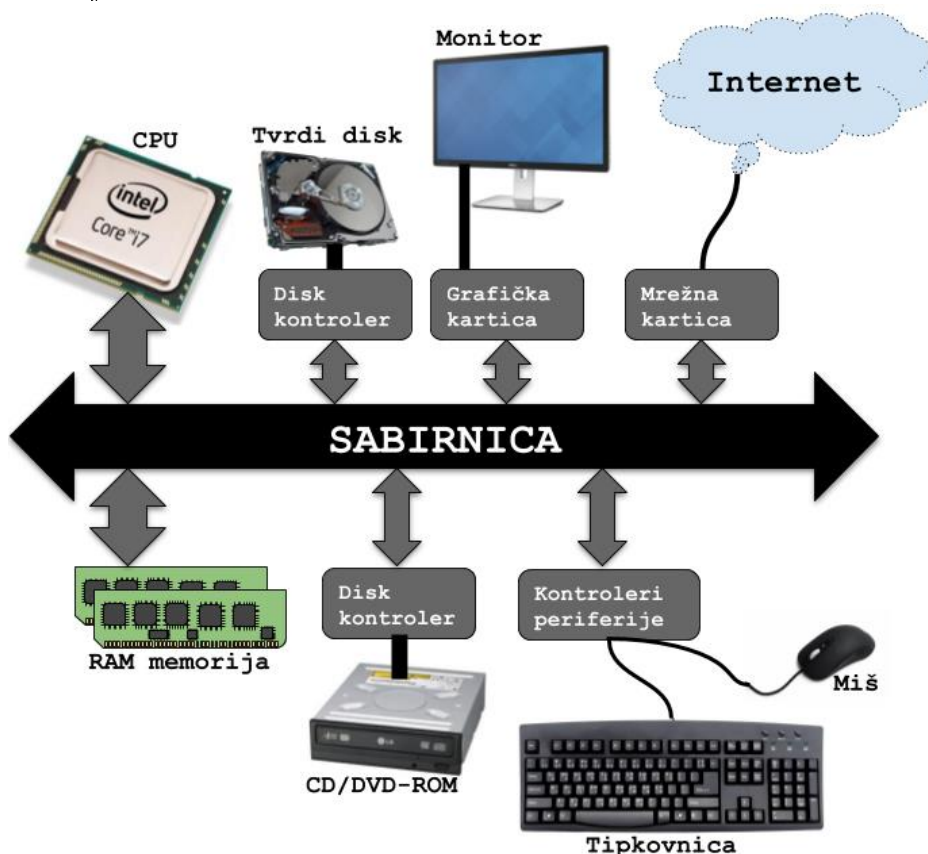
- Centralnog procesora (*CPU*) – za detalje pogledajte poglavlje: **10.7.**
- Primarne memorije s nasumičnim pristupom (*RAM* memorije) – za detalje pogledajte poglavlje: **12.**
- Tvrdog (mehaničkog/*rotacijskog*) ili *SSD* diska – za detalje pogledajte poglavlje: **13.**
- Mrežne i drugih kartica – za detalje pogledajte poglavlje: **19.**
- Optičkog uređaja (CD/DVD-ROM).
- Vanjskih uređaja: monitor, miš, tipkovnica, pisač, skener i slično.

Pri tome centralni procesor (*CPU*) odrađuje sve:

- Aritmetičke operacije poput: zbrajanja, oduzimanja, množenja, dijeljenja i drugih matematičkih operacija.
- Logičke operacije poput, logičkih: **I** (*AND*), **ILI** (*OR*), **NE** (*NOT*), **NI** (*NAND*), **NILI** (*NOR*), **XILI** (*XOR*), **XNILI** (*XNOR*).
- Kontrolne operacije, za koje je zadužen dio koji se zove "***Control unit***" (***CO***), a koji na osnovi programskih instrukcija nalaže svim ostalim jedinicama (aritmetičko-logičkoj, ulazno/izlaznoj i memoriji) što i kako da odrade.
- Ulazno/izlazne operacije.

Možemo reći da je *CPU* jezgra svakog računala. On je zadužen za pokretanje, rad i kontrolu cijelog sustava, počevši od njegovog učitavanja s tvrdog diska, *SSD* diska ili nekog drugog medija poput *CD/DVD-ROM*, *USB sticka* i sličnih, do učitavanja operativnog sustava sa svim upravljačkim programima, kao i pokretanja i rada svih programa odnosno aplikacija.

**Slika 41. Logička shema računala.**




Kao što je vidljivo iz slike **41.** sve komponente računala su povezane preko sabirnice.
Većina navedenih komponenti računala i sabirnica, nalaze se na matičnoj ploči računala (engl. *Motherboard*).
Druge komponente, poput tvrdog diska, CD/DVD-ROM, ali i drugih uređaja, poput grafičke ili mrežne kartice se spajaju zasebno, svaka na svoje sučelje na sabirnici, odnosno konektor (utor) na matičnoj ploči.
Tako se primjerice grafička ili mrežna kartica obično spajaju na *PCI Express* ili *PCI* sabirnicu odnosno u njen utor.
S druge strane klasični tvrdi disk ili SSD disk te CD/DVD-ROM uređaji spajaju se na *ATA* ili *SATA* kontroler na matičnoj ploči, preko posebnog *ATA* ili *SATA* kabela te dodatnog kabela za napajanje, sve ovisno je li njihovo sučelje *ATA* ili *SATA* vrste.

**Izvori informacija: (168),(169),(170), (K-9)**

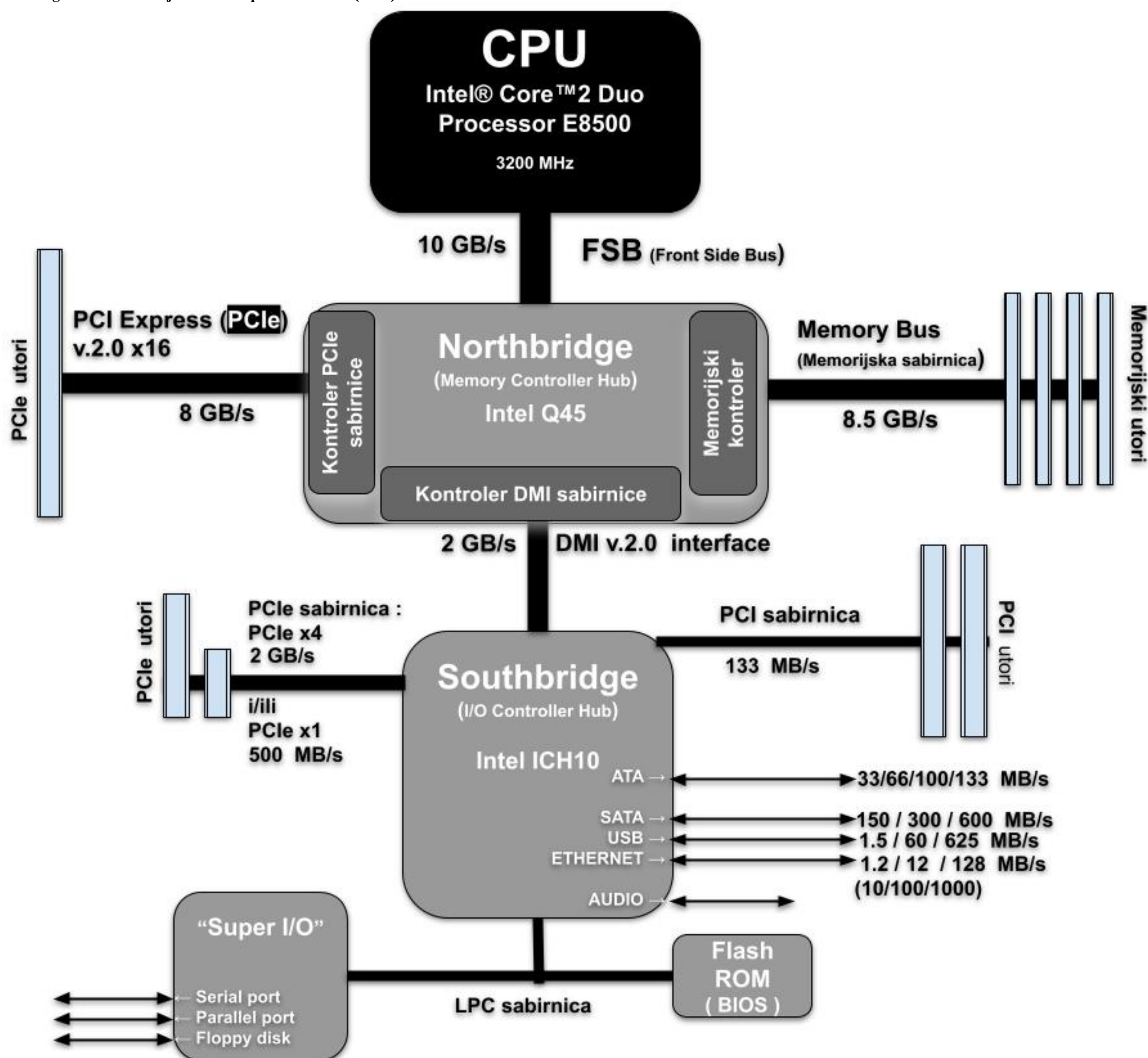
### 10.1.1. Matična ploča, CPU, *chipset* i sabirnica(e)

Matična ploča (engl. *Motherboard*[(171)]) je centralni dio svakog računala. Na njoj se nalaze sve komponente potrebne za funkcioniranje računala; od napajanja za sve komponente na samoj ploči, preko procesora do sabirnica preko kojih sve komponente međusobno komuniciraju. Sâma matična ploča za osobna računala, izrađuje se u nekoliko dimenzija odnosno formata (od većeg prema manjem): standardni *ATX*, mikro *ATX*, mini *ATX* i nekoliko *ITX* formata (*mini*, *nano* i *piko*). Matična ploča je zapravo velika štampana ploča s već ugrađenim (zalemljenim) integriranim krugovima (*čipovima*), ali i posebnim utorima u koje možemo ugraditi komponente poput: centralnog procesora (*CPU*), *RAM* memorije i raznih kartica; mrežna, grafička i slično. Pogledajmo koje su to sve komponente koje se već nalaze ugrađene ili ih možemo ugraditi na matičnu ploču:

- CPU (centralni procesor) i RAM memorija.
- Disk kontroler: obično ATA ili SATA(već su ugrađeni/zalemljeni).
- Serijsko i paralelno sučelje (obično su već ugrađeni/zalemljeni).
- USB kontroler i USB sučelja(već su ugrađeni/zalemljeni).
- Zvučna kartica(obično je već ugrađena/zalemljena).
- A ponekad su integrirani ili dolaze odvojeno, kao zasebne kartice, a to su: grafička ili mrežna kartica.

Kako bismo razumjeli rad računala, pogledajmo logičku shemu malo starijeg dizajna matične ploče s pripadajućim *CPU-om* (*Intel Core 2 Duo E8500*) i njegovim *čipsetom*: *Intel* Q45 i ICH10. Na ovoj logičkoj shemi se vidi pozicija svih važnih dijelova matične ploče. *CPU* (engl. *Central Processing Unit*) tj. centralni procesor na vrhu slike, komunicira s ostatkom računala preko posebne vrste sabirnice. Ova sabirnica povezuje *CPU* sa tzv. *čipsetom* (*chipset)*, koji je na ovoj generaciji procesora, odnosno u to vrijeme, bio podijeljen u dva dijela, kako je vidljivo na slici **42**. Prvi dio je bio tzv. *sjeverni most* odnosno ***Northbridge*** dio.

**Slika 42. Logička shema dizajna matične ploče računala (Intel).**




***Northbridge*** **dio čipseta**[(173)]

Ovaj dio *čipseta*, koji je izveden na zasebnom *sklopu* i nalazi se najbliže procesoru, se naziva i *Memory Controller Hub (MCH)*. On je primarno zadužen za komunikaciju između procesora i RAM memorije. Stoga se u njemu nalazio i memorijski kontroler za RAM memoriju. Osim toga on je zadužen za rad primarne sabirnice visoke propusnosti, koja se uglavnom koristi za potrebe spajanja grafičke kartice (u ovom slučaju je to *PCI Express* v.2.0, x16) ili drugih kartica koje zahtijevaju vrlo veliku propusnost. Dodatna uloga ovog dijela *čipseta* je komunikacija s dodatnim *čipsetom (tzv.* ***Southbridge****)*, putem **DMI (***Direct Media Interface***)** sabirnice, kako je vidljivo na slici.

***Southbridge*** dio[174]

*Southbridge* je također izveden kao zaseban *sklop* ili komponenta, a komunicira s gore navedenim *čipsetom*, također preko posebne **DMI** sabirnice. Ovaj dio *čipseta* se naziva i *I/O controller Hub* odnosno središte za ulazno/izlaznu komunikaciju; poput diskova (*ATA* ili *SATA*), mreže, USB, zvučne kartice i slično.
Namjena *Southbridge* odnosno takozvanog *južnog mosta* je višestruka:

- Upravljanje i komunikacija s drugim sabirnicama, poput *PCI* ili *PCI Express* (ali x4 ili x1) koje su znatno sporije od sabirnice kojom upravlja ***Northbridge*** sklop, stoga što je propusnost prema *Southbridge* čipu ograničena propusnosti **DMI** (*Direct Media Interface*) sabirnice.
- Unutar njega se obično nalaze i disk kontroleri za *ATA* (poznat i kao *PATA*) ili *SATA* diskove te mrežna kartica, USB kontroler, a često i zvučna kartica.
- On je veza s *LPC* (*Low Pin Count*) sabirnicom na koju su preko tzv. *Super I/O* kontrolera spojeni uređaji koji za svoj rad trebaju vrlo male brzine komunikacije, poput serijskih ili paralelnih portova, *PS/2* portova za spajanje miševa i tipkovnica, kao i *BIOS* (*ROM*) računala, koji također radi na vrlo maloj brzini pa mu nije potrebna brža sabirnica za rad i komunikaciju.

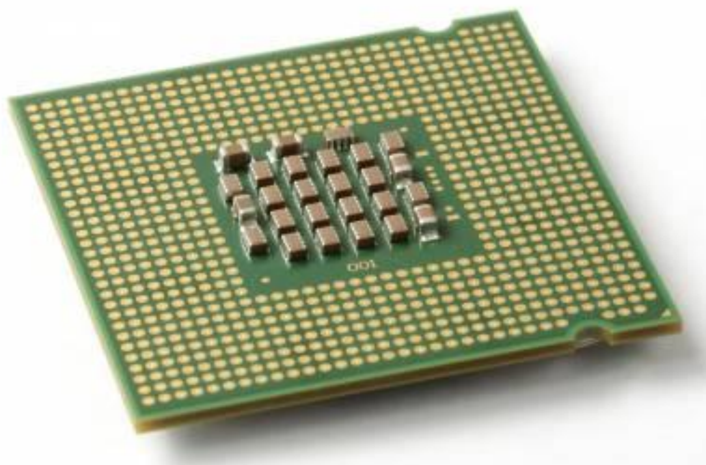
Cjelokupni *čipset* je zadužen za komunikaciju između procesora i memorije te svih ostalih komponenti računala. Od disk kontrolera i diskova, preko sabirnice do grafičke, mrežne, zvučne i drugih kartica. Kao i USB i serijskih, paralelnih i drugih sučelja (*portova*). Dakle možemo reći kako sprega procesora i *čipseta (sklopovlja)* čine matičnu ploču.
Mana ovakvog dizajna (*CPU + Northbridge + Southbridge*) pokazala se zbog potrebe za sve većim brzinama komunikacije između procesora i sjevernog mosta (*Northbridge)* jer je povećanjem brzine obrade podataka novijih procesora bilo potrebno ubrzati i komunikaciju s RAM memorijom, ali i s vrlo brzom *PCI Express* sabirnicom; za spajanje grafičkih i drugih ekstremno brzih kartica poput primjerice 10Gbps (ili bržih) mrežnih kartica.

Pogledajte i stvarnu sliku procesora koji pripada generaciji s ovakvim dizajnom (CPU + *Northbridge* + *Southbridge*).
Na slici **43** je ***Intel Pentium 4*** (*Prescott 2M*) radnog takta 3.2GHz, FSB 800MHz, sa 2MB priručne (engl. *Cache*) memorije.

**Slika 43. Prednja i stražnja strana mikroprocesora (CPU).**




Prednja strana *CPU-a Intel Pentium 4 (640)*:

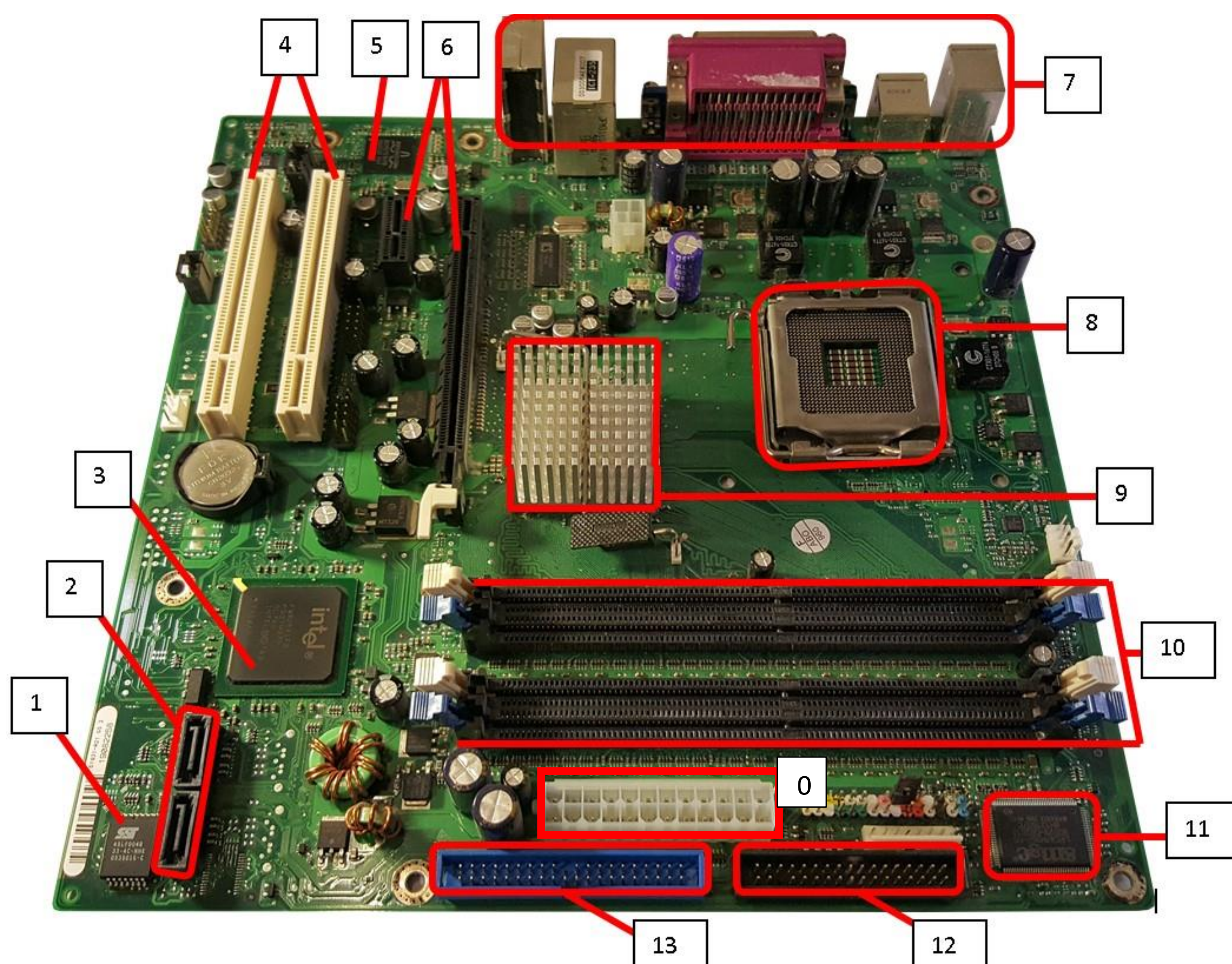


Stražnja strana navedenog *CPU-a*:

Autor slika CPUa: ***Eric Gaba (Wikipedia)***

Pogledajte i sliku pripadajuće matične ploče za ***Intel Pentium 4: 640 CPU** (kodnog imena Prescott)*.

**Slika 44. Pogled na matičnu ploču računala.**




Slijedi opis komponenti matične ploče označenih na slici:

***0.** Konektor za napajanje (ATX) cijele matične ploče i svih komponenti na ploči.*
***1.** BIOS i LPC sučelje.*
***2.** SATA sučelja za spajanje diskova: mehaničkih, SSD ili optičkih uređaja(CD/DVD/BlueRay).*
***3.** Southbridge sklop (chip).*
***4.** PCI utori (PCI sabirnice).*
***5.** Kontroler (chip) mrežne kartice.*
***6.** PCI-Express (x4 i x16) utori (PCI Express sabirnice).*
***7.** Vanjski konektori: VGA, LAN, audio, serijski i paralelni portovi, USB, PS/2.*
***8.** CPU utor: Socket 775 utor za Pentium 4 Prescott 2M procesor.*
***9.** Northbridge sklop (chip) s aluminijskim hladnjakom.*
***10.** Utori za RAM memoriju (DDR-400).*
***11.** Super I/O kontroler (chip).*
***12.** Floppy disk konektor (34.pin).*
***13.** ATA (PATA) disk konektor (40. pin.).*

Probajte usporediti komponente koje su označene u opisu, s onima koje smo opisali u teoriji, na sâmom početku ovog poglavlja.

**A sada pogledajmo nešto noviju generaciju procesora (Intel i5 6500) i njegov pripadajući *chipset* (Z170)**

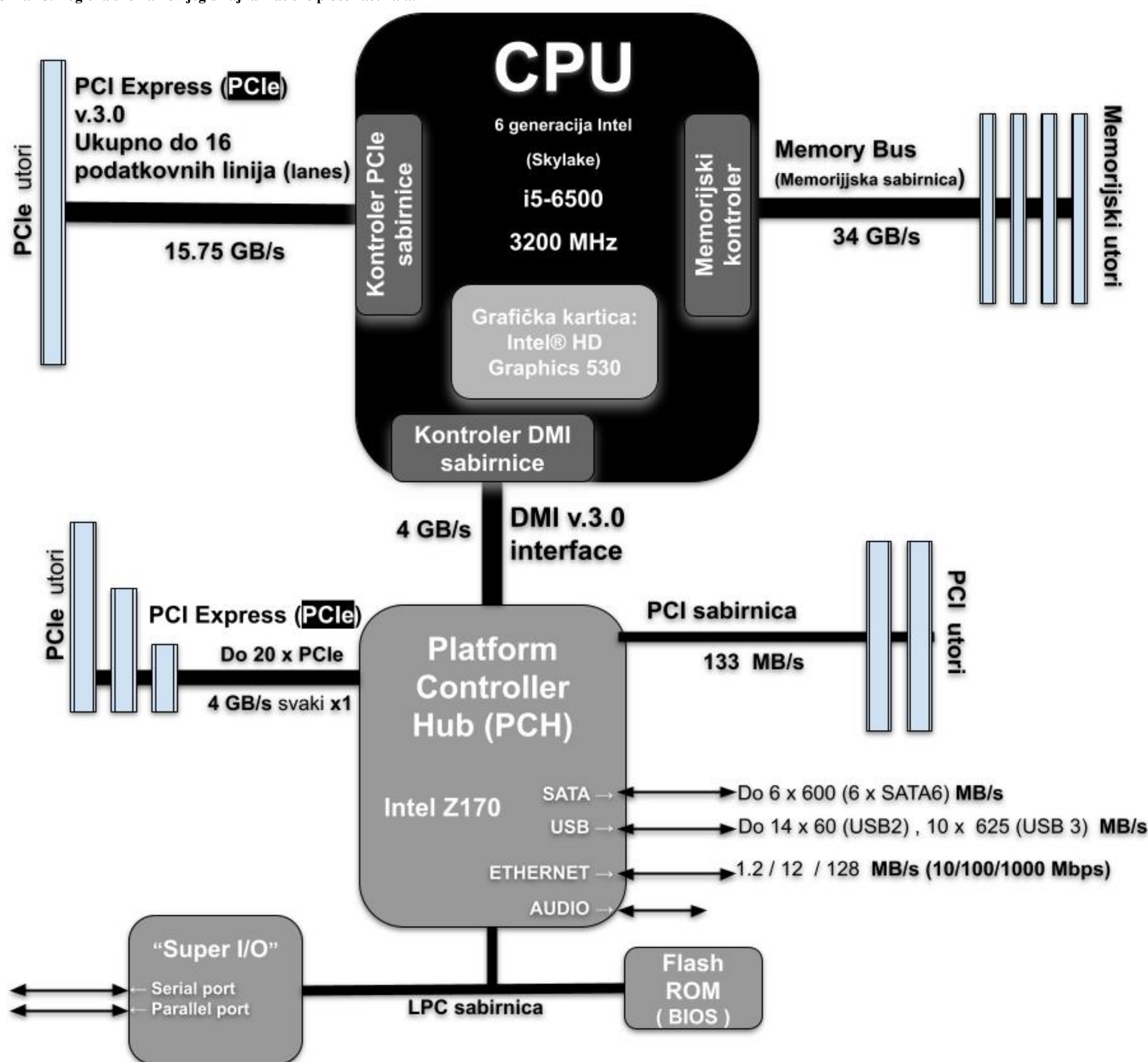
U najnovijim generacijama procesora i *čipseta*, navedeni problem starih *čipseta* je riješen tako što je memorijski kontroler prebačen unutar sâmog procesora. Osim toga i kontroler zadužen za ekstremno brzu *PCI Express* sabirnicu se sada također nalazi integriran unutar procesora. Dodatno, i kontroler ***DMI*** sabirnice je sada integriran unutar procesora, a vrlo često se integrira i sama grafička kartica.

Dakle potpuna funkcionalnost sjevernog mosta (*Northbridge)* je sada integrirana unutar sâmog procesora.

Ovakav dizajn drastično je ubrzao pristup RAM memoriji, *PCI Express* sabirnici te komunikaciju preko *DMI* sabirnice, kako ćete vidjeti na slici.

Sljedeći logičan korak je bio ubrzati i vezu sa novim dijelom *čipseta*, koji se sada uobičajeno naziva *Platform Controller Hub* (**PCH**) što je i učinjeno s novijom **DMI** sabirnicom veće propusnosti (na slici 45. dolje).

Slika 45. Logička shema novijeg dizajna matične ploče računala.




Pošto je i *Platform Controller Hub* (**PCH**)*)* dobio veću propusnost prema procesoru preko brže **DMI** sabirnice, on je sada u mogućnosti povećati propusnost prema *PCI Express (PCIe)* sabirnici za koju je zadužen (za sporija: *PCIe* x4 i x1 sučelja). Nadalje, povećane su i brzine rada *SATA* kontrolera, prema novim standardima (*SATA* 3, koji sada omogućavaju brzinu od 600MB/s po disku). USB sučelja (koja sada podržavaju USB 3.0 standard koji omogućava brzinu od 625MB/s) i slično.

U novom dizajnu je zadržana ***LPC*** sabirnica za komunikaciju sa *Super I/O*: odnosno da serijskim i paralelnim portovima, *BIOS-om* i drugim sporijim komponentama.

I na kraju priče o sabirnicama, pogledajmo brzine *PCI*, *PCI-X* i *PCI Express* sabirnica, na koje spajamo različite kartice poput: brzih mrežnih, *SAS*/*SATA* ili *RAID* kontrolera, grafičkih, zvučnih, ali i drugih kartica.

PCI i PCI-X sabirnice:

| Naziv | Brzina rada (MHz) | Širina sabirnice | Propusnost Mb/s |
|---|---|---|---|
| PCI | 33 MHz | 32 bitna | 1.064 Mb/sec |
| PCI | 33 MHz | 64 bitna | 2.128 Mb/sec |
| PCI | 66 MHz | 32 bitna | 2.128 Mb/sec |
| PCI | 66 MHz | 64 bitna | 4.256 Mb/sec |
| PCI-X | 100 MHz | 64 bitna | 6.400 Mb/sec |
| PCI-X | 133 MHz | 64 bitna | 8.192 Mb/sec |

S lijeve strane u tablici, pogledajmo brzine sabirnica, na koje spajamo mrežne (i druge) kartice odnosno uređaje. Ovdje vidimo starije vrste sabirnica: **PCI**[(175)] i **PCI-X**[(176)] koje su se koristili prije, ali se ponekad koriste i danas.

U tablici dolje pogledajmo i noviju ***PCI Express***[177] sabirnicu, koja se označava kao **PCIe**, kod koje se komunikacija odvija kroz komunikacijske kanale odnosno takozvane trake (engl. *Lane*). Pri tome je svaki komunikacijski kanal odnosno traka dvosmjerna, a zapravo se sastoji od dva kanala odnosno para: jednog za primanje i jedan za slanje podatka.

*PCI Express* (*PCIe*) sabirnica:

| **Jedna traka** *(Per lane – **1x**)* | **Propusnost GT/s (zaokruženo)** u svakom pojedinom smjeru | **Način kodiranja** | | **Propusnost MB/s** jednosmjerno |
|---|---|---|---|---|
| PCIe v.1. — **1x** | 2,5 GT/s | **NRZ** *(Non-return-to-zero)* | ***8b/10b*** | 250 MB/s |
| PCIe v.2. — **1x** | 5 GT/s | | | 500 MB/s |
| PCIe v.3. — **1x** | 8 GT/s | | ***128b/130b*** | 985 MB/s |
| PCIe v.4. — **1x** | 16 GT/s | | | 1.969 MB/s (1,97 GB/s) |
| PCIe v.5. — **1x** | 32 GT/s | | | 3.938 MB/s (3,94 GB/s) |
| PCIe v.6. — **1x** | 64 GT/s | **PAM-4** | ***1b/1b*** | 7.536 MB/s (7,54 GB/s) |

Oznaka **GT/s** oznaka je za milijardu (*giga*) transfera u sekundi, pa je tako za ***PCIe v.1.*** i 2,5 GT/s propusnost od 2,5 Gbps tzv. *bit-rate* propusnost. Ako se maknu zaglavlja koja nosi **PCIe** protokol, to konkretno iznosi 2 Gbps odnosno 250MB/s, za *8b/10b* način kodiranja. To također znači da kod *8b/10b* kodiranja 20% ukupno prenesenih podataka čine kontrolni podaci.
U tablici vidimo i da **PCIe v.1**. i **v.2.** koriste **NRZ**: *8b/10b* način kodiranja, dok **PCIe v.3.** do **v.5.** koriste **NRZ:** *128b/130b* način kodiranja. Dok **PCIe v.6.** koristi takozvano **PAM-4** (engl. *Pulse-amplitude_modulation*): *1b/1b* kodiranje.

Tablica dolje prikazuje maksimalne brzine za *PCI Express (PCIe)*, koje su prema tome, sa 16 traka (***16x***) u uporabi, sljedeće:

| **Prema trakama** (***Per lane – 16x***) | **Propusnost GB/s (zaokruženo)** — u svakom pojedinom smjeru |
|---|---|
| PCIe v.1. — 16x | **4 GB/s** |
| PCIe v.2. — 16x | **8 GB/s** |
| PCIe v.3. — 16x | **15,75 GB/s** |
| PCIe v.4. — 16x | **31,5 GB/s** |
| PCIe v.5. — 16x | **63 GB/s** |
| PCIe v.6. — 16x | **121 GB/s** |

Za ostale brzine (x2, x4 i x8) propusnost si izračunajte sami. U svakom slučaju, važno je znati da sabirnica mora biti dovoljno brza kako bi podnijela zbroj svih potrebnih propusnosti svih kartica koje ugrađujete u računalo odnosno poslužitelj.

***Slijedi napredni dio!***

## PCI, PCI-X i PCI Express (PCIe) sabirnice detaljnije

## PCI sabirnica

Moderni centralni procesori imaju ugrađen memorijski kontroler koji preko memorijske sabirnice komunicira s RAM memorijom te takozvani PCI **Host Bridge** sklop preko kojeg se odvija komunikacija s PCI sabirnicom. Često se koriste i dodatni sklopovi koji se zovu **PCI bridge** sklopovi, a koji produžuje sabirnicu zbog električnih gubitaka (pogledajte shemu 45.1. na drugoj stranici). Pri tome je ***PCI Host Bridge*** nužan za rad PCI sabirnice, a neki od njegovih osnovnih zadataka su sljedeći:

- Omogućuje komunikaciju između procesora (CPU) i RAM memorije te PCI sabirnice i uređaja na njoj.
- Preusmjerava zahtjeve između RAM memorije i PCI memorije ili I/O adresnog prostora PCI kartica.
- Odgovoran je za pristup PCI konfiguracijskom prostoru svakog uređaja na PCI sabirnici.
- Omogućuje sustavu da detektira, inicijalizira i konfigurira PCI uređaje tijekom procesa pokretanja računala (*boot*).
- Preusmjerava signale prekida (**IRQ**) generirane od PCI uređaja prema procesoru te podržava različite signale prekida:
  - ***INTx*** – stariji hardverski signali prekida koji koriste posebne PCI vodove [*linije*].
  - ***MSI/MSI***-X – moderni brži i učinkovitiji signali prekida koji se šalju kao poruke (paketi) na sabirnici.
- Arbitraža i upravljanje prometom između procesora, RAM memorije i PCI uređaja te **DMA** prijenos.

Naime, već na PCI sabirnici bila su poznata fizička ograničenja kao što je maksimalna duljina sabirnice (zbog slabljenja električnih signala), ali i standardna logička granica mogućnosti učitavanja samo osam uređaja na sabirnicu od 33 MHz, a još manje na sabirnicu od 66 MHz. Zbog tih ograničenja u dizajn je i dodan premosni sklop (***PCI Bridge***) koji dopušta produljenje i proširenje PCI sabirnice. Tako se došlo do strukture stabla međusobno povezanih ulazno/izlaznih (I/O) sabirnica podržanih nizom navedenih premosnih sklopova. U takvoj strukturi podređeni premosni sklopovi PCI sabirnice mogu se hijerarhijski proširiti ispod glavnog premosnog sklopa kako bi se omogućilo proširenje jednog sustava sabirnice u složeni sustav s više sekundarnih sabirnica.
Pri tome se PCI uređaji (to jest *kartice*) mogu spojiti na jednu ili više navedenih sekundarnih (ili čak tercijarnih) sabirnica.

## PCI-X sabirnica

Nešto novija **PCI-X** sabirnica uvela je rad ne većim frekvencijama, a samim time i veću propusnost. Međutim i **PCI** i **PCI-X** implementirali su paralelni način komunikacije kod koje ista sabirnica povezuje nekoliko uređaja s procesorom.
Glavni nedostatak ove arhitekture jest taj da samo jedan PCI/PCI-X uređaj (*kartica*) može komunicirati s računalom (u konačnici s procesorom) u isto vrijeme. Nadalje, osim spomenutog premosnog sklopa koji se obično naziva **PCI Bridge** možemo naići i na sklopove naziva: ***PCIto-PCI bridge*** ili dodatno ***PCI-to-ISA bridge***, jer je i ovdje moguća veza s PCI na ISA sabirnicu koju ovakvi sklopovi odrađuju.

## PCI Express (PCIe) sabirnica

Umjesto paralelne veze, najnovija ***PCI-Express*** (**PCIe)** sabirnica koristi serijsku vezu koja radi prema principu od točke do točke (tzv. *point-to-point*). U ovakvom načinu rada svaki uređaj na sabirnici komunicira s procesorom putem vlastitog skupa komunikacijskih kanala (engl. *lane*) tako da može komunicirati izravno s takozvanim ***Root complex*** sklopom koji potom komunicira s procesorom bez potrebe da čeka druge uređaje na sabirnici. Naime, uređaji na *PCI Express* sabirnici komuniciraju slanjem posebne vrste paketa. To znači da je osim klasične podatkovne komunikacije (slanja uobičajenih podataka kroz sabirnicu) moguće slati i druge poruke, poput ***DMA*** zahtjeva ili ***IRQ*** zahtjeva (korištenjem ***MSI*** ili ***MSI-X*** poruka).

***PCIe Root Complex***

Na vrhu logičke hijerarhije ***PCI Express*** (***PCIe***) sabirnice nalazi se takozvani ***Root Complex*** čiji zadaci su sljedeći**:**

- Povezivanje CPU-a i memorije s ***PCIe*** sabirnicom, slično kao što ***PCI Host Bridge*** radi sa PCI sabirnicom, te:
  - o Generiranje i obrada *PCIe* transakcija između računala (CPU i RAM) i ***PCIe*** uređaja.
  - o *Koristi Memory-Mapped I/O* (**MMIO**) za upravljanje ***PCIe*** uređajima.
- Rad s tzv. ***Root portovima***:
  - o *Root Complex* inicijalno sadrži jedan ili više *Root Portova* koji predstavljaju izlazne točke prema ***PCIe*** sabirnici.
  - o Svaki ***Root Port*** može voditi do: ***PCIe Switcha*** ili ***PCIe Bridge-a*** ili direktno do ***PCIe*** uređaja (*kartice*).
- Konfiguracija ***PCIe*** uređaja:
  - o Upravljanje *PCIe* konfiguracijskim prostorom i alokacija resursa tijekom inicijalizacije sustava.
  - o Podrška za hardverske signale prekida putem **MSI** i **MSI-X** poruka te **DMA** prijenos.

**PCIe switch**

Za proširenje **PCIe** sabirnice koristi se **PCIe Switch.** On je s jedne strane spojen na ***Root Complex***, a s druge strane se na njega spajanju ***PCIe*** uređaji. Njegova zadaća je sljedeća:

- Proširenje *PCIe* sabirnice spajanjem više *PCIe* uređaja na jedan ***Root port*** (na ***Root complexu***).
- Usmjerava *PCIe* transakcije između uređaja na *downstream* portovima i *upstream* portu prema ***Root Complexu***.
  - o Preklapanje *PCIe* prometa u oba smjera (*downstream* i *upstream).*
- Zadužen je i za međusobnu komunikaciju između više ***PCIe*** uređaja: omogućuje *PCIe* uređajima međusobnu izravnu komunikaciju (*peer-to-peer*) bez prolaska kroz ***Root Complex*** (npr. između dvije grafičke kartice).

**PCIe Bridge**

***PCIe Bridge*** je komponenta koja služi za povezivanje ***PCIe*** sabirnice s drugim vrstama sabirnica (ili proširenim *PCIe* domenama). Postoje različite vrste *PCIe Bridge-ova*, kao što su: ***PCIe-to-PCI Bridge*** i ***PCIe-to-PCIe Bridge***.
U slučaju ***PCIe-to-PCI Bridge-a***, on prevodi protokole između ***PCIe*** i starijih ***PCI*** uređaja.

Pogledajmo sažetak razlika komponenti **PCIe** sabirnice:

| Komponenta | Zadaća |
|---|---|
| **PCIe Root Complex** | Povezuje CPU i RAM memoriju s ***PCIe*** sabirnicom, konfigurira ***PCIe*** uređaje, upravlja signalima prekida (IRQ) i DMA. |
| **PCIe Switch** | Proširuje ***PCIe*** sabirnicu, usmjerava promet, omogućuje *peer-to-peer* komunikaciju i *Hot-Plug*. |
| **PCIe Bridge** | Povezuje različite vrste sabirnica (***PCIe-PCI***, ***PCIe-PCIe***) i upravlja adresnim mapiranjem |

**Pogledajmo i ključne razlike između PCI i PCIe sabirnica po pitanju hardverskih signala prekida te DMA prijenosa.**
Za usporedbu upotrebe hardverskih signala prekida (IRQ) pogledajte tablicu:

| Karakteristika | PCI sabirnica | PCI Express (PCIe) sabirnica |
|---|---|---|
| **Korištenje PIC-a** | Potreban za fizičke signale prekida preko zasebnih vodova (**INT#**). | Nije potreban zbog ***MSI/MSI-X***. |
| **Korištenje APIC-a/LAPIC-a** | Koristi se za obradu hardverskih signala prekida (IRQ). | Koristi se za obradu ***MSI/MSI-X PCIe*** paketa. |
| **Metoda prijenosa hardverskih signala prekida (IRQ)** | Fizički vodovi (linije). Novije inačice PCI standarda - ***MSI***. | Paketi preko ***PCIe*** sabirnice (***MSI/MSI-X***). |
| **Ograničenja** | Dijeljenje IRQ linija i ograničenje broja hardverskih signala prekida. | Skalabilnost i paralelizam zahvaljujući ***MSI*** i ***MSI-X***. |

Za *DMA* prijenos podataka pogledajte tablicu:

| Karakteristika | PCI sabirnica | PCI Express (PCIe) sabirnica |
|---|---|---|
| **Bus mastering** | Centralizirani *arbitar* upravlja zahtjevima za kontrolu sabirnice i prijenos podataka. | Nema potrebe za *arbitrom* ili kontrolom sabirnice zbog *point-to point* načina rada. |
| **Vrsta sabirnice** | Dijeljena; svi uređaji koriste istu sabirnicu. | Svaki uređaj ima zasebnu vezu prema ***Root Complexu*** i/ili ***PCIe Switchu***. |
| **Arbitar sabirnice** | ***PCI Host Bridge.*** | Nije potreban. |
| **DMA kontroler** | Ugrađen u svaki uređaj, ali mora koristiti dijeljenu sabirnicu. | Ugrađen u svaki uređaj s direktnim pristupom PCIe komunikacijskoj traci (engl. *lane*). |



Pogledajte i poglavlja:
**10.5.1. IRQ (*Interrupt request*).**
**10.6. Direktan pristup memoriji – DMA.**
**10.7.1.4. Ulazno-izlazni memorijski kontroler (IOMMU).**
**10.7.2.2.1. NUMA I/O topologija.**

Pogledajmo sve navedene sklopove i s naredbom `lspci` (ispis smo drastično skratili):

```
lspci  |grep -i bridge

00:00.0 Host bridge: Intel Corporation Xeon E5/Core i7 DMI2 (rev 07)
00:01.0 PCI bridge: Intel Corporation Xeon E5/Core i7 IIO PCI Express Root Port 1a (rev 07)
00:02.0 PCI bridge: Intel Corporation Xeon E5/Core i7 IIO PCI Express Root Port 2a (rev 07)
00:02.2 PCI bridge: Intel Corporation Xeon E5/Core i7 IIO PCI Express Root Port 2c (rev 07)
```

Uočite da se prvi redak ispisa odnosi na `Host bridge`, koji povezuje centralni procesor sa sabirnicom, te da su ispod njega `PCI bridge` sklopovi. Označili (podcrtali smo) dva ***PCI Bridge*** sklopa s oznakama (`00:02.0` i `00:02.2`) jer su na njih spojene mrežne kartice te RAID disk kontroler koje ćemo dalje promatrati. Ovdje vidljive oznake, pr. `00:02.0` znače redom: ***sabirnica:uređaj.funkcija***. Ovakva adresna shema se naziva *Bus/Device/Function* (**BDF**).

Nadalje, pošto na sabirnici uvijek imamo i ***PCI bridge*** ili čak i više njih, događa se da osim ovakve gornje sheme označavanja u kojoj imamo vidljivu primarnu sabirnicu, možemo imati i sekundarne sabirnice, pa logički imamo sljedeću vezu:

*Primarna-sabirnica:uređaj.funkcija* → *sekundarna-sabirnica:uređaj.funkcija* → ***PCI uređaj*** ***(****sabirnica:uređaj.funkcija****)***

← Moguće je imati i tercijarne ili čak i još dublje razine pod sabirnica.
← Kod **PCIe** sabirnice je moguće imati i portove na ***PCIe switchu***.

Sada pogledamo stablo ovih veza s istom naredbom (ispis smo ponovno drastično skratili):

```
lspci -tv

-+-[0000:3f]-+-08.0  Intel Corporation Xeon E5/Core i7 QPI Link 0
 |           +-09.0  Intel Corporation Xeon E5/Core i7 QPI Link 1
 \-[0000:00]-+-00.0  Intel Corporation Xeon E5/Core i7 DMI2
             +-01.0-[11]--
             +-02.0-[04]--+-00.0  Emulex Corporation OneConnect 10Gb NIC (be3)
             |            +-00.1  Emulex Corporation OneConnect 10Gb NIC (be3)
             |            +-00.2  Emulex Corporation OneConnect 10Gb NIC (be3)
             |            +-00.3  Emulex Corporation OneConnect 10Gb NIC (be3)
             +-02.2-[03]----00.0  Hewlett-Packard Company Smart Array Gen8 Controllers
```

← Ovdje vidimo sljedeće:

- `02.0` - je PCI oznaka **PCI bridge** sklopa (`00:02.0`) vidljivog iz prijašnjeg ispisa. Na tom **PCI bridge** sklopu je sabirnica na koju su spojena četiri uređaja sa svojim oznakama:
  - `[04]--+-00.0` – to jest `04:00.0` - na toj PCI poziciji (utoru) je prva *Ethernet* mrežna kartica.
  - `[04]--+-00.1` – to jest `04:00.1` - na toj PCI poziciji (utoru) je druga *Ethernet* mrežna kartica.
  - `[04]--+-00.2` – to jest `04:00.2` - na toj PCI poziciji (utoru) je treća *Ethernet* mrežna kartica.
  - `[04]--+-00.3` – to jest `04:00.3` - na toj PCI poziciji (utoru) je četvrta *Ethernet* mrežna kartica.
- `02.2` - je PCI oznaka **PCI bridge** sklopa (`00:02.2`) od prijašnjeg ispisa. Na tom **PCI bridge** sklopu je sabirnica na koju je spojen jedan uređaj sa svojim oznakama:
  - `[03]--+-00.0` – to jest `03:00.0` - na toj PCI poziciji (utoru) je RAID kontroler.

Na prethodnom ispisu smo vidjeli hijerarhijsku strukturu i pozicije na kojima se nalaze PCI/PCIe kartice (uređaji). Pri tome je važno razumjeti da oznake koje smo vidjeli identificiraju poziciju uređaja na sabirnici i u konačnici utoru (*PCI/PCIe slot*) u koji je ugrađena kartica odnosno na koji je spojena. Primjerice prva mrežna kartica ovdje ima PCI oznaku `04.00.0`.
To je konkretno oznaka **PCIe** porta na ***PCI switchu*** broj `4`, uređaja broj `00` i funkcije broj `0`, a ne samog uređaja koji je spojen na sabirnicu.

Pogledajmo sada ispis naredbe `lspci` koja će nam otkriti još jednu oznaku (unutar uglatih zagrada **[]**):

```
lspci -nn | grep -i Ethernet

04:00.0 Ethernet controller [0200]: Emulex Corporation OneConnect 10Gb NIC (be3) [19a2:0710]
04:00.1 Ethernet controller [0200]: Emulex Corporation OneConnect 10Gb NIC (be3) [19a2:0710]
04:00.2 Ethernet controller [0200]: Emulex Corporation OneConnect 10Gb NIC (be3) [19a2:0710]
04:00.3 Ethernet controller [0200]: Emulex Corporation OneConnect 10Gb NIC (be3) [19a2:0710]
```

Ovdje smo za svaki uređaj spojen na sabirnicu, osim oznaka sabirnice, to jest hijerarhijske veze kako je spojen na sabirnicu, primjerice `04:00.0`, dodali i takozvani ***PCI identifikator*** uređaja odnosno kartice, koji je upisan unutar uglatih zagrada, primjerice `[19a2:0710]`. Navedeni PCI identifikator se naziva **PCI ID**. Naime, svaki uređaj (pr. mrežna kartica) spojen na sabirnicu računala ima zapisan svoj **PCI ID** identifikacijski broj koji se sastoji od dva heksadecimalna broja odvojena dvotočkom:

- Prvi dio broja (prije dvotočke) identificira proizvođača; primjerice: *IBM, Intel, Broadcom,…*
- Dok drugi dio broja identificira točnu vrstu (tip) i model uređaja.

Nemojte pobrkati navedenu **PCI ID** i oznaku sabirnice i utora u kojem se nalazi uređaj (kartica), mada izgledaju slično. Ovaj **PCI ID** identifikator je samog PCI uređaja (kartice), koji sustav koristi da bi za njega uspio pronaći prikladan upravljački program.



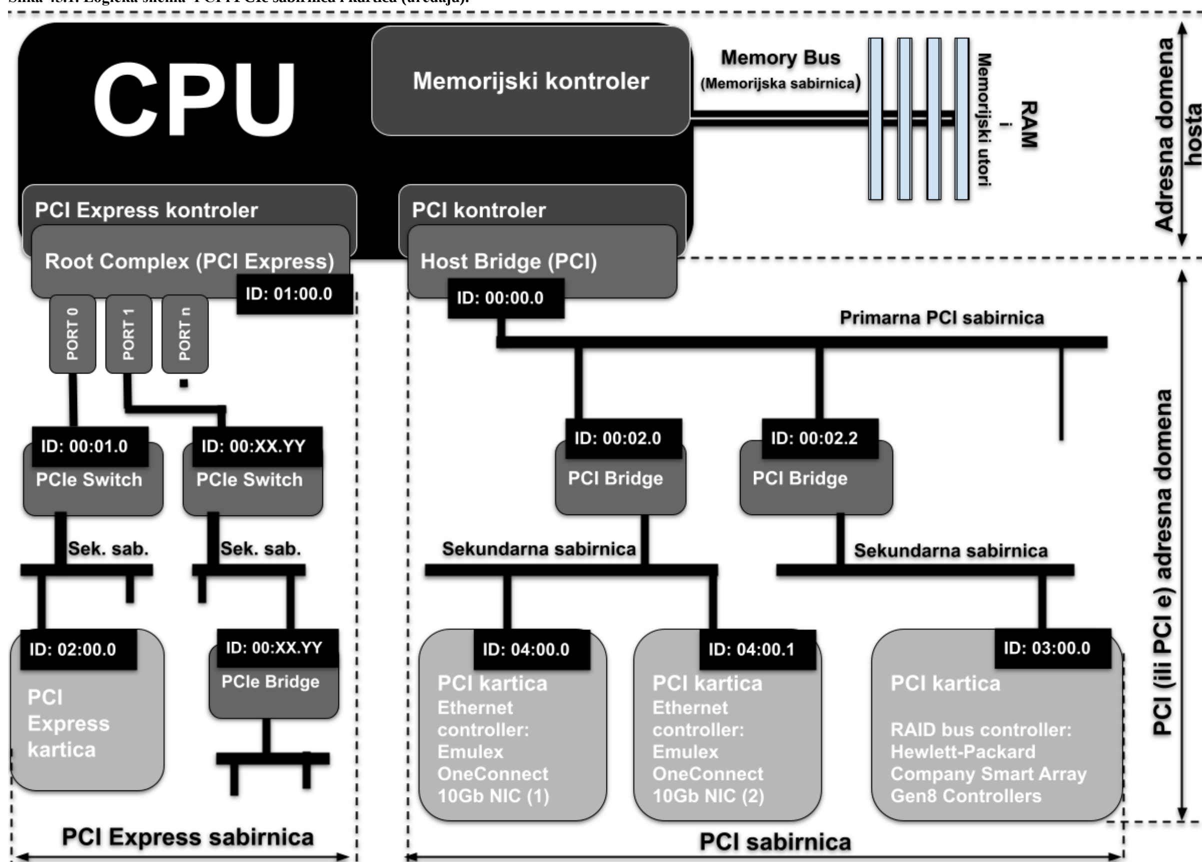
Za detalje o **PCI ID** identifikatorima, pogledajte poglavlja:
**10.2.3. Nadogradnja baze podataka PCI identifikatora (PCI baze).**
**11.1.2.1. Uređaji (*devices*) detaljnije**.

Grafički lijep prikaz sabirnica, ***PCI bridge*** sklopova i uređaja možemo dobiti s naredbom (dolazi u programskom paketu `hwloc`):

```
lstopo --whole-io
```

Sada konačno pogledajmo i logičku shemu hijerarhijskih veza ***PCI/PCIe*** sabirnica, kako je vidljivo na slici 45.1.:

**Slika 45.1. Logička shema PCI i PCIe sabirnica i kartica (uređaja).**



## PCI Express (PCIe) sabirnica detaljnije

Na najnižoj razini, **PCIe** sučelje koristi diferencijalne parove komunikacijskih vodova (*traka*) za prijenos podataka, što znači da se podaci prenose kao razlika napona između dva voda. Svaki par vodova ima svoju funkciju, npr. jedan par vodova se koristi za slanje podataka (**TX** linije), dok se drugi par koristi za primanje podataka (**RX** linije).

**PCIe** sučelje koristi više parova vodova kako bi se postigla veća propusnost, a ovisno o inačici i izvedbi *PCIe* sučelja, može se koristiti od 1 do 32 parova vodova. *PCIe* sučelje koristi protokol za serijski prijenos podataka (engl. *serial transfer protocol*) koji podatke dijeli u male pakete, zvane poruke. Ove poruke odnosno paketi se zovu *Transaction Layer Packet* (**TLP**).
Svaka ova poruka sadrži informacije o vrsti operacije koja se izvršava (npr. čitanje ili pisanje), adresu kojoj se pristupa, podatke koji se prenose te kontrolne podatke za provjeru ispravnosti podataka.
*PCIe* kontroler u računalu (uobičajeno u procesoru) i kontroler na *PCIe* kartici zajednički upravljaju prijenosom podataka, koristeći protokol koji omogućuje siguran i pouzdan prijenos podataka. Uz to, ***PCIe*** sučelje ima i posebne vodove (linije) za upravljanje i signalizaciju, koje se koriste za komunikaciju između računala i *PCIe* kartice.
Na primjer, linija ***Reset*** se koristi za resetiranje kartice, a linija ***Clock*** se koristi za sinkronizaciju prijenosa podataka. Sve ove komponente zajedno omogućuju brzi i učinkovit prijenos podataka između računala i *PCIe* kartica, što je važno za performanse računala u mnogim aplikacijama.

## PCI i PCIe konfiguracijski adresni prostor

Konfiguracijski prostor **PCI** i **PCIe** uređaja (kartice) definiran je hijerarhijski; drugim riječima, lokacija perifernog uređaja određena je njegovom fizičkom lokacijom unutar međusobno povezanog stabla mostova (***PCI bridge***) na *PCI/PCIe* sabirnici. Uređaj se locira prema oznaci sabirnice i oznaci uređaja (utora). Svaki periferni uređaj sadrži skup definiranih konfiguracijskih registara u svom PCI/PCIe konfiguracijskom prostoru. Ti registri se koriste ne samo za identifikaciju uređaja, već i za prijenos podataka.  Na primjer, ***Base Address Register*** u konfiguracijskom prostoru uređaja mora biti mapiran u RAM memoriju prije nego što uređaj može odgovoriti na pristup podacima.

## BAR (*Base Address Register*)

**BAR** (***Base Address Register***) adrese se koriste u komunikaciji između računala i *PCIe* kartice kako bi se omogućilo pristup memoriji na kartici. *PCIe* kartica može imati vlastitu memoriju koju koristi za pohranu podataka, poput konfiguracijskih informacija, operativnih parametara ili podataka koje procesor treba obraditi. Da bi procesor mogao pristupiti memoriji na *PCIe* kartici, treba znati koja je adresa memorije kojoj treba pristupiti.
Pri tome upravljački program (engl. *device driver*) inicijalizira *PCIe* karticu i konfigurira je kako bi postavila adrese memorije kojima se pristupa pomoću *BAR* registara na kartici. Kada centralni procesor (CPU) želi pristupiti memoriji na *PCIe* kartici, on koristi adresu memorije koju je postavio upravljački program u odgovarajući ***BAR*** registar.

Sumirajmo za što se sve koristi **BAR**:

- **Za pristup registrima uređaja** - omogućuje procesoru da pristupi kontrolnim i statusnim registrima uređaja što se koristi primjerice za konfiguraciju mrežne kartice ili upravljanje grafičkim procesorom. To radi tako da se registri uređaja mapiraju u memorijski prostor sustava putem BAR-a, pa procesor može čitati i pisati u te registre kao da su dio memorije sustava.
- **Pristup memoriji uređaja** – omogućuje procesoru pristup radnoj memoriji uređaja. Primjerice dio memorijskog prostora uređaja, npr. *framebuffer* grafičke kartice se mapira na adresni prostor koji procesor može koristiti za slanje podataka. To u konačnici znači da primjerice grafička kartica može imati vlastitu VRAM memoriju koja se mapira putem BAR-a.
- ***Razmjena podataka između uređaja i procesora*** – olakšava prijenos podataka između procesora i uređaja tako što BAR mapira memorijski prostor za brzu razmjenu podataka bez potrebe za dodatnom softverskom obradom. Primjerice uređaji koriste BAR omogućavajući procesoru ***DMA*** prijenosu podataka ili pristup skupovima naredbi koje procesor šalje uređaju ili uređaj procesoru kako bi izvršio određene operacije.
- **Konfiguracija uređaja** – koristi se za inicijalizaciju i konfiguraciju uređaja tijekom pokretanja sustava. OS ili *firmware* (BIOS) može čitati iz BAR-a kako bi saznao koja memorijska područja ili resurse uređaj zahtijeva. Naime, kada se tijekom pokretanja sustava otkrije uređaj na PCI/PCIe sabirnici, *firmware* (BIOS/UEFI) ili operativni sustav čita BAR registre uređaja kako bi saznao koliko memorijskog/I/O prostora uređaj zahtijeva. Na osnovu ovih informacija, *firmware* ili OS dodjeljuje odgovarajuće adrese u memorijskom prostoru sustava.
- **Mapiranje I/O prostora** – mapira ulazno-izlazne (I/O) portove uređaja u poseban adresni prostor (rijetko korišten na modernim sustavima). Koristi se za starije uređaje poput serijskih ili paralelnih portova tako što BAR definira I/O adrese, koje procesor koristi za slanje ili primanje podataka od uređaja.
- **Podrška za DMA (*Direct Memory Access*)** – omogućuje uređajima izravan pristup RAM memoriji sustava bez upotrebe procesora tako što BAR mapira memorijski prostor koji uređaj može koristiti za DMA prijenos podataka.
- **Proširivost za različite veličine memorijskog prostora** - omogućuje korištenje 32-bitnih ili 64-bitnih adresa za podršku uređajima koji zahtijevaju više memorijskog prostora. Primjerice moderni grafički procesori koriste 64-bitne BAR-ove za pristup velikim količinama VRAM memorije.

**Izvori informacija: (171),(173),(174),(175),(176),(177),(225),(1462),(1463),(1464),(1465),(1466),(1467),(1470),(1471),(1472),(1773),(1774),(1775), (K-9).**

# 10.2. (Re)konfiguracija sabirnice i uređaja na njoj

***Slijedi napredno poglavlje (10.2.x)!***
U određenim situacijama, kod visoko optimiziranih sustava, odnosno onih koje moramo znatno optimizirati, ponekad je potrebno konfigurirati uređaje na **PCI** sabirnici, na najnižoj razini, kako bi se njihov rad maksimalno optimizirao odnosno ubrzao.
U primjerima koji slijede, analizirat ćemo optimizacije preporučene za mrežne kartice koje rade na brzini od **100 Gbps**. Konkretno ćemo govoriti o *Mellanox Technologies MT27700* (***ConnectX-4***) mrežnoj kartici.
Prvo ćemo pronaći ovu karticu na *PCI express* sabirnici upotrebom naredbe `lspci` na sljedeći način:

```
lspci | grep Ethernet

04:00.0 Ethernet controller: Mellanox Technologies MT27700 Family [ConnectX-4]
```

Sada ćemo pogledat malo više detalja o ovom ***PCI** utoru*: `04:00.0` na koji je spojena ova kartica:

```
lspci -s 04:00.0 -vvv | egrep „LnkCap|LnkSta"

    LnkCap: Port #0, Speed 8GT/s, Width x8, ASPM not supported, Exit Latency L0s unlimited,
                L1 unlimited
   LnkSta: Speed 8GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
```

Oznaka `04:00.0` označava *PCI* sabirnicu i to redom: `04` označava sabirnicu, a `:00` označava prvi utor u njoj te `.0` označava funkcije ili uređaje: ako jedan uređaj ima više funkcija ili sučelja odnosno portova to će se ovdje razlikovati prema broju.
U našem ispisu zanima nas koja je propusnost i širina *PCI express* (*PCIe*) sabirnice za ovu karticu. Ovdje vidimo sljedeće:

- `LnkCap` - govori o tome koje mogućnosti kartica prijavljuje na sabirnici: `Speed 8GT/s, Width x8`. Dakle radi se o *PCI express x8* sabirnici.
- `LnkSta` - vidimo koji su parametri i stvarno u upotrebi: `Speed 8GT/s, Width x8`. Dakle koristi se *PCI express x8* što znači da je sve u redu jer su i prijavljeni parametri i oni koji su stvarno u upotrebi vidljivo identični.

Napomena vezana za *PCIe* propusnost, kod *x8* širine sabirnice:

- `Speed 2.5 GT/s` označava *Gen1 (generacija 1) PCIe sabirnice.*
- `Speed 5 GT/s` označava *Gen2 (generacija 2) PCIe sabirnice.*
- `Speed 8 GT/s` označava *Gen3 (generacija 3) PCIe sabirnice.*

Glavna razlika između *PCIe* generacija, osim podržane brzine, je kodiranje paketa i sâmo zaglavlje paketa, odnosno dio paketa na sâmoj sabirnici, koji ne nosi korisne već kontrolne podatke. Za generacije 1 i 2, svaki paket poslan na *PCIe* sabirnicu ima 20% *PCIe* zaglavlja. Ovo je poboljšano u generaciji 3, pri čemu je gornja granica zaglavlja smanjena na 1,5%.
Generacija *PCIe* sabirnice se može vidjeti i sa sljedećom naredbom (konkretno za uređaj: `04:00.0`):

```
lspci -s 04:00.0 -vvv |  grep PCIeGen
[V0] Vendor specific: PCIeGen3 x8
```



Identifikator ***PCI*** uređaja (takozvani ***PCI ID***) može nam dati naredba `lspci`, ali samo, ako koristimo i prekidač `-nn` pa ćemo u uglatoj zagradi uz svaki uređaj dobiti oznaku poput: `[14e4:1677]`. Zbog opisnog prikaza, ove ***PCI ID*** oznake se povezuju s bazom podataka u kojoj je lista svih *PCI* identifikatora i pripadajućih naziva uređaja, a koja se nalazi u datoteci: `/usr/share/hwdata/pci.ids`. Ova datoteka dolazi u softverskom paketu: `hwdata`, koji obično dolazi instaliran na sustav i stalno se nadograđuje.

Naime svaki uređaj spojen na sabirnicu računala ima zapisan svoj ***PCI ID*** identifikacijski broj koji se sastoji od dva heksadecimalna broja odvojena dvotočkom:

- Prvi dio broja (prije dvotočke) identificira proizvođača; primjerice: *IBM, Intel, Broadcom,…*
- Drugi dio broja točnu vrstu (tip) i model uređaja.



**Nemojte pobrkati navedeni PCI ID i oznaku sabirnice i utora u kojem se nalazi uređaj (kartica), mada izgledaju slično. PCI ID je identifikator uređaja (primjerice mrežne kartice, disk kontrolera i slično), a ne sabirnice.**



Pogledajte i poglavlja:
**11.1.2.1. Uređaji (*devices*) detaljnije** te poglavlje:
**13.9.2.1. *Systool* programski paket.**

## Problematične PCI ili PCI express kartice

Ovisno o vašem hardveru, u određenim konfiguracijama, obično poslužitelja, ako podržavaju takozvani **hot-plugging**, možda je moguće resetirati ili isključiti i ponovno uključiti PCI kartice pomoću posebnih programa ili hardverskog prekidača.
Ali ako nijedna od tih opcija ne postoji ili ne radi, sljedeće metode ponovnog pokretanja PCI kartica mogle bi vam biti korisne.

Dodatno, pitanje je je li kernel kompiliran s podrškom za takozvani ***hot-plug*** to jest uključivanje i isključivanje ***PCI/PCI express*** kartica tijekom rada računala (pod uvjetom da to podržavaju matična ploča, BIOS i sama kartica):

```
grep CONFIG_HOTPLUG_PCI /boot/config-`uname -r`

CONFIG_HOTPLUG_PCI_PCIE=y
CONFIG_HOTPLUG_PCI=y
CONFIG_HOTPLUG_PCI_ACPI=y
```

← Ove dvije opcije govore da je **hot-plug** u kernelu omogućen.

Za početak, za problematičnu **PCI** ili ***PCI express*** karticu trebat će nam njena PCI domena, sabirnica, oznaka uređaja i adresa uređaja. Pokrenimo naredbu `lspci`, kako bismo pronašli PCI adresu svoje željene kartice; primjerice mrežne kartice.
Pogledajmo naš slučaj u kojem imamo četiri ***PCI express*** mrežne kartice:

```
lspci | grep -i  Ethernet

04:00.0 Ethernet controller: Emulex Corporation OneConnect 10Gb NIC (be3) (rev 01)
04:00.1 Ethernet controller: Emulex Corporation OneConnect 10Gb NIC (be3) (rev 01)
04:00.2 Ethernet controller: Emulex Corporation OneConnect 10Gb NIC (be3) (rev 01)
04:00.3 Ethernet controller: Emulex Corporation OneConnect 10Gb NIC (be3) (rev 01)
```

← Ovdje vidimo tri mrežne kartice s njihovim PCI oznakama. U našem slučaju problematična kartica (koja je ostala u zaglavljenom stanju) je ona s PCI oznakom: `04:00.0`.

PCI uređaji su prema PCI oznakama (identifikatorima) vidljivi u posebnom ***/sys/*** pseudo datotečnom sustavu kernela.
Za konkretnu PCI oznaku je to direktorij: `/sys/bus/pci/devices/0000\:04\:00.0/`

Unutar tog (i drugih *PCI/devices*) direktorija postoji nekoliko posebnih datoteka:

- `reset` – neki uređaji omogućuju resetiranje svojih (određenih) funkcija. Postavljanjem vrijednosti jedan (1) u ovu datoteku resetiraju se njene funkcije, što je ponekad, ako se uređaj čudno ponaša, korisno.
- `reset_method` – za uređaje koji podržavaju gornju (`reset`) metodu, a nude više odvojenih funkcija u radu, moguće je koristiti i samo određenu funkciju uređaja koju želimo resetirati; odnosno nizati ih jednu po jednu (odvojene razmakom). Standardno su u ovoj datoteci navedene sve dostupne funkcije uređaja koje se mogu resetirati.
- `remove` – neki uređaji omogućuju i isključivanje samog uređaja odnosno takozvani ***hot-remove*** koji će odspojiti uređaj (karticu) sa PCI sabirnice, ako u ovu datoteku postavimo vrijednosti jedan (1). Da bi to radio računalo i sama kartica moraju podržavati **hot-plugging** uređaja na sabirnici.

Naime, ako imamo slučaj  u kojem se PCI kartica čudno ponaša, a ne želimo restartati cijelo računalo ili poslužitelj, imamo dvije opcije:

**1.** Resetiranje funkcija kartice, koje možemo napraviti ovako:

```
echo "1" > /sys/bus/pci/devices/0000\:04\:00.0/reset
```

Međutim, ako nam to ne pomogne, imamo sljedeću opciju!

**2.** Odspajanje kartice sa sabirnice: tako da će se PCI uređaj isključiti i odspojti (tzv. ***hot-remove***), što ćemo napraviti sa:

```
echo "1" > /sys/bus/pci/devices/0000\:04\:00.0/remove
```

Nako toga potrebno je pokrenuti kernel proceduru koja je ponovno skenirati sve PCI uređaje (kartice) spojene na ***PCI/PCI express*** sabirnicu.

Pričekat ćemo još par trenutaka da se PCI kartica uredno odspoji.
Potom pokrenimo:

```
echo "1" > /sys/bus/pci/rescan
```

Nakon ove naredbe kernelu, kernel će skenirati sve PCI uređaje te će ponovno pronaći našu ***PCI Express*** mrežnu karticu koju smo prethodno odspojili to jest isključili u radu.



Ovakvo isključivanje **PCI** ili **PCI Express** kartica i dalje ovisi o vašoj matičnoj ploči i BIOS-u, ali i o samoj kartici to jest podržava li ona ovakvo isključivanje i uključivanje (tzv. **hot-plugging**), tijekom rada računala.

**Izvor informacija:(1461),(1468),** `man lspci`,`man 7 pcilib`.

### 10.2.1. PCIe Max Payload Size

Vrijednost ***PCIe*** *Max Payload* veličine određuje maksimalnu veličinu *PCIe* paketa ili *PCIe MTU* (slično mrežnim protokolima i *MTU* na mrežnom protokolu). To znači kako su veće *PCIe* transakcije razbijene u pakete veličine tzv. ***PCIe MTU*** vrijednosti. Ovaj parametar postavlja sâmo sustav i ovisi o arhitekturi *chipseta* (npr. *X86_64*, *Power8*, *ARM*, itd.). Veličinu *PCIe Max Payload*, možete vidjeti pomoću naredbe `lspci`. Pogledajmo ovu vrijednost za našu karticu (`04:00.0`):

```
lspci -s 04:00.0 -vvv | grep DevCtl: -C 2

   DevCap: MaxPayload 512 bytes, PhantFunc 0, Latency L0s unlimited, L1 unlimited
    ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset+
    DevCtl: Report errors: Correctable- Non-Fatal+ Fatal+ Unsupported-
    RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+ FLReset-
    MaxPayload 256 bytes, MaxReadReq 4096 bytes
```

Dakle ova vrijednost je vidljiva u zadnjem redu: *`MaxPayload:`* `256 bytes` - vidimo kako je ovdje ona veličine 256 *bajta* i nju ne možemo mijenjati.

**Izvor informacija: `man lspci`.**

---

### 10.2.2. PCIe Max Read Request

***PCIe Max Read Request*** određuje maksimalni dopušteni zahtjev za čitanje sa *PCIe* sabirnice, za određeni uređaj (*karticu*) spojen na nju. Sâm *PCIe* uređaj obično prati broj zahtjeva čitanja koji su na čekanju, zbog toga što je potrebno pripremiti memorijski međuspremnik (engl. *Buffer*). Veličina zahtjeva za čitanjem *PCIe Max Read Request*, utječe na broj zahtjeva u redu čekanja za operacije čitanja sa sabirnice. Dakle ovo je praktično maksimalna veličina memorije koja se može dohvatiti u jednom zahtjevu za čitanje, sa *PCIe* sabirnice/uređaja.
Ponovno upotrijebite naredbu `lspci` kako biste provjerili vrijednost *PCIe Max Read Request* za već navedenu karticu (`04:00.0`):

```
lspci -s 04:00.0 -vvv | grep MaxReadReq

        MaxPayload 256 bytes, MaxReadReq 4096 bytes
```

Vidimo kako je ovaj međuspremnik trenutno postavljen na veličinu od 4096 *bajta* (`MaxReadReq  4096  bytes`), što je uglavnom i maksimalna moguća vrijednost. Svaka od ovih vrijednosti se pohranjuje u nekom od registara.

Ako želite vidjeti listu svih mogućih registara, za cijelu sabirnicu, koristiti ćemo naredbu: `setpci`. Stoga napišite:

```
setpci —dumpregs
```

Primjer ispisa (skraćen):

```
cap pos w name
     00 W VENDOR_ID
     02 W DEVICE_ID
     04 W COMMAND
     06 W STATUS
     08 B REVISION
     09 B CLASS_PROG
     0a W CLASS_DEVICE
     0c B CACHE_LINE_SIZE
     0d B LATENCY_TIMER
     0e B HEADER_TYPE
```

Opis stupaca:

- Prvi stupac (`cap`) — predstavlja oznaku mogućnosti uređaja (u našem primjeru ovaj stupac je prazan).
- Drugi stupac (`pos`) — pr. 04 je pozicija odnosno adresa registra koji nas zanima.
- Treći stupac (`w`) označava vrstu registra:
  - `.B` — bajt (*byte).*
  - `.W` — riječ (*word*).
  - `.L` — dugačka riječ (*long word*).
- Četvrti stupac (`name`) — ovdje se nalazi naziv konkretnog registra.

Dodatno, za one koje zanima još više detalja o registrima, možete vidjeti apsolutno sve registre dostupne određenom uređaju, sa sljedećom naredbom:

```
lspci -s 04:00.0 -xxxx
```

Pozivanjem prethodne naredbe, dobit ćete nešto poput:
```
04:00.0 Ethernet controller: Mellanox Technologies MT27700 Family [ConnectX-4]
00: df 10 20 07 46 05 10 00 10 00 00 02 10 00 80 00
10: 0c 00 f0 92 00 00 00 00 0c 00 d2 92 00 00 00 00
20: 0c 00 d0 92 00 00 00 00 00 00 00 00 3c 10 35 19
30: 00 00 00 00 40 00 00 00 00 00 00 00 0b 01 00 00
40: 01 48 e3 c1 08 00 00 00 11 c0 1f 80 00 20 00 00
50: 00 30 00 00 0f c6 c0 90 40 0f 01 c0 01 14 00 00
```
U našem slučaju, zanima nas registar samo za naš uređaj *PCIe ID:* `04:00.0` za *PCIe Max Read Request* funkcionalnost odnosno njegova vrijednost, koju možemo i mijenjati, ali oprezno.

Naravno sve je ovisno o uređaju i preporukama za određeni uređaj odnosno mrežnu karticu u našem slučaju.
Promijene se rade s naredbom `setpci`, za specifičan registar, prema posebnom načinu indeksiranja/označavanja, o kojem sada nećemo govoriti. Stoga detaljno proučite specifikacije konkretne mrežne kartice od strane proizvođača.

**Izvori informacija: (259),(260),(261),(262),(263), `man lspci`, `man setpci`.**

## 10.2.3. Nadogradnja baze podataka PCI identifikatora (PCI baze)

Svaki uređaj spojen na ***PCI*** (ili ***PCIe***)sabirnicu računala ima ugrađen svoj ***PCI ID*** identifikacijski broj koji se sastoji od dva heksadecimalna broja odvojena dvotočkom. Ovi brojevi se dekodiraju po imenu, upotrebom ***PCI ID*** baze.

Nadograditi (bazu) odnosno datoteku `/usr/share/hwdata/pci.ids` u kojoj je lista svih ***PCI ID*** identifikatora i pripadajućih naziva uređaja, možemo napraviti s naredbom `update-pciids`.

Pogledajmo samo mali djelić sadržaja ove PCI baze to jest datoteku: `/usr/share/hwdata/pci.ids`.
```
# This is a relabelled RTL-8139
        8139  AT-2500TX V3 Ethernet
. . .
0014  Loongson Technology LLC
        7a03  Gigabit Ethernet Controller
        7a10  Hyper Transport Bridge Controller
        7a14  EHCI USB Controller
        7a15  Vivante GPU (Graphics Processing Unit)
        7a19  PCI-to-PCI Bridge
        7a24  OHCI USB Controller
        7a29  PCI-to-PCI Bridge
0070  Hauppauge computer works Inc.
        7801  WinTV HVR-1800 MCE
. . .
```

Naredba `update-pciids` s kojom nadograđujemo PCI bazu dolazi u paketu `pciutils`.
Pogledajmo kako napraviti nadogradnju PCI baze uređaja, to jest datoteke `/usr/share/hwdata/pci.ids`:
```
update-pciids
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  251k  100  251k    0     0  1074k      0 --:--:-- --:--:-- --:--:-- 1081k
```
Nakon ove nadogradnje (s interneta), imat ćemo najnoviju inačicu navedene datoteke identifikatora ***PCI*** uređaja (*kartica*).



Identifikator ***PCI*** uređaja (takozvani ***PCI ID***) može nam dati naredba `lspci`, ali samo, ako koristimo i prekidač `-nn` pa ćemo u uglatoj zagradi uz svaki uređaj dobiti oznaku poput: `[14e4:1677]`. Zbog opisnog prikaza, ove ***PCI ID*** oznake se povezuju s bazom podataka u kojoj je lista svih *PCI* identifikatora i pripadajućih naziva uređaja, a koja se nalazi u datoteci: `/usr/share/hwdata/pci.ids`.
Ova datoteka dolazi u softverskom paketu: `hwdata`, koji obično dolazi instaliran na sustav i stalno se nadograđuje.

Naime svaki uređaj spojen na sabirnicu računala ima zapisan svoj ***PCI ID*** identifikacijski broj koji se sastoji od dva heksadecimalna broja odvojena dvotočkom:

- Prvi dio broja (prije dvotočke) identificira proizvođača; primjerice: *IBM, Intel, Broadcom,…*
- Drugi dio broja točnu vrstu (tip) i model uređaja.

**Nemojte pobrkati navedeni PCI ID i oznaku sabirnice i utora u kojem se nalazi uređaj (kartica), mada izgledaju slično.**

**Izvori informacija: (259),(260),(261),(262),(263), `man update-pciids`.**

## 10.2.4. Univerzalna serijska sabirnica (*Universal Serial Bus, USB*)

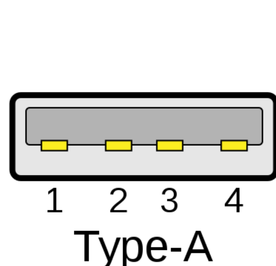
Za razliku od brze **PCI** ili ekstremno brzih inačica ***PCI express*** (**PCIe**) sabirnica, u novije vrijeme (od 1996.g.) pojavila se sabirnica zvana univerzalna serijska sabirnica (engl. *Universal Serial Bus*, **USB**). Njena prvotna namjena je bila zamjena *serijskog, paralelnog, PS/2, FireWire* i drugih sličnih sučelja, ali i olakšavanje umetanja (spajanja) i uklanjanja (odspajanja) vanjskih uređaja u radu, odnosno bez potrebe za gašenjem te ponovnim pokretanjem računala. *USB* je industrijski standard koji utvrđuje specifikacije za kabele i konektore te protokole za povezivanje, komunikaciju i napajanje (sučelja) između računala i perifernih uređaja te između drugih računala. Ovu sabirnicu odnosno standard su razvile tvrtke: *Compaq, DEC, IBM, Intel, Microsoft, NEC i Nortel*. Dalji razvoj *USB* standarda održava neprofitna organizacija ***USB Implementers Forum***.
Do danas je definirano nekoliko generacija *USB* specifikacija: **USB 1.x**, **USB 2.0**, **USB 3.x** i **USB4**.

Pogledajmo neke njihove karakteristike u tablici:

| **Konektor/standard** | **USB 1.0/1.1** | **USB 2.0** | **USB 3.0** | **USB 3.1** | **USB 3.2** | **USB 4** |
|---|---|---|---|---|---|---|
| **Propusnost** *(Efektivna propusnost je 20% manja od navedene zbog 8b/10b načina kodiranja signala)* | **1.5 Mbit/s** *(Low Speed)* **12 Mbit/s** *(Full Speed)* | **1.5 Mbit/s** *(Low Speed)* **12 Mbit/s** *(Full Speed)* **480 Mbit/s** *(High Speed)* | **5 Gbit/s** *(SuperSpeed)* | **10 Gbit/s** *(SuperSpeed+)* | **20 Gbit/s** *(SuperSpeed+)* | **40 Gbit/s** *(SuperSpeed+)* |
| **Maksimalna duljina kabela** *(Zbog el. signala i maksimalnog kašnjenja signala [round-trip])* | **5m** *(Full Speed)* **3m** *(Low Speed)* | **5m** *(High Speed)* | **3m** *(SuperSpeed)* | **3m** *(SuperSpeed+)* | **3m** *(SuperSpeed+)* | **3m** *(SuperSpeed+)* |

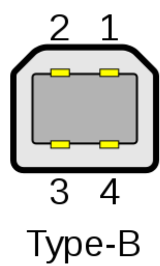
Vezano za *USB* konektore, postoji ih više vrsta (tipova), a najčešće su u uporabi: **A**(slika 45.1), **B**(slika 45.2) i **C**(slika 45.3).

**Slika 45.1. USB tip A.**



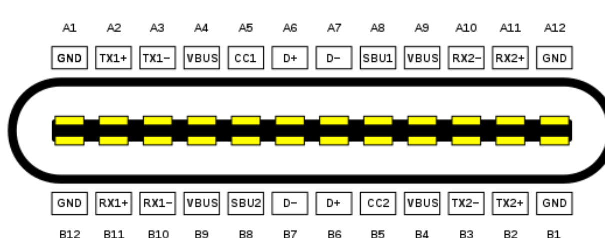

**Autor: Fred the Oyster (Wikimedia.org).**

**Slika 45.2. USB tip B.**




**Autor: Fred the Oyster (Wikimedia.org).**

**Slika 45.3. USB tip C.**




**Autor: Chindi.ap (Wikimedia.org).**

Osim navedenih vrsta konektora: **A**(slika 45.1), **B**(slika 45.2) i **C**(slika 45.3), u čestoj uporabi su i njihove manje inačice: **mini**: **A**, **B** i **AB** te **mikro**: **A**, **B** i **AB**, ali i još poneki drugi. Napajanje svih USB uređaja je standardno **5V** s dozvoljenim maksimalnim odstupanjem ±5%. Ograničenje napajanja uređaja navodi se u jedinici opterećenja (engl. *Unit load*) standardno 100 mA ili 150 mA za uređaje koji rade na *SuperSpeed* brzini.
Uređaji male snage mogu podnijeti najviše jednu jedinicu opterećenja (1x100mA), a svi se USB uređaji moraju konfigurirati kao uređaji male snage, tijekom inicijalizacije uređaja. USB uređaji velike snage moraju biti konfigurirani tako, da nakon inicijalizacije mogu povući do pet jedinica opterećenja (5x100mA=500 mA) ili čak do šest jedinica opterećenja (6x100mA=900 mA) za *SuperSpeed* uređaje.
Tamo gdje USB uređaji (primjerice mehanički diskovi velike brzine rada) zahtijevaju više energije nego što USB sučelje može ponuditi (više od 900mA), oni će funkcionirati neispravno, ili uopće neće raditi.

Naime preko USB-a se osigurava i napajanje USB uređaja, što u ovakvom slučaju neće biti moguće jer navedeni USB uređaj za stabilan rad zahtjeva veću snagu od dopuštene. Ipak, takvi uređaji mogu imati kabel u obliku slova **Y** koji ima dva USB priključka: jedan za napajanje i prijenos podataka, a drugi samo za napajanje, kako bi dvostruko USB napajanje (do maksimalno 2x900mA) bilo dovoljno.
Takav kabel je nestandardan sa specifikacijom usklađenosti s USB-om koja kaže da je upotreba **Y** kabela (kabel s dva A-utikača) zabranjena na bilo kojoj USB periferiji. To znači da, ako USB periferija zahtijeva više energije nego što dopušta USB specifikacija za koju je dizajniran, tada se on mora napajati vlastitim napajanjem.

Moguće je imati i portove (sučelja) s posebnom namjenom za USB punjenje baterije (engl. *Battery Charging*).
Ovo sučelje definira priključak za punjenje, koje može biti s podacima ili namjenski priključak za punjenje bez podataka.
Namjenski priključci za punjenje mogu se naći na USB adapterima za napajanje za pokretanje priključenih uređaja i baterija.

U srpnju 2012.g. *USB Promoters Group* najavila je finalizaciju specifikacije **USB Power Delivery** (***PD***) specifikacije (USB ***PD*** rev. 1), proširenje koje navodi korištenje certificiranih ***PD*** kabela s USB kabelom sa standardnim USB priključcima tipa **A** i tipa **B** (kasnije i **C**) za isporuku povećane snage.

Pogledajmo dio ovih specifikacija s posebnom namjenom za USB punjenje baterija, kako je vidljivo u tablici:

| Specifikacija | Jakost struje (**A**) | Napon (**V**) | Snaga (**W**) |
|---|---|---|---|
| Battery Charging (BC) 1.1 | 1,5 A | 5 V | 7,5 W |
| Battery Charging (BC) 1.2 | 5 A | 5 V | 25 W |
| USB-C | 3 A | 5 V | 15 W |
| | | | |
| Power Delivery 1.0 *Micro-USB* | 3 A | 20 V | 60 W |
| Power Delivery 1.0 *Type-A/B* | 5 A | 20 V | 100 W |
| Power Delivery 2.0/3.0 *Type-C* | 5 A | 20 V | 100 W |
| Power Delivery 3.1 *Type-C* | 5 A | 48 V | 240 W |

Vratimo se na standardni USB. USB je dizajniran za standardizaciju povezivanja perifernih uređaja s osobnim računalima, kako za komunikaciju tako i za opskrbu električnom energijom. U velikoj je mjeri zamijenio sučelja poput *serijskih* i *paralelnih* portova te je postao uobičajen na širokom rasponu uređaja. Primjeri perifernih uređaja koji su povezani putem USB-a uključuju računalne tipkovnice i miševe, video kamere, pisače, skenere, prijenosne medijske *playere*, mobilne (*smart*) telefone, diskove, mrežne adaptere, 4G/5G modeme i mnoge duge uređaje.

Standardni USB konektori su dizajnirani da budu robusniji od prethodnih vrsta konektora. To je zato što je USB i zamišljan da se USB uređaji mogu na živo (bez gašenja računala) uključivati i isključivati [engl. *Hot plug*], a samim time bi se konektori češće koristili, od prethodnih vrsta konektora. Standardni USB ima minimalni životni vijek od 1.500 ciklusa umetanja i uklanjanja, mini-USB utičnica povećava to na 5.000 ciklusa, dok su noviji Micro-USB i USB-C priključci dizajnirani za minimalni životni vijek od 10.000 ciklusa umetanja i uklanjanja.

USB ima asimetrični dizajn i sastoji se od poslužitelja (engl. *hosta*) i više jedinica koje se povezuju s poslužiteljem kao grane, stvarajući zvjezdastu strukturu. Međutim, na poslužitelj se može spojiti poseban koncentrator (*USB Hub*), na koji se mogu povezati druge grane, što stvara strukturu stabla. Prema standardu, pomoću USB-a moguće je imati do pet razina grananja po poslužitelju na kontroler, a moguće je ukupno povezati do 127 uređaja, umanjeno za svako čvorište (koncentrator) spojeno na isti poslužitelj. Koncentrator (*USB HUB*) također može imati vlastito napajanje ili koristiti napajanje iz USB sabirnice.
U prvom slučaju, iz USB sabirnice se preuzima maksimalno trenutno opterećenje i omogućuje svakom spojenom uređaju da ima jedinično opterećenje. S druge strane koncentrator sa svojim napajanjem, svakom uređaju može pružiti maksimalnu snagu koju mu daje njegovo napajanje.

**Izvori informacija: (985),(986),(987).**

## 10.2.4.1. USB i Linux

USB uređaje spojene na računalo, kao i sav ostali hardver, prepoznaje ***udev***(d) servis, na Linuxu.



Vezano za **udev** servis, pogledajte poglavlje:
**11.1.2.1. Uređaji (*devices*) detaljnije.**

Naime nakon što uključimo ***USB*** uređaj u *USB* utor na računalu, ***udev***(d) servis će prepoznati *USB* hardver na osnovi njegovog **USB ID** broja odnosno njegove jedinstvene oznake. Slično kao što svaki ***PCI*** uređaji (pr. grafička, mrežna, zvučna kartica i sl.) spojen na *PCI* sabirnicu ima svoj **PCI ID** identifikator (sâmog) uređaja, tako i USB uređaji moraju imati svoj jedinstveni **USB ID** identifikator. Naime na osnovu **USB ID** identifikatora, ***udev***(d) servis prepoznaje *USB* hardver, tako da gleda u posebnu tablicu s **USB ID** identifikatorima, te pomoću nje prepoznaje o kojem se uređaju radi, a potom za njega učitava potreban upravljački program (kernel modul). **USB ID** baza se nalazi pohranjena u datoteci: `/usr/share/hwdata/usb.ids`.

### Naredba lsusb

Naredba `lsusb` koristi se za izlistanje USB uređaja spojenih na USB sabirnicu računala.
Ona dolazi u softverskom paketu ***usbutils***, koji ćemo sada instalirati:

```
yum -y install usbutils
```

Nakon toga, naredba `lsusb` je spremna za korištenje. Pokrenimo ju bez ikakvih opcija i prekidača:

```
lsusb
Bus 005 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 004 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

U ispisu vidimo samo USB koncentratore (USB `root hub`) na koje još nije spojen niti jedan USB uređaj.

Sada ćemo spojiti USB mrežnu karticu (`ASIX Electronics Corp. AX88772B`), pa ponovno provjerimo stanje:

```
lsusb
Bus 005 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 004 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 0b95:772b ASIX Electronics Corp. AX88772B
```

Vidimo da se ona pojavila na kraju ispisa. Pogledajmo i malo više detalja o njoj, prema njenom ***USB ID***-u (`-d` prekidač):

```
lsusb -d 0b95:772b -v

Bus 002 Device 002: ID 0b95:772b ASIX Electronics Corp. AX88772B
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass          255 Vendor Specific Class
  bDeviceSubClass       255 Vendor Specific Subclass
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x0b95 ASIX Electronics Corp.
  idProduct          0x772b AX88772B
  MaxPower              200mA
  Device Status:     0x0000
  . . .
```

Ispisali smo samo neke vrijednosti, koje smo dobili; pa tako vidimo proizvođača USB uređaja (pod `idVendor`), model uređaja (pod `idProduct`), maksimalnu striju koju uređaj može povući (pod `MaxPower`) i drugo.

## Nadogradnja baze podataka USB identifikatora (USB ID baze)

Slično kao što je s vremena na vrijeme potrebno nadograditi **PCI ID** bazu, isto tako je potrebno i nadograđivati bazu **USB ID** identifikatora. Pogotovo ako vaš novi USB uređaj nije prepoznat od strane sustava. ***USB ID*** baza se nalazi pohranjena lokalno na disku u datoteci: `/usr/share/hwdata/usb.ids`, a globalno se održava i osvježava (za nove uređaje) na Internet repozitoriju, na stranici: http://www.linux-usb.org/usb-ids.html. Odnosno njena zadnja inačica se nalazi u datoteci: http://www.linux-usb.org/usb.ids.



Vezano za **PCI ID** bazu **PCI** identifikatora, pogledajte poglavlje:
**10.2.3. Nadogradnja baze podataka PCI identifikatora (PCI baze).**

Struktura podataka unutar ove baze gotovo je identična ***PCI ID*** bazi, dakle ***USB ID*** baza sadrži identifikatore USB uređaja, koji se sastoje od nekoliko brojčanih oznaka, od kojih prva (najviša razina hijerarhije) označava proizvođača. Svaki proizvođač ima 2-bajtni ID, pohranjen u bazi podataka kao 4 heksadecimalne znamenke (s malim slovima). Primjerice oznaka **04e8** označava tvrtku ***Samsung Electronics Co***. Svaki proizvođač vodi popis svojih uređaja i dodjeljuje im ID-ove.
Svaki uređaj na popisu ima (sljedeći) 2-bajtni ID, pohranjen na isti način kao i ID proizvođača (4 heksadecimalne znamenke). Primjerice cijela ***USB ID*** oznaka: **04e8 3004** označava redom: tvrtku ***Samsung***, i to njen USB pisač ***ML-4600***.



Identifikator ***USB*** uređaja (takozvani ***USB ID***) može nam dati naredba `lsusb`, pa ćemo uz svaki uređaj, nakon oznake za sabirnicu (*Bus*) i uređaj (*Device*), dobiti ***USB ID*** oznaku poput: `[1d6b:0002]`. Zbog opisnog prikaza, ove ***USB ID*** oznake se povezuju s bazom podataka u kojoj je lista svih *USB* identifikatora i pripadajućih naziva uređaja, a koja se nalazi u datoteci: `/usr/share/hwdata/usb.ids`. Ova datoteka dolazi u softverskom paketu: `hwdata`, koji obično dolazi instaliran na sustav i stalno se nadograđuje.

Naime svaki uređaj spojen na sabirnicu računala ima zapisan svoj ***USB ID*** identifikacijski broj koji se sastoji od dva heksadecimalna broja odvojena dvotočkom. Prvi dio broja (prije dvotočke) identificira proizvođača; primjerice: *IBM, Intel, Broadcom*,… a drugi dio broja točnu vrstu (tip) i model *USB* uređaja.

---

Za nadogradnju baze ***USB ID*** oznaka, dovoljno je osvježiti softverski paket `hwdata`, odnosno cijelu distribuciju Linuxa:

```
yum -y update
```

U ekstremnim slučajevima možete i sami kopirati najnoviju ***USB ID*** bazu s Internet lokacije: http://www.linux-usb.org/usb.ids u datoteku: `/usr/share/hwdata/usb.ids`. Preporuka je prije ove operacije, napraviti kopiju stare ***USB ID*** baze.

**Izvori informacija: (985),(986),(987), `man lsusb`.**

---

## 10.2.5. System Management Bus (SMBus) sabirnica i DMI/SMBIOS

*System Management Bus* **(SMBus)** sabirnica odnosnos sabirnica za upravljanje sustavom je dizajnirana kao jednostavna dvožilna sabirnica. **SMBus** je jednostavna dvosmjerna serijska komunikacijska sabirnica, a koristi se za komunikaciju s različitim perifernim uređajima u računalima i drugim elektroničkim uređajima.
Najčešće se nalazi implementirana u čipovima matičnih ploča za komunikaciju s uređajima niske propusnosti.
**SMBus** se koristi za različite zadatke u računalima, uključujući:

- Praćenje stanja sustava – omogućuje praćenje temperaturnih senzora (**CPU**, **GPU**, …), naponskih razina, brzina ventilatora, i drugih kritičnih parametara hardvera.
- Baterijske upravljačke sustave – koristi se u prijenosnim računalima za komunikaciju s baterijskim modulima, omogućujući praćenje napunjenosti baterije, zdravstvenog stanja baterije i drugih parametara.
- Kontrolu ventilatora – SMBus omogućuje kontrolu brzine ventilatora na temelju temperaturnih očitanja.
- EEPROM - može se koristiti za pristup EEPROM-ima (programabilnim memorijama za čitanje), gdje se pohranjuju različite konfiguracijske postavke sustava.
- Očitanje informacija iz *Serial Presence Detect* (**SPD**) čipova na RAM modulima (memoriji), uključujući kapacitet, brzinu, i druge parametre RAM memorije.
- *Power Supply Units* (**PSUs**) – za praćenje statusa napajanja, izlaznog napona, struje i temperature napajanja.

Međutim točna funkcionalnost i hardverska sučelja razlikuju se ovisno o proizvođaču.
SMBus su definirali tvrtke ***Intel*** i ***Duracell*** 1994. godine. Ova sabirnica prenosi signal takta, podatke i intrukcije, a temelji se na ***Philipsovom*** **I²C** protokolu serijske sabirnice.


Veza između **I2C**-a i **SMBus**-a je u tome što su kompatibilni na fizičkoj razini (koriste iste linije **SDA** i **SCL**), ali **SMBus** dodaje dodatna pravila i protokole iznad osnovnih **I2C** protokola.

**SMBus preko PCI/PCIe sabirnice**
***SMBus*** veza je moguća i preko **PCI** ili **PCIe** sabirnice i može se koristiti za razne zadatke upravljanja sustavom i monitoringa rada sustava. U načelu postoje dva načina integracije s PCI/PCIe uređajima:

- ***SMBus*** je često integriran u čipset matične ploče. Povezuje razne komponente na matičnoj ploči, kao što su senzori, EEPROM, ali i drugi periferni uređaji, omogućujući komunikaciju između sustava i tih uređaja.
- I sami ***PCI/PCIe*** uređaji mogu imati *SMBus* sučelje koje omogućuje operativnom sustavu i softveru interakciju s upravljačkim funkcijama tih uređaja. Primjerice, mrežna kartica ili grafička kartica mogu preko *SMBus-a* pružati informacije o temperaturi i drugim parametrima.

**Način komunikacije**
Čipset ili namjenski kontroler djeluju kao ***SMBus master***, inicirajući komunikaciju s ***SMBus slave*** uređajima, koji uključuju senzore i druge uređaje. Uređaji na ***SMBus-u*** imaju jedinstvene adrese. *SMBus master* koristi ove adrese za komunikaciju sa specifičnim uređajima koristeći standardne protokole za čitanje i pisanje podataka.

**PCI** i **PCIe** uređaji imaju konfiguracijski adresni prostor kojem operacijski sustav može pristupiti.
Dio tog prostora može se koristiti za mapiranje **SMBus** kontrolera i njegovih registara.
U radu operativni sustav zahtijeva specifične upravljačke programe (kernel module) za pristup *SMBus* kontroleru i komunikaciju s uređajima. Primjerice, u Linuxu se često za *Intel* čipsete koristi `i2c-i801` kernel modul.

**Način rada**
Prilikom pokretanja sustava detektira se ***SMBus*** kontroler (i **I²C**); primjerice detekcija je vidljiva s naredbom `dmesg`:

```
dmesg |egrep -i "i2c|smbus"
[    0.456728] i2c_dev: i2c /dev entries driver
[    0.828156] i801_smbus 0000:00:1f.3: SPD Write Disable is set
[    0.828177] i801_smbus 0000:00:1f.3: SMBus using PCI interrupt
[    0.835657] i2c i2c-0: 4/4 memory slots populated (from DMI)
[    0.836056] i2c i2c-0: Successfully instantiated SPD at 0x50
[    0.836602] i2c i2c-0: Successfully instantiated SPD at 0x51
[    0.836941] i2c i2c-0: Successfully instantiated SPD at 0x52
[    0.837434] i2c i2c-0: Successfully instantiated SPD at 0x53
```

U ispisu vidimo sljedeće:

- Vidimo da je pronađen ***SMBus*** kontroler, na PCI sabirnici (`00:1f.3`).
- Slijedi dio vezan za **I2C (**samim time i za **SMBus)**:
  - Vidimo podatke o popunjenosti urora za RAM memoriju (`4/4 memory slots populated`).
  - Pronađeni **SPD** su čipovi na memorijskim (RAM) modulima (*`Successfully instantiated SPD`*).

Od tog trenutka sustav je pronašao (ili pokušao) pronaći njegov upravljački program i učitati ga; što konkretno vidimo sa:

```
lspci -s 00:1f.3 -vvv
00:1f.3 SMBus: Intel Corporation 8 Series/C220 Series Chipset Family SMBus Controller
        Subsystem: Dell 8 Series/C220 Series Chipset Family SMBus Controller
        Control: I/O+ Mem+ BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr- Stepping-
        Status: Cap- 66MHz- UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort-
        Interrupt: pin C routed to IRQ 18
        IOMMU group: 9
        Region 0: Memory at f7c39000 (64-bit, non-prefetchable) [size=256]
        Region 4: I/O ports at f040 [size=32]
        Kernel driver in use: i801_smbus
        Kernel modules: i2c_i801
```

← Vidimo koji upravljački program se koristi kao kernel modul (`i2c_i801`).

Naime dio upravljačkog programa je uobičajeno već integriran u kernel, što se može vidjeti sa:

```
grep -i i2c /boot/config-`uname -r` | grep -i "=Y"
```

← Jedan od integriranih upravljačkih programa u kernelu je (`CONFIG_I2C`).

Pogledajmo je li upravljački program koji je dostupan kao kernel modul (`i2c_i801`) učitan:

```
lsmod | grep i2c_i801
i2c_i801               36864  0
i2c_smbus              16384  1 i2c_i801
```

← Vidimo da je uredno učitan te da je uz njega učitan i `i2c_smbus`.

Sada instalirajmo dodatni softver za **SMBus** (i **I2C**):

```
yum -y install i2c-tools
```

Sada možemo pokrenuti detekciju **SMBus/I2C** uređaja s novom naredbom `i2ctools`:

```
i2cdetect -l

i2c-0   smbus           SMBus I801 adapter at f040              SMBus adapter
i2c-1   i2c             i915 gmbus ssc                          I2C adapter
i2c-2   i2c             i915 gmbus vga                          I2C adapter
i2c-3   i2c             i915 gmbus panel                        I2C adapter
i2c-4   i2c             i915 gmbus dpc                          I2C adapter
i2c-5   i2c             i915 gmbus dpb                          I2C adapter
i2c-6   i2c             i915 gmbus dpd                          I2C adapter
i2c-7   i2c             AUX C/DDI C/PHY C                       I2C adapter
i2c-8   i2c             AUX D/DDI D/PHY D                       I2C adapter
```

← U ispisu vidimo sve **I2C/SMBus** uređaje na sabirnici.
← Svi ovi prepoznati uređaji i senzori pojavit će se kao datoteke u `/dev/` direktoriju (pr. `/dev/i2c-0`, `/dev/i2c-1`, …).

Sada možemo instalirati programski paket za rad sa senzorima:

```
yum -y install lm_sensors
```

← Nakon ovog paketa dobivamo i još nekoliko korisnih naredbi.

Krenimo s pronalaženjam ***SMBus/I2C*** senzora na raznim sabirnicama, s naredbom `sensors-detect` (ispis smo skratili):

```
sensors-detect
Intel digital thermal sensor...                         Success!
    (driver `coretemp')
Some Super I/O chips contain embedded sensors. We have to write to
standard I/O ports to probe them. This is usually safe.
Do you want to scan for Super I/O sensors? (YES/no): y
Next adapter: SMBus I801 adapter at f040 (i2c-0)
Do you want to scan it? (YES/no/selectively): y
Next adapter: i915 gmbus ssc (i2c-1)
Do you want to scan it? (yes/NO/selectively): y
Next adapter: AUX C/DDI C/PHY C (i2c-7)
Do you want to scan it? (yes/NO/selectively): y
Next adapter: AUX D/DDI D/PHY D (i2c-8)
Do you want to scan it? (yes/NO/selectively): y
```

← Na sva pitanja o prepoznavanju sabirnica i senzora na njima odgovaramo sa **`y`**.

```
Now follows a summary of the probes I have just done.
Just press ENTER to continue:
```

← Na kraju potvrđujemo sa *ENTER*. Zatim slijedi još jedno pitanje:

```
Do you want to add these lines automatically to /etc/modules? (yes/NO)yes
```

← I ovdje potvrđujemo sa (**`yes`**), da se potrebni kernel moduli automatski učitavaju tijekom pokretanja sustava.
To znači da će ih ova skripta dodati u datoteku `/etc/modules`.

Cjelokupna konfiguracija `sensors-detect` procedure se upisuje u datoteku: `/etc/sensors3.conf` ili `/etc/sensors.conf`.
Sada s naredbom `sensors`, možemo vidjeti očitanja senzora koje imamo dostupne na našem računalu:

```
sensors
acpitz-acpi-0
Adapter: ACPI interface
temp1:        +27.8°C
temp2:        +29.8°C

Adapter: ISA adapter
Package id 0:  +36.0°C  (high = +80.0°C, crit = +100.0°C)
Core 0:        +33.0°C  (high = +80.0°C, crit = +100.0°C)
Core 1:        +31.0°C  (high = +80.0°C, crit = +100.0°C)
Core 2:        +30.0°C  (high = +80.0°C, crit = +100.0°C)
Core 3:        +30.0°C  (high = +80.0°C, crit = +100.0°C)
```



Vezano za primjenu **SMBus** sabirnice, pogledajte i poglavlje:
**26.8.3. Intelligent Platform Management Interface (IPMI).**

## Desktop Management Interface (DMI) i SMBIOS

DMI je često korišten termin kada se govori o ***SMBIOS*** informacijama, ali ovdje se ne radi o *Direct Media Interface-u*, već o ***Desktop Management Interface.*** Naime **DMI** je stari naziv za ***SMBIOS***. Kada se govori o *DMI* u kontekstu SMBIOS-a, zapravo se misli na prikupljanje i organizaciju sistemskih informacija pomoću SMBIOS standarda a ne o *Direct Media Interface* sučelju između procesora i čipseta. **SMBIOS** je standard razvijen od strane *Distributed Management Task Force* (**DMTF**) koji definira način na koji BIOS (ili UEFI) sustav može pružiti informacije o hardveru i hardverskim komponentama operativnom sustavu.
**SMBIOS** koristi *DMI* tablice (*DMI tables*) za pohranu i organizaciju ovih podataka. Ove tablice su strukturirane tako da olakšavaju pristup i čitanje informacija o sustavu. Ovdje se nećemo fokusirati na detalje rada *DMI*, već na DMI sučelje prema Linuxu. Naime, na Linux sustavima, *DMI* informacije o računalu dostupne su nam putem *sysfs* datotečnog sustava.
Ove informacije uključuju detalje o hardverskoj konfiguraciji sustava, kao što su proizvođač, model, serijski broj, verzija BIOS-a, itd. To je korisno za svrhe inventarizacije, dijagnostike, ali i upravljanja sustavom.

**Kako radi SMBIOS?**

- BIOS/UEFI – kada se sustav pokreće, BIOS ili UEFI prikupljaju podatke o hardveru i pohranjuju ih u SMBIOS tablice.
- Pristup iz operativnog sustava - operativni sustav, poput Linuxa, može pristupiti ovim tablicama kako bi prikupio informacije o hardveru. Ove informacije se mogu koristiti za dijagnostiku, inventarizaciju i upravljanje sustavom.

*DMI* informacijama možete pristupiti kroz `/sys/class/dmi/id/` direktorij.
Drugi i jednostavniji način isčitanja podataka o komponentama računala iz DMI podataka, jest pomoću naredbe: `dmidecode`.
Pogledajmo znatno skraćen ispis ove naredbe:

```
dmidecode

BIOS Information:
        Vendor: Dell Inc., Version: A25 , . . .        . . .
Base Board Information:
        Serial Number: /CPVN6Z1/CN722003820142/, . . .     . . .
Memory Device
        Size: 8 GB, Type: DDR3, Serial Number: 0250CC01A014, . . .
```



Vezano za primjere rada naredbe `dmidecode`, pogledajte poglavlja:
**3.4.3. Osnovne naredbe vezane uz operativni sustav i komponente računala.**
**12.3.3.7. Naredba *dmidecode*.**



Softver za virtualizaciju također može generirati **SMBIOS** tablice za korištenje unutar virtualnih računala!

**Izvori informacija: (1668),(1669),(1670),(1671),(1672),(),(),(),** **`man i2ctools`**, **`man sensors-detect`**, **`man sensors`**, **`man 5 sensors.conf`**, **`man 5 modules-load.d`**, **`man dmidecode`**.

# 10.3. Tipkovnica, kodne stranice, *enkodiranje* i drugo

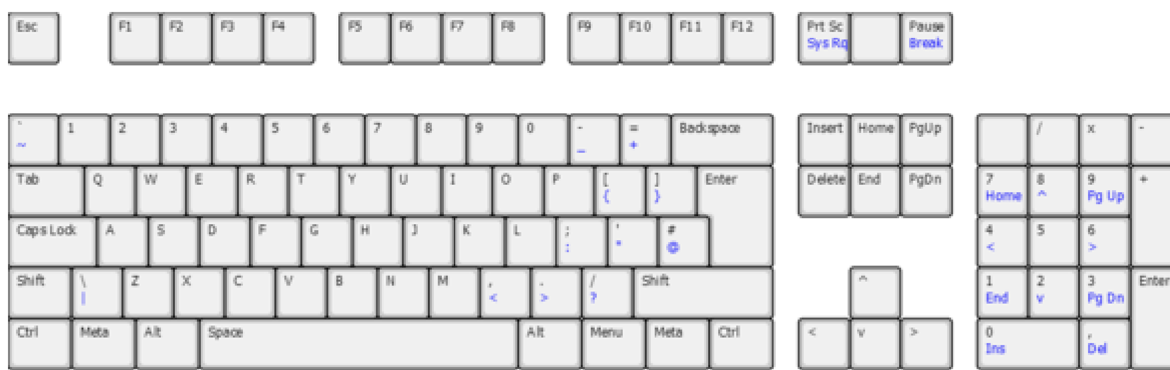
## 10.3.1. Tipkovnica

Tipkovnica pripada kategoriji ulaznih uređaja. Bez obzira na vrstu ili način spajanja, bilo to preko *PS/2*, *USB* ili nekog drugog sučelja, rad svih tipkovnica je isti. Sve tipkovnice na sebi imaju tipke na kojima su ugravirana ili otisnuta slova, brojevi i posebni znakovi. Osim navedenih osnovnih tipki, postoje i tipke za posebne namjene:

- Poput funkcijskih tipki (`F1` do `F12`).
- Tipke za prebacivanje između velikih i malih slova (engl. *Caps lock*).
- Tipki s posebnom funkcionalnošću kao što su: `Shift`, `Ctrl`, `Alt`, `Alt Gr`, `Print Scrn`, `SysRq`, `ESC`
- Te ostalih tipki poput: `Insert`, `Delete`, `Home`, `Pause`/`Break` i drugih.
- I numeričkog dijela tipkovnice, s brojevima i osnovnim matematičkim operatorima (`+`, `-`, `*`, `/`).

Izgled standardne *ISO* tipkovnice sq *QWERTY* rasporedom tipki, vidljiv je na slici 46.

**Slika 46. Izgled standardne ISO tipkovnice sa QWERTY rasporedom tipki.**




**Izvori informacija: (178),(179)**

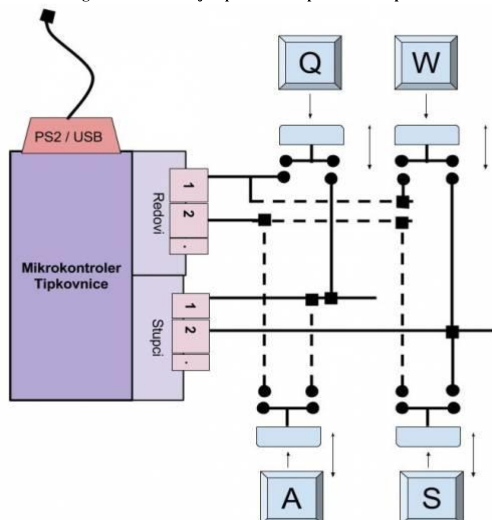
### 10.3.1.1. Od čega se sastoji tipkovnica ?

Tipkovnica se sastoji od:

- Tipki i pripadajućih prekidača (sklopki).
- *Mikrokontrolera* tipkovnice.
- Utikača (PS/2, USB, …).

Tipkovnica se sastoji od tipki ispod kojih se nalaze prekidači odnosno sklopke.
Bez obzira radi li se o membranskoj, mehaničkoj ili nekoj drugoj tipkovnici, pritiskom na tipku aktivira se njen prekidač koji zatvara strujni krug koji predstavlja stisnutu tipku, kako je vidljivo na slici 47.

**Slika 47. Logička shema dizajna prekidača tipki unutar tipkovnice.**




Svaki pritisnuti prekidač *mikrokontroleru* (procesoru tipkovnice) daje podatak o poziciji tipke koja je stisnuta. Slika 47. prikazuje logičku shemu rada tipkovnice.

Naime stiskanjem pojedine tipke zatvara se jedan vertikalni i jedan horizontalni strujni krug, koji su pod kontrolom *mikrokontrolera* tipkovnice.
Za više detalja o radu tipkovnice pogledajte sljedeće poglavlje.

**Izvori informacija: (178),(179),(181)**

### 10.3.1.2. Kako radi tipkovnica?

***Slijedi napredna cjelina (10.3.1.2.x)!***
*Mikrokontroler* tipkovnice na osnovi pozicije stisnute tipke, odnosno reda i stupca strujnog kruga kojeg zatvara pojedina pritisnuta tipka, "zna" koja je tipka stisnuta. Svaka tipka stiskanjem zatvara strujni krug određenog reda i strujni krug određenog stupca kao što je vidljivo na slici 47. Tako se prema slici, primjerice stiskanjem tipke **Q** zatvaraju strujni krugovi prvog reda i prvog stupca. Drugim riječima svaka tipka stiskanjem zatvara strujne krugove "x" i "y" osi.
U ovom pojednostavljenom primjeru dobilo bi se očitanje pozicije stisnute tipke odnosno koordinata: **x**1, **y**1.



Mikrokontroler tipkovnice skenira sve redove i stupce u pravilnim vremenskim intervalima. Tako je za **USB** tipkovnice interval skeniranja između 18.77ms i 32.75ms dok je za **PS/2** tipkovnice interval između 2.83ms i 10.88ms dakle znatno kraći. To u praksi znači da su **PS/2** tipkovnice "*responzivnije*" odnosno da se pritisnuta tipka brže registrira.

Stoga budimo svjesni činjenice kako se stisnuta tipka ne očitava isti tren kada je stisnuta već u prvom sljedećem intervalu u kojem ju očitava mikrokontroler tipkovnice. To znači kako kod najsporijih modela USB tipkovnica, ako ste stisnuli tipku, a tek je prošao vremenski interval skeniranja, potrebno je do 33 ms (33 tisućinke sekunde) da tijekom sljedećeg intervala skeniranja, stisnuta tipka bude registrirana. Kada je tipka očitana od mikrokontrolera tipkovnice, on gleda u svoju matricu tipki (engl. *Key matrix*), kako bi pronašao koja je tipka stisnuta.
Potom ju uspoređuje s tablicom koja se zove *Character Map* tablica, a koja je pohranjena u njegovoj ROM memoriji.
Tako se dobiva jedinstveni kôd za stisnutu tipku odnosno takozvani ***Character code***.
Mikrokontroler tipkovnice tada traži signal prekida (engl. *Interrupt request [IRQ]*) od centralnog procesora (CPU).
Kernel operativnog sustava "hvata" zahtjev za prekid (*IRQ*) i kada naš prekid dođe na red, preuzima ga.
Preuzeti zahtjev se prosljeđuje prema upravljačkom programu (engl. *Driver*) za tipkovnicu.
Tek tada operativni sustav *Character code* stisnute tipke učitava preko mikrokontrolera tipkovnice.



Operativni sustav u procesu obrade *Character code*-a, može koristiti svoju tablicu karaktera (slova/brojeva/znakova) te preći (pregaziti) preko one koju je dobio od mikrokontrolera tipkovnice. Ovo je korisno ukoliko se koriste tipke na tipkovnici (i njeni pripadajući kôdovi) koje nemaju ispravan ekvivalent na trenutnim (tzv. *regionalnim*) postavkama našeg Linux sustava.

#### 10.3.1.2.1. Što se događa u ovom koraku ?

U trenutku kada smo dobili numerički kôd, koji predstavlja stisnutu tipku, a taj kod se zove *Scan code*, on se potom preslikava u tzv. *Key code* koji predstavlja svako pojedino slovo/znak/broj ili funkcijsku tipku.



Definicija tipkovnice s rasporedom te regionalnih postavki tipki, nalazi se u datoteci:
`/etc/sysconfig/keyboard`.

Sâma definicija tipkovnice se automatski podešava kod instalacije Linuxa.
U datoteci: `/etc/sysconfig/keyboard` se nalazi nekoliko postavki:

- `KEYBOARDTYPE="sun|pc"` — moguće opcije su ili "**SUN**" ili "**PC**", a u većini slučajeva to je uvijek PC
  tu se još nalaze i:
  - `MODEL=` kojim se definira točan model tipkovnice.
  - `LAYOUT=` kojim se definira raspored tipki na tipkovnici.
- `KEYTABLE="<file>"` — označava upravo tablicu s preslikavanjima tipki o kojoj smo maloprije govorili.
  Unutar direktorija: `/lib/kbd/keymaps/i386/` nalaze se pod direktoriji s rasporedom tipki; pa tako imamo: `querty`, `quertz`, `azerty` itd.. Unutar tih direktorija se nalazi definicija tipki za pojedine modele tipkovnica.
  Za hrvatski raspored tipki i "naših" znakova (***š đ č ć ž***) koristi se tablica definirana unutar datoteke `croat.map.gz` koja je zapravo komprimirana datoteka `croat.map`, a koja se koristi u slučaju kada koristimo hrvatsku tipkovnicu.

**Za RedHat/Centos 7.x** imamo male razlike u odnosu na inačicu **6.x**, pa se konfiguracija nalazi u više konfiguracijskih datoteka koje se ne moraju mijenjati ručno već s naredbom `localectl`. Pogledajmo kako konfigurirati Hrvatske postavke:

```
localectl set-keymap hr
```

Ako imamo potrebu, listu svih tablica tipkovnica možemo dobiti s naredbom:

```
localectl list-keymaps
```

Dok trenutne regionalne postavke možemo vidjeti sa sljedećom naredbom:

```
localectl status
```



Pogledajte poglavlje:
**7.3.2.5. Drugi korisni programi i komponente unutar *systemd* paketa** i to posebno cjelinu:
„***Podešavanje lokalnih postavki sustava***".



Preslikavanje (mapiranje) *Key codea* u ***Character set*** se zove (**en)kodiranje**.

Sada možemo i provjeriti koja tablica je učitana i koristi se, sve s pomoću naredbe `dumpkeys`:

```
dumpkeys > my_keymap
```

Dodatno sve možemo snimiti u datoteku: `my_keymap` te ju prema potrebi pregledati kasnije.

```
dumpkeys > my_keymap
```



Tablica s mapiranjima (poveznicama) ***Scan Code*** u ***Key code***(181) ovisi o tipkovnici koju koristimo.

**I na kraju sumirajmo**
Što se događa kod pritiskanja tipke na tipkovnici:

- Recimo da smo stisnuli tipku `a`, nakon stiskanja tipke događa se sljedeće:
  - Pročitan je "*Scan code*" stisnute tipke.
  - Potom je zaprimljen taj "*Scan code*" - u ovom slučaju `1e 9e`.
  - Ovaj kôd se prema tablici *Key codea* i pripadajućeg seta karaktera (pr. ***ASCII***) preslikava kao `30`.

## Koji su još elementi uključeni u ovu priču

Ovdje postoji još jedna komponenta, a to je **konzola** i/ili program u kojem radimo. Konzola je obično naš zaslon odnosno ekran.

Osim klasične konzole, pristup na konzolu može biti i preko t**erminal** programa za udaljeni pristup, poput:

- ***Telnet*** programa odnosno terminala za udaljeni pristup.
- ***Ssh*** terminala za udaljeni pristup.

Svaka konzola kao i terminal moraju imati i odabrani skup znakova odnosno alfabet (engl. *Character set)* poput:

- *ASCII skupa znakova* — pogledajte ASCII tablicu na kraju knjige: **29.1. ASCII tablica.**
- *UTF-8* ili nekog drugog.

Dakle sada se primjerice *Key code* `30` preslikava (ako koristimo ASCII) u ASCII kôd `97`. Ovaj korak preslikavanja se zove (***en)kodiranje*** (engl. *Encode*). Zatim sve ide dalje prema programu u kojem radimo (tipkamo) te se iscrtavaju stisnute tipke odnosno slova (ili što smo već zapravo stisnuli na tipkovnici) i to korištenjem *Fonta* koji je odabran.

**Dio priče s fontovima**
**Font** definira stil i oblik svih pojedinih slovnih znakovnika (engl. *Glyph*) i slovnih simbola, odnosno svakog tipografskog karaktera. Ovisno o *fontu* i njegovoj veličini, slova se iscrtavaju na vizualno drugačiji način.
Pogledajmo isti tekst ispisan s različitim fontovima iste veličine (konkretno veličine 10):

| | | | |
|---|---|---|---|
| Font: Times New Roman | Font: Century | Font:Courier New | **Font:Arial Black** |
| Font: Arial Narrow | Font:Book Antiqua | Font:DejaVu Sans | Font:Noto Serif |
| Font: Noto Serif | FONT: ALGERIAN **Font: Bauhaus 93** | FONT: CASTELLAR | |

**Izvori informacija; (72),(73),(180),(181),** `man localectl`, `man dumpkeys`, `man 5 locale`, `man 7 locale`, `man 5 locale.conf`.

## 10.3.2. Kodiranje, dekodiranje i kodne stranice

***Slijedi napredno poglavlje (10.3.2.x)!***

### 10.3.2.1. Kodiranje i dekodiranje

Sada ćemo se malo vratiti unazad kako bismo uopće razumjeli sljedeće poglavlje. Moramo biti svjesni činjenice kako računalo ne može pohraniti slova, brojeve, slike ili bilo što slično. Jedino što računalo može pohraniti i s čime može raditi su bȋtovi, odnosno nule (0) i jedinice (1). Svaki bȋt može biti u samo dva stanja odnosno može imati samo dvije vrijednosti: da ili ne, istina ili laž, 1 ili 0 ili bilo što drugo čime želite nazvati te dvije vrijednosti. Budući da računalo radi s električnim signalima, stvarni bȋt je naponski skok (1) ili njegov nedostatak (0). U svakom slučaju to se obično prikazuje pomoću: `1` i `0`, čega ćemo se i pridržavati dalje u tekstu. Kako bismo upotrijebili bȋtove za predstavljanje bilo čega osim bȋtova, potrebna su nam određena pravila. Prema standardu je dogovoreno da će kratki nȋzovi bȋtova (nula i jedinica) točno definirane duljine predstavljati slova, brojeve i posebne znakove. Ovakva reprezentacija točnoj određenog nȋza bȋtova koji predstavljaju slova, brojeve i posebne znakove se zove ***(en)kodiranje*** ili ponekad *kodiranje*.
Pogledajmo kako to izgleda za nekoliko slova (u prvom redu je binarna reprezentacija, a u drugome ono što ona predstavlja):

```
01100001 00100000 01100010 00100000 01100011
   a        b        c        d        e
```

*I prije pojave prvih računala, postojala je potreba za kodiranjem slova i brojeva u signale koje je bilo lako prenijeti na odredište, što se primjerice koristilo za brzojave (telegrafe). Ovdje se koristio **Morseov kod**, koji je kodirao slova i brojeve u duge ili kratke impulse odnosno signale tj. točke i crtice. Primjerice slovo **A** se kodiralo kao: .- a slovo **B** kao: -... i tako dalje.*

Prema jednoj od osnovnih tablica ***(en)kodiranja***, koja se zove *ASCII* (engl. *American Standard Code for Information Interchange*), svako slovo, broj ili poseban znak definiran je kao nȋz od osam (8) bȋtova. Tako je malo slovo **`a`** definirano kao binarni nȋz **`01100001`** odnosno kako bismo bolje razumjeli, dekadski je to vrijednost **`97`**. Potom je malo slovo **`b`** definirano kao binarni nȋz **`00100000`** i tako dalje.
Podsjetimo se binarnog sustava za 8 bitnu vrijednost, na primjeru za malo slovo **`a`**, koje je kodirano kao `01100001`:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | **Potencije broja 2** |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | **Dekadska vrijednost** |
| | | | | | | | | |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | **Binarna reprezentacija** |

Pojednostavljeno, ako želimo pretvoriti binarni broj u dekadski, zbrajaju se vrijednosti sȃmo tamo gdje imamo vrijednost `1`, pa je to: 64 + 32 + 1 = 97 dekadski.

Vratimo se na ***kodiranje***. Dakle *ASCII* tablica, skraćena na sȃmo par slova, bi izgledala ovako:

| Binarno | Znak/slovo/broj | Dekadski |
|---|---|---|
| . . . | . . . | . . . |
| 01100001 | **a** | 97 |
| 00100000 | **b** | 98 |
| 01100010 | **c** | 99 |
| 00100000 | **d** | 100 |
| 01100011 | **e** | 101 |
| . . . | . . . | . . . |

U samoj *ASCII* tablici postoji definirano *kodiranje* za 95 slova i brojeva koja uključuju slova od A do Z i to posebno velika, a posebno mala slova, brojeve od 0 do 9, kao i posebne znakove poput: . , ; : / () [] {} „ @ i druge, kao i 33 posebne funkcionalnosti, poput znaka razmaka, oznake za novi redak i slično.
Prema navedenom *ASCII* standardu, definirano je ukupno 127 kodova (u 7 bitova) za slova, brojeve i posebne znakove.



Prema definiciji: ***(en)kodiranje*** je pretvorba (konvertiranje) u kȏdirani oblik.

Iako se na računalu svi podaci pohranjuju u binarnom obliku, ovakav način nije ljudima jednostavan za čitanje, pa se automatski konvertira odnosno dekodira u nama prihvatljiviji oblik, odnosno: slova, brojeve i posebne znakove.
Osim toga, mnogi programi nude nam prikaz, ali i upotrebu u oktalnom ili heksadecimalnom obliku pa bi tablica pretvorbe za navedenih nekoliko slova izgledala ovako:

| Binarno | Znak/slovo/broj | Dekadski | Oktalno | Heksadecimalno |
|---|---|---|---|---|
| . . . | . . . | . . . | . . . | . . . |
| 01100001 | **a** | 97 | 141 | 61 |
| 00100000 | **b** | 98 | 142 | 62 |
| 01100010 | **c** | 99 | 143 | 63 |
| 00100000 | **d** | 100 | 144 | 64 |
| 01100011 | **e** | 101 | 145 | 65 |
| . . . | . . . | . . . | . . . | . . . |



**Osim ASCII standarda za (en)kodiranje postoji i cijeli nîz drugih standarda, poput njegove preteče EBCDIC te novijih: UTF-8, UTF-16 i drugih.**

Pogledajmo i primjere. Kreirajmo datoteku imena: `tekst.txt` i u nju upišimo samo slova `abcde` na sljedeći način:

```
echo abcde > tekst.txt
```

Sada ćemo pomoću nekoliko programa pogledat kako izgleda ova datoteka.
Prvo ćemo probati s programom `cat`, koji će nam kao i gotovo svi drugi programi ispisati sadržaj datoteke, normalno dekodiran, pa ćemo dobiti upravo ono što smo i upisali.

Provjerimo što smo zapisali u ovu datoteku s programom `cat` (to smo mogli napraviti i s drugim programima):

```
cat tekst.txt
```

I dobit ćemo slijedeći ispis sadržaja ove datoteke:

```
abcde
```

Međutim, pošto nas zanima kako je to stvarno zapisano odnosno *(en)kodirano* i zapisano na disk u navedenu datoteku, možemo koristiti programe za tu posebnu namjenu. Neki od tih programa su:

- `hexdump` - za ispis (engl. *Dump*) izvornog sadržaja datoteke: binarno, oktalno, heksadecimalno ili dekadski.
- `xxd` - također za ispis, ali i kreiranje (zapisivanje) u datoteku: binarno, oktalno, heksadecimalno ili dekadski.
- `od` - za ispis izvornog sadržaja datoteke: binarno, oktalno, heksadecimalno ili dekadski, slično kao i naredba: `hexdump`.

Pomoću navedenih programa sadržaj bilo koje datoteke možemo prikazati u *kodiranom* obliku i to kako želimo: dekadski, oktalno, binarno ili heksadecimalno. Prvo ćemo ispisati sadržaj datoteke dekadski, pomoću programa `od`:

```
od -An -vtu1 tekst.txt
```

I dobit ćemo:

```
97  98  99 100 101  10
```

To znači kako smo dobili *kodirani* zapis slova: `abcde` koja su u dekadskom zapisu, kako slijedi: `97`, `98`, `99`, `100` i `101`.

Pogledajmo istu datoteku i s programom `hexdump`, ali u heksadecimalnom formatu:

```
hexdump -C tekst.txt

00000000  61 62 63 64 65 0a                                |abcde.|
00000006
```

U prvom stupcu je oznaka/broj retka teksta (`00000000`) dok su u drugom vidljive heksadecimalni brojevi koji predstavljaju slova koja smo upisali. Na kraju, nakon slova `e` koje je heksadecimalno `65`, vidimo heksadecimalno `0a`, što je u dekadskom ponovno `10`. U krajnjem desnom stupcu vidljivo kao: `|abcde.|` vidimo reprezentaciju onoga što je upisano u ovu datoteku, u nama (ljudima) razumljivom obliku, dakle dekodirano.

**Što je zapisano na kraju ove datoteke, odnosno što je dekadski broj `10` odnosno heksadecimalno `0a`?**

Svaki put kada pišemo neki tekst i kada na kraju stisnemo tipku *ENTER* kako bismo otišli u novi redak, odnosno kako bismo mogli početi pisati u novom retku, što se (en)kodiranje tiče, potrebno je zapisati i određeni posebni znak koji označava da se nalazimo na kraju retka, nakon kojega logično slijedi novi redak teksta. U slučaju Linuxa, posebna oznaka stavlja se na kraj retka teksta, i to je upravo ovaj posebni znak čija dekadska vrijednost je `10`. Dakle *kodirano* dekadski `10`, predstavlja posebni znak: **`LF`** (engl. *Line Feed*, odnosno *New Line*), prema *ASCII* shemi *kodiranja*. Ponekad se ovaj posebni znak naziva i krajem retka (engl. *End of Line*) odnosno **EOL**. Ovakvo označavanje kraja retka teksta s oznakom **LF** odnosno dekadski **`10`**, a heksadecimalno **`0a`**, u upotrebi je na svim Linux operativnim sustavima, kao i na većini Unix operativnih sustava, dok je na *Microsoft Windows* operativnim sustavima to malo drugačije.

Naime, kada u operativnom sustavu *Microsoft Windows* stisnemo tipku **ENTER**, kojom ćemo tijekom zapisivanja teksta u datoteku, označiti kako slijedi novi redak teksta, više neće kao u Linuxu biti zapisan sâmo posebni znak **LF**, već će uz njega biti zapisan još jedan znak: **CR** (*engl. Carriage Return*) odnosno u konačnici oba znaka zajedno, sljedećim redoslijedom: **CR LF**. To znači da se za sve što snimate u operativnom sustavu *Microsoft Windows*, na kraju svakog retka teksta, zapisuju dva posebna znaka: **CR LF**. Pri tome **CR** ima decimalnu vrijednost **`13`**, što bi u heksadecimalnom bilo vidljivo kao **`0d`**.

To opet znači kako bi svaki redak teksta, na kraju retka, ako ga gledamo heksadecimalno, u *Microsoft Windowsima* izgledao odnosno završavao ovako*:*

```
0d 0a
```

Zbog navedenog načina zapisivanja teksta, potreban je oprez, ako određene dokumente uvozimo iz Linuxa u *Windowse* i obratno. Naime vrlo je često da programi u kojim radimo ne prepoznaju takozvani ***Windows*** ili ***Linux*** format, pa je stoga potrebno napraviti konverziju, koju ponekada, a danas sve češće, možemo napraviti i iz sâmog programa u kojem radimo, ako već nije prepoznao format teksta ili dokumenta na kojem radimo.

## Dodatna razina (en)kodiranja

Kao što smo naučili: **ASCII** standard za kodiranje teksta koristi 128 jedinstvenih vrijednosti (0–127) za prikaz abecednih, numeričkih i interpunkcijskih znakova koji se obično koriste na engleskom jeziku, te nîz kontrolnih kodova koji ne predstavljaju znakove koji se mogu ispisati. Na primjer, veliko slovo **A** je ASCII znak 65, broj **1** je ASCII 49, znak **@** je ASCII 64, a znak za novi red (**CR**) je ASCII 13.

Možemo imati i neke binarne podatke ili podatke koji sadrže posebne karaktere odnosno kontrolne ili takozvane neprintabilne odnosno neispisive znakove poput: xA9EOTx86NULFFEOT* koje želite prenijeti preko mreže ili u nekim slučajevima pohraniti unutar neke specifične aplikacije. U svakom slučaju to obično ne radite tako da sâmo prebacujete nîzove znakova (slova/brojeva/posebnih znakova) preko mreže ili u neku vašu aplikaciju, u sirovom formatu (vidljivom gore*). Zbog toga što neki programi s druge strane potencijalno nisu u stanju pravilno zaprimiti poslane podatke jer su dizajnirani za primanje sâmo teksta bez posebnih znakova. Pošto nikad ne znate što se očekuje s druge strane ili u prijenosu podataka preko mreže jer i neki mrežni protokoli mogu protumačiti vaše (binarne) podatke kao kontrolne znakove, ili bi se vaši binarni podaci mogli poremetiti jer bi primjerice mrežni protokol za prijenos mogao misliti da ste unijeli kombinaciju posebnih znakova. Jedan od primjera ovakve komunikacije koja bi bila problematična je i slučaj upotrebe **FTP** protokola koji prevodi kontrolne znakove za kraj retka (**CR** i **CR/LF**) te bi na drugoj strani imali podatke, koji više ne bi odgovarali izvornim podacima.

Kako bismo zaobišli ove probleme, (en)kodirat ćemo (binarne) podatke u znakove koji se bez problema mogu prenositi jer ne sadrže nikakve posebne znakove, i to pomoću posebnog algoritma odnosno posebnom vrstom kôdiranja za tu namjenu.

***Base64*** je jedna od tih vrsta kôdiranja. ***Base64*** koristi samo 64 znaka koji su prisutna u mnogim skupovima znakova (pr. ASCII), pa možete biti prilično sigurni da će vaši podaci završiti na drugoj strani (mreže) nekorumpirani.

Naime izlazni rezultat *Base64* kodiranja koristi sâmo mala slova (a-z), velika slova (A-Z), brojeve (0-9) i znak (/).

Probajmo napravit *Base64* kôdiranje teksta `TeSt12!` pomoću programa `base64` na sljedeći način:

```
echo 'TeSt12!' | base64
VGVTdDEyIQo=
```

Kao rezultat smo dobili *Base64* kodirani nîz: `VGVTdDEyIQo=`.

Sada probajmo i obratno; dekodirati podatke iz *Base64* nîza koji smo dobili, u izvorni oblik, sa sljedećom naredbom:

```
echo 'VGVTdDEyIQo=' | base64 --decode
TeSt12!
```

I dobili smo nazad izvorni tekst. Dakle i za *Base64* kôdiranje i dekôdiranje koristili smo naredbu `base64`.



Osim ***Base64***, u upotrebi su i: ***Base16***, ***Base32***, ***Base36***, ***Base58***, ***Base85***, ***Base91***, ***Base122***, ***BinHex***, ***MIME***, ***Z85*** te mnogi drugi standardi. **Base64** se primjerice koristi za prijenos binarnih podataka (pr. datoteka ili umetaka) unutar elektroničke pošte.

**Izvori informacija: (300),(302),(303),(569), `man hexdump`, `man xxd`, `man od`, `man base64`, `man 7 unicode`.**

---

### 10.3.2.2. Kodne stranice

## Kako su nastale i što su kodne stranice

Prva osobna računala su razvijana za englesko govorno područje, pa je prema tome prvo bilo potrebno razviti određeni znakovnik slova i interpunkcijskih znakova, kao i brojeva i posebnih znakova za engleski jezik. Tako je otprilike i nastao *ASCII* znakovnik (skup/*set*), za koji je odlučeno kako je dovoljno adresirati samo 7 bitova, što omogućava definiranje ukupno 128 slova, brojeva i posebnih znakova. Pogledajte i ***ASCII tablicu*** na kraju knjige. Unutar ovog znakovnika odnosno tablice definiran je osnovni nîz slova, brojeva i znakova koji je bio zadovoljavajući u trenutku njegovog stvaranja. Kasnije se pojavila potreba za dodavanjem novih posebnih znakova, tako što se ovaj znakovnik proširio na 8 bitova, pa je sada bilo moguće koristiti do 256 znakova, odnosno slova, brojeva i posebnih znakova. Tako je nastao **prošireni ASCII** znakovnik.

Ovaj znakovnik je nadogradnja osnovnog *ASCII* znakovnika, tako da počinje od 128 pozicije jer osnovni *ASCII* znakovnik završava na poziciji 127, sve do 255-te, gdje se sada nalaze dodatni odnosno posebni znakovi. Kako su se vremenom osobna računala koristila i izvan engleskog govornog područja, pojavila se potreba za korištenje znakova koji su specifični za druge jezike. Ideja je bila koristiti osnovni *ASCII* znakovnik (pozicije 0-127) kao standardan, a proširiti ga sa znakovnikom specifičnim za pojedini jezik ili više njih: slično kao kod proširenog *ASCII* znakovnika.

Ovakva upotreba se naziva upotrebom takozvanih *kôdnih stranica* odnosno "***Code pages***" na engleskom. Jedna od prvih kôdnih stranica je bila kôdna stranica oznake: **CP437**, koja je sadržavala posebne znakove; grčka i još poneka slova, ali ne i naša (HR). Na osnovi nje je nastala i jedna od prvih kodnih stranica koja je uključivala naša slova tj. veliko i malo slovo: **šđčćž**. Dakle kôdna stranica oznake **8859-2**, koja je znana i kao ***Latin 2***. Ona je napravljena za sve jezike centralne i istočne Europe, s latiničnim pismom.

Osim kôdne stranice **8859-2** postoji i **8859-1** (znana i kao *Latin 1*), ali ona je za jezike zapadne Europe (Njemački, Francuski, …). Unutar nekih operativnih sustava napravljene su vlastite kôdne stranice poput **Windows 1250**, koja ima latinična slova, poput *Latin 2* (***8859-2***). Dakle za centralno i istočno europske jezike. Problem se javlja kada trebate pisati odnosno koristiti nešto za što su vam potrebne dvije ili više kodnih stranica jer unutar jedne od njih nemate sve znakove ili slova koja su vam potrebna. Ako recimo radite na nekom dokumentu, morate biti sigurni da će i primatelj vašeg dokumenta imati sve (iste) kôdne stranice koje ste i vi koristili. Kako ne bi bilo zabune postoji i cijeli nîz kôdnih stranica za druge jezike.

### *Unikod (unicode)*

*ASCII* znakovnik je prilično ograničen što se tiče posebnih znakova te se u novije vrijeme često koristi (en)kodiranje prema tzv. **unikod** standardu. ***Unikod*** je razvijen kako bi objedinio sve znakove iz svih trenutačno i povijesno korištenih ljudskih jezika u jednu veliku kodnu stranicu, uklanjajući potrebu za razlikovanjem različitih kodnih stranica pri rukovanju s digitalno pohranjenim tekstom. *Unikod* pokušava zadržati kompatibilnost unatrag s mnogim naslijeđenim kodnim stranicama, kopirajući neke kodne stranice 1:1 u procesu dizajna.

*Unicode Transformation Format* to jest **UTF** format je implementiran u: ***UTF-8***, ***UTF-16*** ili ***UTF-32*** odnosno to su implementacije ove ideje koje unutar sebe podržavaju kompletan nîz znakova iz *ASCII* znakovnika te ga znatno proširuju. On je dovoljno velik da u njega stanu i gotovo sve kôdne stranice koje postoje ili koje vam trebaju. Upravo je to i bila ideja razvoja *unikod* standarda implementiranog u ***UTF-8***: praktično upotrebe jednog znakovnika za sve jezike. *ASCII* znakovnik podržava do 128 karaktera odnosno slova, znakova, brojeva i posebnih znakova dok konkretno ***UTF-8*** podržava njih preko milijun, jer podržava pohranu i 32-bitnih vrijednosti, iako je inicijalno 8-bitan.
***UTF-8*** je dominantan standard za (en)kodiranje znakova na Webu (*World Wide Web*), te se prema statistikama za 2022. godinu koristi na oko 98% svih web stranica na svijetu, te čak na 100% stranica za određene (ljudske) jezike.



Pogledajte i poglavlje:
**6.2.2 Sistemske (Environment) varijable** i to primjer s terminalima i (en)kodiranjem odnosno kodnim stranicama.

Dodatno pogledajte i poglavlje:
**4.7. Format datoteka (*MIME type*).**

**Izvori informacija: (73),(74),**(182),(183),(184),(185),(186),(187),(188),(1139), `man 7 unicode`,`man 7 utf-8`.

## 10.3.3. Tipke s posebnom funkcionalnošću

***Slijedi napredno poglavlje (10.3.3.x)!***
U narednom naprednom djelu objasnit ćemo nekoliko tipki koje imaju posebne funkcionalnosti, a koje je potrebno razumjeti.

### 10.3.3.1. Tipka *Alt Gr*

`Alt Gr` odnosno *Alt Graph* ili *Graphic* daje nam mogućnost ispisa posebnih znakova odnosno *karaktera* na tipkovnici. Dodatno je moguć i ispis posebnih znakova prema *kôdu karaktera*, ako nam je poznata njena tablica odnosno *kôd* za karaktere, poput primjerice **ASCII** tablice. S jedne strane možemo dobiti znak odnosno karakter, kombinacijom tipke `Alt Gr` i bilo koje tipke na našoj tipkovnici, koja nam daje taj poseban znak. Primjerice stiskanjem `Alt Gr` te tipke odnosno slova `V` dobivamo znak `@`.
S druge strane, ako znamo točan kôd za željeni karakter, to isto možemo dobiti stiskanjem tipke `Alt Gr` i broja kôda.

Tako je prema **ASCII** tablici za naš znak `@` kodni broj `64`. Stisnimo `Alt Gr` i na numeričkoj tipkovnici (dok držimo stisnut `Alt Gr`) upišimo `64` te pustimo tipku `Alt Gr` i dobit ćemo istu stvar kao u primjeru gore.



Tipku `Alt Gr` moramo koristiti poput tipke `shift` - držimo ju stisnutu i istovremeno stišćemo drugu tipku.
Tipku `Alt Gr` često nazivamo i desni `Alt`, a na tipkovnicama koje ju nemaju, istu funkcionalnost dobivamo istovremenim stiskanjem tipki `Ctrl` i `Alt`..

Pogledajmo tablicu s osnovnim znakovima odnosno *karakterima*, najčešćim na Hrvatskim tipkovnicama.

| Karakter/znak koji ćemo dobiti | Kombinacija: `Alt Gr` i tipka: |
|---|---|
| \ | **Q** |
| \| | **W** |
| [ | **F** |
| ] | **G** |
| @ | **V** |
| { | **B** |
| } | **N** |
| § | **M** |
| ~ | **1** |
| ^ | **3** |
| ˘ | **4** |
| ` | **7** |
| ˙ | **8** |
| ´ | **9** |
| ˝ | **0** |

**Izvor informacija: (189)**

---

## 10.3.3.2. Tipke *SysRq* i *Print Scrn*

### `SysRq` funkcionalnost

Jedna od rijetko korištenih tipki, kao i njene funkcionalnosti, je tipka `SysRq`.
Na nekim tipkovnicama je označena samo kao tipka `Print Scrn`.



Izvorna namjena ove tipke je bila pokretanje više operativnih sustava na prvim PC AT računalima 80286, koja su radila na 6MHz. Iako je izvorna namjena `SysRq` odavno napuštena, neki operacijski sustavi i dalje ju koriste za svoje potrebe.

Pritiskom na tipku `SysRq` uzrokuje se izvršavanje posebnog signala prekida (engl. *Interrupt* odnosno skraćeno ***IRQ***).
Linux i dalje ima sposobnost slušanja ovog signala prekida ili pritiskanja ove tipke, ali u kombinaciji s drugim tipkama.
Ova kombinacija `SysRq` i drugih tipki se u Linux terminologiji zove ***Magic SysRq Key***. Preciznije: funkcionalnost ove tipke je implementirana unutar upravljačkog programa tipkovnice. To znači unutar trenutnog (radnog) linux kernela. Prema tome, dokle god radi Linux kernel, radi i ova funkcionalnost. Ova kombinacija tipki se često koristi u procesu oporavka uslijed neke velike greške u radu sustava ili u slučajevima kada je potrebno napraviti Tzv. **Kernel Dump/Crashdump** proceduru.



**Magic SysRq** funkcionalnost je povezana direktno s Linux kernelom te je stoga potencijalno opasna (osim ako ne znate što radite).



Za više detalja pogledajte poglavlje:
**16. Kernel Dump/Crashdump/core dump.**

Ovu kombinaciju tipki možemo svrstati i u kategoriju *Escape* sekvenci (nîzova).
Navedena kombinacija tipki se sastoji od istovremenog stiskanja tipki: `Alt` i `SysRq` i tipke koja označava željenu funkcionalnost. Tablica dolje predstavlja najčešće korištene funkcionalnosti *SysRq*a:

| Funkcionalnost | Tipka/Znak/Slovo |
|---|---|
| Razina logiranja (*Log Level*) za konzolu (poruke koje će se prikazivati na konzoli (ekranu)). | 0 1 2 3 4 5 6 7 8 9 |
| Nasilni i trenutni restart sustava (bez sinkronizacije datotečnog sustava iz RAM memorije na disk). | **b** |
| Pokreni gašenje sustava uz *crashdump* proceduru (ako je omogućena). | **c** |
| Pošalji signal SIGTERM svim procesima osim **init** procesu (on uredno gasi sve procese/programe). | **e** |
| Pošalji signal SIGKILL svim procesima osim **init** procesu (on nasilno gasi sve procese/programe). | **i** |
| Resetiraj „*nice*" prioritete za sve procese koji su u kategorijama *high-priority* i *Real-Time* . | **n** |
| Sinkroniziraj sve *montirane* datotečne sustave iz RAM memorije na disk (`sync`). | **s** |
| Napravi *demontiranje* svih *montiranih* datotečnih sustava kao *read-only* (bez prava zapisivanja, već samo čitanja). | **u** |

## Primjeri upotrebe

Najčešća upotreba ove funkcionalnosti se svodi na nekoliko koraka.
**1.** Potrebno je osigurati se da je ova funkcionalnost uključena. Privremeno ju uključimo sa sljedećom naredbom:

```
echo 1 > /proc/sys/kernel/sysrq
```

**2.** Slanje/pozivanje željene *SysRq* funkcionalnosti.

Zamislimo slučaj u kojem se naše računalo ili poslužitelj potpuno zaglavio te ga želimo restartati ovom metodom jer ništa drugo ne pomaže. Ova metoda radi i preko udaljenog pristupa (pr. *telnet* ili *ssh*).
Mi ćemo pozvati sljedeće funkcionalnosti, točno navedenim redoslijedom:

- 1. Uredno ugasimo sve procese/programe (`e`).
- 2. Pogasimo sve ostale programe koji su se zablokirali i ne reagiraju na prethodno uredno gašenje (`i`).
- 3. Napravimo sinkronizaciju podataka na disk (`s`), kako ne bi nešto izgubili te da datotečni sustav ne bi postao nekonzistentan.
- 4. Sve datotečne sustave koji su spojeni na sustav (*mountani*) odvojimo i spojimo ponovno, ali kao *read-only*. Kako se na njih ne bi moglo ništa više zapisivati odnosno kako ne bi došlo do ne konzistencije podataka (`u`).
- 5. Potom ćemo napraviti restart sustava (`b`).

Krenimo s implementacijom, pomoću slijedećih naredbi (naredbe 1-4) u nîzu, jedna za drugom:

```
echo e > /proc/sysrq-trigger
echo i > /proc/sysrq-trigger
echo s > /proc/sysrq-trigger
echo u > /proc/sysrq-trigger
```

I potom konačno slijedi i korak pet (restart):

```
echo b > /proc/sysrq-trigger
```

Čim smo napravili zadnji korak računalo će se bezuvjetno restartati.

**Izvor informacija: (190),(191)**

---

## `Print Scrn` funkcionalnost

Korištenjem `Print Scrn` funkcionalnosti odnosno sâmo stiskanjem ove tipke (na istoj tipki je i `SysRq`) kopiramo sadržaj našeg zaslona (Engl, *Screenshot*) u radnu memoriju. Stiskanjem tipke `Print Scrn` kopiramo naš cijeli grafički "ekran", sa svim prozorima i radnom površinom. Kopira se sve što vidimo na našem zaslonu monitora u grafičkom načinu rada koji se prema *Unix*/*Linux* terminologiji naziva ***X Window*** grafički sustav.
Kasnije je moguće snimiti/pohraniti ovu *sliku* doslovno kao sliku (*bitmapu*) u nekom od programa za grafičku obradu slika, kao što je primjerice program ***GIMP*** (***G****NU* ***I****mage* ***M****anipulation* ***P****rogram*). On je najpopularnija aplikacija otvorenog koda za stvaranje i obradu rasterske grafike. Koristi se za retuširanje i uređivanje odnosno obradu slika tj. digitalnih fotografija.

**Slika 48. Logo programa GIMP**



U slučaju kada želimo kopirati sâmo sadržaj jednog otvorenog prozora, potrebno je označiti taj prozor, kako bi postao aktivan, te istovremeno stisnuti kombinaciju tipki: `Alt Gr` i `Print Scrn`.
Nakon ovog procesa također možemo snimiti odnosno pohraniti *sliku* označenog prozora, kao takozvanu *bitmap sliku* odnosno kao rastersku sliku.

**Izvor informacija: (192)**

---

### 10.3.4. Rekonfiguracija tipkovnice

U slučaju kada imamo potrebu promijeniti postavke tipkovnice s recimo Američke (`us`) na Hrvatsku (`croat`) to je vrlo jednostavno. Rekli smo kako se definicije svih tipkovnica sa pripadajućim *Key Code*-ovima nalaze unutar direktorija `/lib/kbd/keymaps/i386` i to pozicionirane unutar pôddirektorija prema prvim tipkama na tipkovnici, nakon tipke *TAB*, pa tako imamo: *azerty*, *qwerty*, *qwertz* i druge tipkovnice. Definicija za Hrvatsku tipkovnicu zapisana je u datoteci `/lib/kbd/keymaps/i386/qwertz/croat.map.gz`

Za probu, učitajmo našu (hrvatsku) tipkovnicu pomoću naredbe `loadkeys`.
Hrvatska tipkovnica ima oznaku `croat` dok Američka/Engleska ima oznaku `us`

```
loadkeys croat
Loading /lib/kbd/keymaps/i386/qwertz/croat.map.gz
```

Sada možemo koristiti našu (HR) tipkovnicu.
Ovo uglavnom nije potrebno raditi jer se Linux pri instalaciji već pobrinuo kako bi se sve ispravno konfiguriralo.
Što se tiče trajne konfiguracije, za ***RedHat/CentOS*** 6.x. ručne promjene je potrebno napraviti u datoteci: `/etc/sysconfig/keyboard`.

**Izvor informacija:** `man loadkeys`, `man 5 keymaps`.

---

# 10.4. Postavke vremenske zone i sata te regionalne postavke

Definiranje vremenske zone je vezano za usklađivanje sata za vremensku zonu u kojoj se nalazimo.
Ove opcije se konfiguriraju u toku instalacije sâmog operativnog sustava, ali ih je moguće mijenjati i naknadno.
Lokalne odnosno regionalne postavke vezane su za:

- Tipkovnicu i kôdne stranice.
- Postavke monetarne valute; primjerice: **KN**, **U$**, **€** i slično.
- Formata vremena (sata) i datuma, u obliku: godina/mjesec/dan ili godina/dan/mjesec ili nekog drugog.
- Razne programe i biblioteke te za njihov pravilan prikaz i formatiranje teksta.

U daljim poglavljima govorit ćemo o konfiguraciji vremena i vremenske zone, na klasičan Linux (*init*) način. Međutim, uvođenjem ***systemd*** servisa i programskih paketa, postoji i druga metoda konfiguracije (za ***RedHat/CentOS*** v.7.x/8.x).


Za konfiguriranje Linux sustava, po pitanju postavki vremena i vremenske zone pomoću ***systemd*** programa i servisa pogledajte poglavlje: **7.3.2.5. Drugi korisni programi i komponente unutar *systemd* paketa**.

---

## 10.4.1. O hardverskom i sistemskom satu

Na svakoj matičnoj ploči pronaći ćete i bateriju poput ove na slici 49. dolje. Navedena baterija napaja integrirani sat koji je ugrađen u matičnu ploču. Praktično se radi o običnom kvarcnom oscilatoru odnosno satu; vrlo sličnom kao na ručnim ili zidnim satovima koje svakodnevno koristimo. Ovaj sat na matičnoj ploči se zove "*Real Time Clock*" ili **RTC**, a on radi bez prestanka, bez obzira je li računalo uključeno ili nije. Stoga mu je i potrebna baterija za napajanje i rad. Na starijim računalima, postojao je poseban integrirani sklop (*chip)* unutar kojeg je bio upravo ovakav sat, te *SRAM* vrsta memorije u kojoj su se čuvale postavke BIOS-a. Sva današnja računala sve postavke BIOS-a spremaju u neku vrstu *Flash* memorije koja se nalazi na matičnoj ploči te im više nije potrebna ovakva *SRAM* memorija koja zahtjeva bateriju za održavanje sadržaja odnosno stanja memorije. Stoga je danas jedini razlog postojanja ove baterije, napajanje integriranog sata.

**Slika 49. CR2032 baterija na matičnoj ploči**




Na novijim računalima *RTC* je integriran u "*Southbridge*" integrirani sklop (engl. *Chip*).
*RTC* odnosno hardverski sat, pohranjuje i prikazuje datum: godinu, mjesec, dan i vrijeme: sat, minutu i sekundu. On nije u mogućnosti koristiti našu vremensku zonu ili promjene vezane za ljetno ili zimsko računanje vremena.

Kada se operativni sustav pokreće, učitava se točan datum i vrijeme iz hardverskog (*RTC*) sata. Nakon toga se postavlja **sistemski sat** operativnog sustava, koji više ne ovisi o hardverskom satu. *Unix* odnosno linux vrijeme se od tog trenutka računa kao broj sekundi od ponoći prvog siječnja 1970. godine jer je to službeni datum rođenja *Unix*-a.


Ovo vrijeme koje počinje 1.1 1970.g u: 00:00:00 se naziva i ***UNIX epoha***.

Poslije postavljanja i rada sistemskog odnosno UNIX/Linux sata, u bilo kojem trenutku može se raditi konverzija u kalendarski sat, koji i prikazuje većina programa zbog olakšavanja rada, nama (ljudima). Sistemski sat nadalje više uopće ne ovisi o hardverskom satu, već se za točno računanje vremena pouzdaje u neke od generatora vremena to jest takta (mjerača vremena).
Najčešće korišteni izvori takta (vremena) odnosno mjerači vremena na ***x86*** kompatibilnim računalima su:

**i8254 – PIT (programmable interrupt timer [`pit`])**
Jedan od prvih dostupnih mjerača vremena je programibilni mjerač vremena signala prekida odnosno tzv. **PIT (***programmable interrupt timer***)**. *PIT* ima fiksnu frekvenciju odnosno osnovni takt od 1,193182 MHz te tri kanala koji se mogu programirati za isporuku periodičnih ili jednokratnih vremenskih signala.

**RTC (realtime clock)**
Drugi uređaj koji je bio dostupan u izvornim PC kompatibilnim računalima bio je **MC146818** tj, već navedeni **RTC** (*real time clock*). Ovakav izvorni uređaj je sada zastario i obično ga oponaša sistemski čipset, ponekad pomoću takozvanog ***HPET***-a ili nekog od mehanizama usmjeravanja signala prekida (IRQ). RTC-u se pristupa preko CMOS varijabli, a on generira signale prekida koji se obično usmjeravaju na IRQ 8. RTC ažurira trenutno vrijeme (sat) pomoću baterije (slika 49. ) čak i dok je sustav (računalo) isključen. Njegov interni sat koristi kristalni oscilator koji radi na frekvenciji od 32,768 kHz.
Linux do RTC sata dolazi pomoću posebne datoteke `/dev/rtc0` a očitanje RTC sata je moguće dobiti na sljedeći način:

```
cat /proc/driver/rtc
```

**APIC (Advanced Programmable Interrupt Controller)**
Od *Pentium* generacije procesora i svim novijima, ugrađeni mjerač vremena dostupan je za svaki CPU kao dio naprednog programibilnog kontrolera signala prekida zvanog APIC (*Advanced Programmable Interrupt Controller*).
APIC-u se pristupa preko memorijsko mapiranih registara. Iako je u teoriji APIC siguran i stabilan izvor signala (kao mjerač vremena), u praksi su se pojavile mnoge greške zbog kojih se on ne smatra toliko pouzdanim izvorom.


Za **APIC** (i PIC), pogledajte poglavlje:
**10.5.1. IRQ (Interrupt request).**

**HPET (High Precision Event Timer [`hpet`]), TSC (Time Stamp Counter [`tsc`])**
**i ACPI Power Management Timer [`acpi_pm`]**
**HPET** (*High Precision Event Timer*) je prilično složen, a izvorno je trebao zamijeniti **PIT** i **RTC** za x86 kompatibilna računala. On je u usporedbi sa starijim vrstama mjerača vremena dostupnih u x86 arhitekturi računala najtočniji i najsigurniji. Stoga se HPET koristi i kod aplikacija koje su vrlo osjetljive na vrijeme.

Drugi mehanizam često u primjerni je takozvani bojač vremenskih oznaka **TSC** (*Time Stamp Counter*) a on je dostupan preko registra prisutnog na svim x86 procesorima od *Pentiuma*. Njegov zadatak je brojanje CPU ciklusa procesora. **TSC** je nekoć bio izvrstan način za dobivanje podataka o vremenu (ciklusima/taktu) CPU-a. Međutim s pojavom višejezgrenih modernih procesora (SMP i NUMA), odnosno sustava s više CPU jezgri i operativnih sustava koji u nekom trenutku mogu biti u hibernaciji, ne može se više toliko pouzdati u TSC. Pogotovo ako u obzir uzmemo i mogućnost procesora da zbog uštede energije smanji svoj radni takt.

**ACPI Power Management Timer** je jednostavan brojač vremena čiji oscilator radi na frekvenciji (taktu) od 3,579545 MHz, a čiji rad ovisi o tome postoji li podrška za **ACPI** značajku na sustavu.
Osim toga, nekoliko kartica, čak i neke mrežne kartice (pr. Intel ***e1000***) imaju zasebne sklopove za mjerenje vremena ugrađene u same kartice koji mogu imati dostupne registre za njihovo iščitavanje.


Za problematiku očitavanja sata (točnog vremena**)** u virtualizaciji pogledajte poglavlje:
**27.1.3.7. Sistemski sat virtualnih računala.**

Naučili smo da sistemski sat od trenutka pokretanja računala više uopće ne ovisi o hardverskom satu i vremenu (i datumu) koje je pročitao iz njega, već se za točno računanje vremena pouzdaje u neke od navedenih generatora takta i to obično odabirom najpouzdanijeg od njih. Listu dostupnih generatora takta (vremena) možemo vidjeti sa sljedećom naredbom:

```
cat /sys/devices/system/clocksource/clocksource0/available_clocksource

tsc hpet acpi_pm
```

← U ovom konkretnom slučaju vidimo da na sustavu imamo dostupno njih nekolicinu: `tsc`, `hpet` i `acpi_pm`.

S druge strane odabrani izvor se definira (a može se i mijenjati) u varijabli to jest *posebnoj datoteci*. Pogledajmo njen sadržaj:

```
cat /sys/devices/system/clocksource/clocksource0/current_clocksource

tsc
```

← U ovom slučaju je odabran `tsc` (*Time Stamp Counter*) dakle on se koristi za cijeli sustav.

Naime iako Linux kernel može koristiti ***HPET*** kao izvor takta (koji smo u konkretnom slučaju imali dostupan), dokumentacija *Red Hat MRG* inačice 2 navodi da je **TSC** preferirani izvor takta zbog mnogo nižeg opterećenja sustava, ali sustav može koristi **HPET** kao zamjenu. Referentna vrijednost u tom okruženju za 10 milijuna brojanja pristupanja mjeraču vremena otkrila je da je ***TSC-u*** trebalo oko 0,6 sekundi, ***HPET-u*** nešto više od 12 sekundi, a ***ACPI Power Management Timeru*** oko 24 sekunde.


U 2019. godini odlučeno je staviti ***HPET*** na crnu listu u novijim kernelima Linuxa kada rade na nekim Intel CPU-ima (pr. *Coffee Lake* generacija procesora) zbog njegove nestabilnosti.

U slučaju da smo recimo ručno poželjeli sustav postaviti da ipak koristi ***HPET***, to bi mogli napraviti sa:

```
echo "hpet" > /sys/devices/system/clocksource/clocksource0/current_clocksource
```

Inicijalno se odabir izvora takta (vremena) odrađuje u trenutku inicijalizacije sustava odnosno u trenutku pokretanja kernela. Pogledajmo kako je to izgledalo na našem sustavu:

```
dmesg | grep -i clocksource
clocksource: refined-jiffies: mask:0xffffffff max_cycles:0xffffffff, max_idle_ns:
7645519600211568 ns
clocksource: hpet: mask:0xffffffff max_cycles: 0xffffffff,max_idle_ns:133484882848 ns
clocksource: tsc-early: mask:0xffffffffffffffff max_cycles:0x1fb66bb7c2f,
max_idle_ns: 440795306276 ns
clocksource:jiffies:mask:0xffffffff max_cycles:0xffffffff, max_idle_ns:
7645041785100000 ns
clocksource: Switched to clocksource tsc-early
clocksource: acpi_pm: mask: 0xffffff max_cycles: 0xffffff, max_idle_ns: 2085701024 ns
tsc: Refined TSC clocksource calibration: 2200.000 MHz
clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x1fb633008a4, max_idle_ns:
440795292230 ns
clocksource: Switched to clocksource tsc
```

← ova zadnja poruka (`Switched to clocksource tsc`) nam govori kako je ipak odabran ***TSC***.

Nadalje, moguće je definirati odabrani izvor takta (sata) i tijekom inicijalizacije *kernela* u **GRUB** postavki `GRUB_CMDLINE_LINUX` unutar varijable (`clocksource=`**`XY`**) [***XY***= naziv izvora: `hpet`, `tsc`, `acpi_pm`, `pit`, … ].

## Sistemski i hardverski sat

Na kraju, nakon što UNIX/Linux sustav krene s korištenjem svog internog softverskog sata koji se naziva ***sistemski sat***, svako malo se provjerava odmak odnosno kašnjenje između hardverskog (*RTC*) i sistemskog sata. Naime RTC i sistemski sat odmiču se različitim brzinama, pa će se postupno udaljavati jedan od drugoga, a također i od "pravog" vremena. Najjednostavniji način da ih se zadrži u donekle točnom vremenu jest mjeriti njihove stope pomicanja (odmaka međusobnog vremena) te to vrijeme primijeniti kao faktor korekcije. Ova razlika u vremenu se konstantno ažurira u datoteku: `/etc/adjtime`.

Pogledajmo hardverski (***RTC***) sat odnosno datum i vrijeme koji on pokazuje, s naredbom `hwclock`:

```
hwclock

Tue 11 Oct 2016 02:56:16 PM CEST  -0.969103 seconds
```

Vidljivo je kako smo pročitali datum i vrijeme iz hardverskog sata, kao i odmak odnosno kašnjenje između hardverskog i sistemskog sata; konkretno: `-0.969103` sekundi. Ova razlika odnosno kašnjenje se zove i "*Time drift value*".
Sistemski sat je u pravilu znatno točniji od hardverskog sata, ali i on ima odstupanja odnosno nije najprecizniji.
Ako pak želimo uskladiti hardverski sat (*RTC*) sa sistemskim satom odnosno popraviti hardversko vrijeme za ono vrijeme kašnjenja u odnosu na sistemski sat, to možemo uraditi na sljedeći način:

```
hwclock --adjust
```

Nakon nekog vremena, pogledajmo hardverski sat; kašnjenje će biti smanjeno zbog prijašnjeg usklađivanja satova:

```
hwclock

Tue 11 Oct 2016 03:08:32 PM CEST  -0.140972 seconds
```

Vidimo kako je kašnjenje smanjeno sa početnih: `-0.969103` na `-0.140972` sekundi, što je dosta velika razlika.



***RTC*** sat se naziva i *Hardverski* sat ili *CMOS* odnosno *BIOS* sat.

Česti prekidači naredbe `hwclock` za sinkronizaciju vremena između hardverskog (*RTC*) i sistemskog sata su:

- `--hctosys` - za namještanje sistemskog vremena (sata) prema hardverskom (***RTC***) vremenu: ***RTC → Sistem***
- `--systohc` - za namještanje hardverskog (***RTC***) vremena prema sistemskom vremenu (satu): ***Sistem → RTC***

Moguće je i postaviti točno definiran datum i vrijeme na hardverski (***RTC***) sat. Pri tome je sintaksa sljedeća:

```
hwclock --set --date „dd mm yyyy HH:MM"
```

Pri tome:

- `dd` - označava dan [primjerice: `24`], dok `mm` označava mjesec [primjerice: `07`].
- `yyyy` - označava godinu [primjerice: `2021`].
- `HH:MM` - označavaju sate (*HH*) i minute (*MM*) [primjerice: `21:47`].

Sljedeći korak u točnosti je korištenje NTP servisa za periodičko čitanje vremena iz mrežnog izvora točnog vremena iz udaljenih NTP poslužitelja ili tzv. radio sata koji su u konačnici spojeni na neki atomski sat. A potom kontinuirano podešavati brzinu sistemskog sata tako da se vremena uvijek podudaraju, bez izazivanja naglih "skokova" u vremenu sustava (Linuxa).
O NTP protokolu i servisu govorit ćemo u sljedećem poglavlju.

**Izvor informacija: (75),(76),(1484),(1485),(1486),(1488),(1489),(1490),(1491),(1492), `man hwclock`,`man adjtime`**

### 10.4.2. NTP

Za još točnije odnosno preciznije vrijeme se koristi ***NTP*** protokol (engl. *Network Time Protokol*), pomoću kojeg je moguće u redovitim vremenskim intervalima sinkronizirati naš sistemski sat s udaljenim atomskim satom koji podržava ovaj protokol. Dakle *NTP* je mrežni protokol za sinkronizaciju vremena (sata) preko mreže.

On je jedan od najstarijih mrežnih protokola; u upotrebi je od 1985. godine. Sinkronizacijom vremena (sata) s *NTP* poslužiteljem dobivamo preciznost od nekoliko milisekundi u odnosu na *NTP* sat poslužitelja. U slučaju sinkronizacije s *NTP* poslužiteljima na internetu, obično je preciznost od desetak milisekundi uobičajena. *NTP* za komunikaciju koristi *UDP* protokol i to port ***123***, a vrsta komunikacije može biti *Broadcast* ili *Unicast*. Protokol radi na principu: klijent → poslužitelj. Dakle *NTP* klijent (vaše računalo) se spaja na *NTP* poslužitelj (*NTP Server*). Osim ovog načina rada moguć je i Tzv. "*peer-to-peer*" način povezivanja u kojemu se svaki "*peer*" odnosno *NTP* klijent ponaša i kao *NTP server* te jedan drugome mogu biti vršni poslužitelji, kao referentni *NTP* izvori.

Trenutna inačica *NTP* protokola je v.4. i definirana je u standardu **RFC 5905**. *NTP* protokol, odnosno njegov poslužiteljski dio radi na hijerarhijskom, višeslojnom principu. Na vrhu hijerarhijske strukture (**Stratum 0**) se nalazi *Atomski* sat. Ispod njega, odnosno direktno spojeni na njega su *NTP* poslužitelji koji su u hijerarhiji vidljivi kao **Stratum 1**.

Sljedeći ispod njih su *NTP* poslužitelji (spojeni na **Stratum 1** NTP poslužitelje) koji pripadaju hijerarhiji **Stratum 2** i tako prema dolje. Možemo reći kako je jedinica udaljenosti od *Atomskog* sata upravo *Stratum* broj, a što je on veći, to je poslužitelj udaljeniji od njega:

- **Stratum 1 *NTP*** poslužitelji, pošto su spojeni na referentni sat (*Atomski* u konačnici) imaju očekivano odstupanje vremena od samo nekoliko mikro sekundi (µs).
- **Stratum 2 *NTP*** poslužitelji imaju nešto veće kašnjenje, ali se oni stoga mogu spajati (sinkronizirati) na više **Stratum 1** poslužitelja istovremeno, kao i na više **Stratum 2** poslužitelja kako bismo dobili točnije vrijeme.
- **Stratum 3 *NTP*** poslužitelji imaju još malo veće kašnjenje, ali se oni isto mogu spajati (sinkronizirati) na više **Stratum 2** poslužitelja istovremeno, kao i na više **Stratum 3** poslužitelja.
- **Stratum 4 *NTP*** poslužitelji imaju još nešto veće kašnjenje, ali se i oni stoga mogu spajati (sinkronizirati) na više **Stratum 3** poslužitelja istovremeno, kao i na više **Stratum 4** poslužitelja i tako sve do **Stratum 15** poslužitelja koji je zadnji mogući element u hijerarhiji.



**Stratum 16** se koristi kao poruka kada je nemoguća sinkronizacija preko ***NTP*** poslužitelja.

*NTP* klijenti se mogu spajati na bilo koji od *NTP* poslužitelja u hijerarhiji, ali je uvijek cilj spajati se na one *NTP* poslužitelje što bliže *Atomskom* satu odnosno izvoru točnog vremena.



**Atomski sat se smatra najpreciznijim satom koji postoji. Najnovija generacija atomskih satova ima najveću grešku od 1 sekunde u 30 milijuna (30.000.000) godina.**

U praksi se često koristi sinkronizacija sistemskog sata preko *NTP* protokola. U Hrvatskoj imamo nekoliko javno dostupnih *NTP* poslužitelja (nužno je odabrati zemljopisno što bliži), a to su primjerice:

- Za Osijek: *os.ntp.carnet.hr* (**Stratum 2**).
- Za Split: *st.ntp.carnet.hr* (**Stratum 2**).
- Za Zagreb: *zg1.ntp.carnet.hr* (**Stratum 2**) i *zg2.ntp.carnet.hr* (**Stratum 2**).



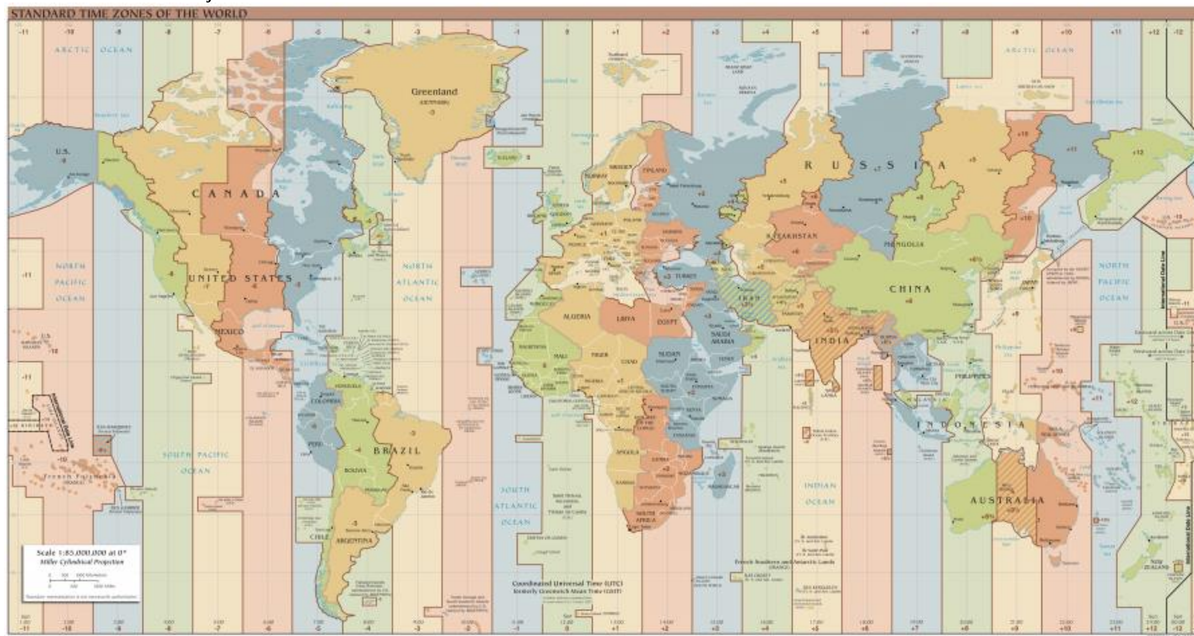
Za više detalja o ***NTP*** protokolu, pogledajte poglavlje**:**
**26.6. NTP.**

**Izvori informacija: (440), (K-12),(K-14), RFC 5905**

### 10.4.3. Vremenska zona

U slučajevima kada smo tijekom instalacije operativnog sustava konfigurirali krivu vremensku zonu ili ju je potrebno promijeniti zbog nekog drugog razloga, to je prilično jednostavno. Ali prvo kratko o vremenskim zonama. **Citat s Wikipedije**:
” *Jedna vremenska zona pokriva područja Zemlje koja su izabrala isto lokalno vrijeme.*
*Granice vremenskih zona u pravilu leže na meridijanima zemljopisne dužine koji su višekratnici 15°, pa razlika između susjednih vremenskih zona iznosi jedan sat. Ipak, ponegdje je razlika i drukčija. Osim toga, kao što se može vidjeti na karti u nastavku, vremenske zone mogu imati prilično nepravilne oblike jer obično prate granice država ili drugih administrativnih područja.* “

***Slika 50. Vremenske zone u svijetu.***




Izvor slike: https://upload.wikimedia.org/wikipedia/commons/thumb/e/e8/Standard_World_Time_Zones.png/1280px-Standard_World_Time_Zones.png

” *GMT je sustav za predstavljanje vremena koji je bio prihvaćen kao svjetsko vrijeme. GMT je engl. kratica za Greenwich Mean Time, a ime je dobilo po nultom meridijanu, koji od 1884. godine prolazi kroz zvjezdarnicu u Greenwichu kraj Londona.*
*Danas ta kratica polako izlazi iz uporabe (osim u pomorstvu, gdje je zadržala svoje tradicionalno ime), zamjenjuje ju UTC što znači Koordinirano svjetsko vrijeme. Koordinirano svjetsko vrijeme (eng. Coordinated Universal Time, UTC) je međunarodni naziv koji je zamijenio GMT, a predstavlja vrijeme nulte, odnosno početne vremenske zone. To je standardno vrijeme prema kojem se određuje vrijeme u drugim dijelovima svijeta. Hrvatska se nalazi u srednjoeuropskoj vremenskoj zoni, UTC+1, što znači da je u Hrvatskoj tijekom zime 13 sati kada je UTC vrijeme 12 sati. Ljeti je u Hrvatskoj propisano pomicanje satova pa je stoga ljeti u Hrvatskoj 14 sati kada je UTC vrijeme 12 sati.* “

Na Linux sustavima moguće je koristiti i ***GMT*** i ***UTC*** za vremenske zone, mada se preporuča ***UTC*** kao noviji, sveobuhvatniji i precizniji standard.

**Vremenske Zone i UTC:**

- **UTC** (*Coordinated Universal Time*) – osnovni vremenski standard koji ne ovisi od vremenskih zona. Služi kao referenca za sve druge vremenske zone.
- **GMT** (*Greenwich Mean Time*) - često se koristi kao sinonim za **UTC**, iako postoje male tehničke razlike.

**Ljetno računanje vremena**
Mnoge zemlje koriste ljetno računanje vremena (**DST** – *Daylight Saving Time*), kod kojeg se satovi pomjeraju unaprijed za jedan sat tijekom ljetnih mjeseci kako bi se bolje iskoristila dnevna svjetlost.

**Primjeri nekih od vremenskih zona:**

- **CET** (*Central European Time*): UTC+1, koristi se u većem dijelu Europe.
- **CEST** (*Central European Summer Time*) je vremenska zona koja se koristi kao ljetno računanje vremena za srednju Europu. **CEST** je dva sata ispred **UTC** (UTC+2). Ova vremenska zona se koristi tijekom ljetnih mjeseci, obično od kraja ožujka do kraja listopada.
- **CST** (*Central Standard Time*): UTC-6, koristi se u centralnom dijelu SAD-a (**Teksas**, **Illinois, …**).
- **EET** (*Eastern European Time*): UTC+2, koristi se u istočnoj Europi (**Bugarska**, **Cipar**, **Grčka**, **Rumunjska, …**).
- **EEST** (*Eastern European Summer Time*): UTC+3 – za ljetno računanje vremena.
- **EST** (*Eastern Standard Time*): UTC-5, koristi se na istočnoj obali SAD-a.
- **IST** (*India Standard Time*): UTC+5:30, koristi se u Indiji.
- **PST** (*Pacific Standard Time*): UTC-8, koristi se na zapadnoj obali SAD-a (**Kalifornija**, **Oregon**, **Washington, …**).

**Izvori informacija: (78), (1113)**

### 10.4.3.1. Rekonfiguracija vremenske zone

Za provjeru trenutnog vremena možemo pokrenuti naredbu `date`, pa pogledajmo i kako:

```
date

Thu Nov  3 16:20:29 CET 2016
```

Dakle dobili smo trenutno vrijeme. Osim toga vidljiva je i vremenska zona `CET` koja znači "*Central Europe Time*" odnosno centralno Europsko vrijeme. Konfiguracija vremenske zone, a koja utječe na ovaj prikaz vremena nalazi se definirana u datoteci: `/etc/localtime`. Malo točnije, ova datoteka bi trebala pokazivati na datoteku u kojoj je definirana vremenska zona. Još preciznije rečeno, ova datoteka je simbolički link na datoteku u kojoj je definirana točna vremenska zona koju koristimo.

Pogledajmo i kako to izgleda za našu vremensku zonu, koja je postavljena kao: **UTC +1** (*Zagreb*):

```
ls -l /etc/localtime

lrwxrwxrwx 1 root root 33 Aug 13  2013 /etc/localtime -> /usr/share/zoneinfo/Europe/Zagreb
```

Ovdje je vidljivo kako je datoteka: `/etc/localtime` zapravo simbolički link (poveznica) na datoteku: `/usr/share/zoneinfo/Europe/Zagreb`.
Naime unutar direktorija `/usr/share/zoneinfo/` se nalazi stablo pôddirektorija, za sve vremenske zone.

Pogledajmo kako to stablo datoteka i direktorija (mapa) izgleda:

```
ls  /usr/share/zoneinfo/

Africa     Brazil  EST     GB-Eire   Hongkong Kwajalein NZ-CHAT  ROC        Universal
America    CET     EST5EDT GMT       Iceland  Libya     Navajo   ROK        W-SU
Antarctica CST6CDT Egypt   GMT+0     Indian   MET       PRC      Singapore  WET
Arctic     Canada  Eire    GMT-0     Iran     MST       PST8PDT  Turkey     Zulu
Asia       Chile   Etc     GMT0      Israel   MST7MDT   Pacific  UCT        iso3166
Atlantic   Cuba    Europe  Greenwich Jamaica  Mexico    Poland   US         posix
Australia  EET     GB      HST       Japan    NZ        Portugal UTC        posixrules right
```

U našem slučaju, za Europu, moguć je slijedeći izbor vremenskih zona:

```
ls  /usr/share/zoneinfo/Europe/

Amsterdam Bratislava Dublin      Kaliningrad Madrid    Oslo      San_Marino Tirane    Vilnius
Andorra   Brussels   Gibraltar   Kiev        Malta     Paris     Sarajevo   Tiraspol  Volgograd
Astrakhan Bucharest  Guernsey    Kirov       Mariehamn Podgorica Simferopol Ulyanovsk Warsaw
Athens    Budapest   Helsinki    Lisbon      Minsk     Prague    Skopje     Uzhgorod  Zagreb
Belfast   Busingen   Isle_of_Man Ljubljana   Monaco    Riga      Sofia      Vaduz     Zaporozhye
Belgrade  Chisinau   Istanbul    London      Moscow    Rome      Stockholm  Vatican   Zurich
Berlin    Copenhagen Jersey      Luxembourg  Nicosia   Samara    Tallinn    Vienna
```

I u konačnici za promjenu vremenske zone potrebno je promijeniti datoteku `/etc/localtime` koja mora biti simbolički link na datoteku koja odgovara našoj vremenskoj zoni. U našem slučaju na: `/usr/share/zoneinfo/Europe/Zagreb`.
To možemo postići s naredbom `ln` koristeći prekidače `-sf` koji znače:

- `-s` - kreirat ćemo simbolički link odnosno poveznicu (engl. *Soft Link*).
- `-f` - odredišna datoteka (ako postoji) će biti pregažena, ili promijenjena u našem slučaju (engl. *Force*).

Za ove promjene potrebno je biti *root* korisnik (*Administrator*). Dakle postavimo vremensku zonu na Hrvatsku (*Zagreb*):

```
ln -sf /usr/share/zoneinfo/Europe/Zagreb /etc/localtime
```

Nakon ove promjene moramo vidjeti simbolički link koji pokazuje na našu novu vremensku zonu:

```
ls -l /etc/localtime

lrwxrwxrwx 1 root root 33 Aug 13  2013 /etc/localtime -> /usr/share/zoneinfo/Europe/Zagreb
```

U slučajevima kada nije definirana datoteka: `/etc/localtime`, provjerava se dostupnost sistemske varijable `TZ` u kojoj se može definirati vremenska zona, ali se ovo ne preporuča u normalnim uvjetima.

Ako to ipak želimo, morali bi postaviti i eksportirati varijablu `TZ` na sljedeći način:

```
TZ='Europe/Zagreb'
export TZ
```

Kao pomoć kod kreiranja vrijednosti ove varijable, možemo koristiti naredbu `tzselect` koja nas vodi korak po korak do naše željene vremenske zone: prvo biramo kontinent pa državu, te na kraju dobivamo i vrijednost `TZ` varijable.

**Izvor informacija: (K-14), `man date`,`man 5 localtime`,`man ln`.**

## 10.4.4. Regionalne postavke odnosno *Locale*

***Slijedi napredno poglavlje (10.4.4.x)!***
Regionalnim postavkama odnosno "*engl. Locale*" definiramo korisničko okruženje koje ovisi o regionalnim i kulturološkim postavkama, kao i postavkama (en)kodiranja. Svaka kategorije ovih postavki je definirana zasebno te sâmim time kontrolira zaseban dio ovog podsustava. S naredbom: `locale` možemo pogledati sve dostupne opcije, koje se definiraju kao vrijednosti sljedećih sistemskih varijabli (navest ćemo samo one najčešće korištene):

- `LANG` je osnovna i najvažnija varijabla. U njoj se prema formatu: *JEZIK_TERITORIJ.KODNA STRANICA/ENKODIRANJE* definiraju postavke na sljedeći način:
  - *JEZIK* je definiran prema **ISO 639-1** kodovima za jezik.
  - *TERITORIJ* je definiran prema **ISO 3166-1**.
  - *KÔDNA STRANICA* odnosno (*EN)KÔDIRANJE* prema definiciji kôdne stranice ili standarda za kôdove (primjerice ***UTF-8***).

Za Hrvatski imamo nekoliko mogućnosti. Pošto je prema ***ISO639-1*** oznaka za Hrvatski `hr`, pogledajmo koje sve mogućnosti imamo. Koristit ćemo naredbu `locale -a` kako bismo vidjeli listu svih mogućih postavki, te ćemo filtrirati samo ključnu riječ `hr_` na sljedeći način:

```
locale -a | grep hr_

hr_HR.iso88592
hr_HR.utf8
```

Vidimo da je moguće koristiti *JEZIK*=`hr` te *TERITORIJ*=`HR` te:

- Kôdnu stranicu `ISO88592`.
- Ili `UTF8` enkodiranje.

Na poslužiteljima se najčešće zbog kompatibilnosti i drugih potencijalnih problema ostavljaju standardne postavke, koje su obično: `LANG=POSIX` ili `LANG=en_US.UTF-8`



Ako je varijabla `LANG` definirana, njene postavke će se uzeti kao standardne za postavljanje svih sljedećih varijabli koje ćemo opisati. Drugim riječima, ako smo definirali neku od donjih varijabli, one će pregaziti standardne (*default)* postavke koje postavljamo za sve varijable, u vršnoj varijabli `LANG`

Dio regionalnih postavki vezanih za vrijeme, definirane su unutar varijable: `LC_TIME`, a čije detaljne postavke možemo vidjeti sa slijedećom naredbom (gledamo ispis za *Američke* postavke), pri tome smo ispis skratili:

```
locale -k LC_TIME

abday="Sun;Mon;Tue;Wed;Thu;Fri;Sat"
day="Sunday;Monday;Tuesday;Wednesday;Thursday;Friday;Saturday"
abmon="Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec"
mon="January;February;March;April;May;June;July;August;September;October;November;Dec
ember"
am_pm="AM;PM"
d_t_fmt="%a %d %b %Y %r %Z"
week-ndays=7
week-1stweek=7
first_weekday=1
first_workday=2
timezone=""
date_fmt="%a %b %e %H:%M:%S %Z %Y"
time-codeset="UTF-8"
```

**Izvori informacija: (K-14), `man locale`, `man 5 locale`, `man locale.conf`, ISO 639-1, ISO 3166-1.**

### 10.4.4.1. POSIX standard

Već početkom 1980-tih postojale su dvije važnije grane UNIXa: AT&T-ova ***System V*** te *Berkeley Software Distribution* to jest ***BSD Unix*** (Sveučilišta u Kaliforniji). Da bi stvari bile još kompliciranije, iz tih grana UNIXa, nastale su i mnoge inačice koje su nazivane „*System V s BSD poboljšanjima*" odnosno kombinacije obije grane UNIXa. Osim toga svaka od ovih grana razvijala se i sve više razlikovala. S obzirom na činjenicu kako su se već vrlo rano u razvoju UNIXa pojavile njegove razne varijante, koje su razvijale mnoge tvrtke, institucije i sveučilišta, pojavila se i potreba za određenom standardizacijom.
Standardi su nastali iz projekta koji je počeo oko 1985. godine. Naziv *POSIX* predložio je *Richard Stallman*, a naknadno je izvedeno značenje "*Portable Operating System Interface*", pri čemu **X** predstavlja UNIX.

**POSIX** je zajednički naziv za skupinu *IEEE* standarda kojima se definira sučelje za programiranje softvera (***API***), kao i naredbene ljuske (*Shell*) te svih ostalih veza između aplikacija, usklađenih s različitim izvedenicama operativnog sustava UNIX (ili Linux). Službena oznaka je *IEEE* 1003, a naziv međunarodnog standarda je **ISO/IEC 9945**.

Na početku je postojao samo jedan dokument koji se ticao samo ***API***-ja, ali se standardizacija širila i na druge dijelove UNIXa, pa je tako narasla na 19 odvojenih dokumenata odnosno pôdstandarda: POSIX.1, POSIX.2, … .
Jedan dio *POSIX* standarda tako definira naredbenu ljusku i mogućnosti skriptiranja, kao i njihovu međuvezu, ali i sistemske varijable. Ovaj dio standarda je preuzet s takozvane **UNIX System V** naredbene ljuske (engl. *Shell*).
Drugi dio standarda je standardizirao ponašanje i rad programa u interakciji s drugim programima i sistemskim servisima. Nadalje, postoji i dio POSIX standarda koji se tiče programskog jezika **C**, kako bi se omogućila portabilnost (prenosivost) programa između različitih operativnih sustava, čak i na razini programskog kôda. To je moguće stoga jer se prema standardu za POSIX **C** API (povrh ANSI C standarda) definiraju mnoge funkcije poput: datotečnih operatora, procesa i programskih nîti, dijeljene memorije i memorijskog menadžmenta, mreže i drugog.

Dio koji se bavi standardizacijom sistemskih servisa definira ponašanje i rad ulazno/izlaznog sustava, terminala i mreže, ali i sustava datoteka i direktorija. Naime većina distribucija Linuxa usklađena je sa standardom hijerarhije datotečnog sustava (***FHS*** - *Filesystem Hierarchy Standard*). ***FHS*** definira strukturu direktorija (mapa) koja se konfigurira u obliku stabla. Prvi direktorij u hijerarhiji je korijenski direktorij, a svi ostali direktoriji, datoteke i posebne datoteke granaju se iz njega. ***Posix*** definira i niz funkcija koje su važne za pohranu, posebno u operacijama s datotekama. On definira i pravila za montiranje datotečnih sustava, za nazive datoteka i postavlja pravila o pristupu datotekama i blokovima unutar njih. Usklađenost s *Posixom* osigurava da postoje jasna pravila o tome kako se datotekama i dijelovima datoteka može pristupati i čitati, ali i zapisivati.

*POSIX* također definira i sustav vezan za višenîtnost (engl. *Threading*) kao i ***API***-je za pristup ovom sustavu. U novije vrijeme su svi zasebni *POSIX* standardi ujedinjeni u jedan veći te su definirani kao: **IEEE POSIX Std 1003.1-2008**, znan i kao **POSIX.1-2008**. Gotovo sve logičke komponente današnjih UNIX ili Linux sustava su definirane i razvijane prema *POSIX* standardu. Određene inačice UNIXa su certificirane kao *POSIX* kompatibilne; a neke od njih su:

- ***AIX*** *[tvrtke IBM]*, ***HP-UX*** *[tvrtke HP]*, ***IRIX*** *[tvrtke Silicon Graphics]*, ***OS X*** (od 10.5 Leopard) [*tvrtke Apple*].
- ***Solaris*** *[tvrtke Sun Microsystems]*, ***Tru64*** *(prije znan kao* ***Digital Unix*** *) [tvrtke DEC, kasnije Compaq pa HP]*.
- ***UnixWare*** *[tvrtke Univel, kasnije Novell pa SCO Unix] i* ***QNX Neutrino*** [*tvrtke QNX*] te još neki drugi.

Neke druge inačice *UNIXoidnih* operativnih sustava su *POSIX* kompatibilne, ali do određene granice.
Pogledajmo i neke od njih:

- ***Linux*** *i* ***Android*** te ***Darwin***, ***Illumos*** *i* ***Minix***.
- ***FreeBSD***, ***NetBSD*** *i* ***OpenBSD***.
- ***VxWorks***, ***Xenix***, kao i mnogi drugi.



Za više detalja o **POSIX**u pogledajte i *Wikipediju*: https://en.wikipedia.org/wiki/POSIX.

*POSIX* definira i lokalizaciju to jest takozvane *regionalne postavke*, koje definiraju jezik i kulturalne konvencije koje se koriste u korisničkom okruženju. Svaka lokalizacija se sastoji od kategorija koje definiraju ponašanje softverskih komponenti, kao što je formatiranje datuma i vremena te monetarno i numeričko oblikovanje sadržaja. Nastavimo s *POSIX* regionalnim postavkama odnosno njihovim varijablama. Spomenut ćemo nekoliko tih varijabli:

- `LC_CTYPE` – definira klasifikaciju karaktera (znakova) kao i konverziju velikih i malih slova te općenito baratanje s karakterima (znakovima). Osim toga programi i biblioteke koje rade s karakterima i/ili stringovima često trebaju klasificirati radi li se o brojevima, slovima, posebnim znakovima, razmaku i slično te na osnovi toga raditi određene konverzije.
- `LC_NUMERIC` – radi s numeričkim vrijednostima koje nisu monetarne.
- `LC_MONETARY` – radi s numeričkim vrijednostima koje su vezane za monetu odnosno valutu.
- `LC_TIME` – radi s datumom i vremenom (satom), definicijom naziva dana i mjeseci, formata datuma i slično.
- `LC_COLLATE` – je vezan za rad sa znakovima kod kojih leksikografske konvencije redoslijeda slova nisu jednake onima prema brojčanim oznakama (kôdovima). Primjerice redoslijed slova ***a***,***b***,***c***,***d*** je prema engleskom redoslijedu točan, ali u hrvatskom je pravilno: ***a***,***b***,***c***,***č***,***ć***. Drugi problem su slučajevi kada imamo i slova koja su sastavljena od dva slova poput ***dž***, ***lj*** i slično, jer tada može doći do krivog redoslijeda. Naime ovdje se definira ispravan način rada s ovim "posebnim" pravilima.
- `LC_MESSAGES` – on specificira "jezik" u kojemu su ili će biti postavljene potvrdne ili negativne poruke unutar raznih programa.

Pogledajmo koje su sve postavljene varijable vezana za regionalne postavke, pomoću naredbe `locale`:

```
locale

LANG=POSIX
LC_CTYPE=en_US.UTF-8
LC_NUMERIC="POSIX"
LC_TIME="POSIX"
LC_COLLATE="POSIX"
LC_MONETARY="POSIX"
LC_MESSAGES="POSIX"
LC_PAPER="POSIX"
LC_NAME="POSIX"
LC_ADDRESS="POSIX"
LC_TELEPHONE="POSIX"
LC_MEASUREMENT="POSIX"
LC_IDENTIFICATION="POSIX"
```

**Konfiguracija**
Na ***RedHat***/***Centos*** 6.x inačicama Linuxa ove postavke se definiraju u datoteci: `/etc/sysconfig/i18n` za cijeli operativni sustav.

Standardno su za poslužitelje postavljene sljedeće vrijednosti:

```
LANG="en_US.UTF-8"
SYSFONT="latarcyrheb-sun16"
```

Dakle globalna varijabla `LANG` koju smo opisali te dodatno globalna varijabla `SYSFONT` koja definira osnovni font koji se koristi za ***X11 Window*** sustav.
Ovdje je najvažnija naravno varijabla `LANG` koju je moguće (i uz sve gore navedene varijable `LC_`) definirati za sve korisnike, u datoteci: `/etc/profile`.
Ako imamo potrebu mijenjati ove varijable na razini svakog korisnika i to je moguće.
Sve što je navedeno i definirano u datoteci: `/etc/sysconfig/i18n` moguće je definirati za svakog korisnika posebno, u njegovom početnom/kućnom (*HOME)* direktoriju u datoteci imena: `.i18n`.



**RedHat**/**CentOS** v.7.x/8.x umjesto: `/etc/sysconfig/i18n` koristi konfiguracijsku datoteku: `/etc/locale.conf` za sve ove postavke.

Pogledajmo sadržaj datoteke: `/etc/locale.conf`, za ***RedHat/CentOS*** v.7+:

```
cat /etc/locale.conf
LANG="en_US.UTF-8"
```

Dakle vidimo da se u njoj nalaze praktično iste postavke kao i u starijim inačicama **RedHat**/**CentOS** Linuxa:

- Postavke jezika su na engleskom (*LANGUAGE*=`en_US`).
- Način kodiranja (zapisa) teksta je `UTF-8`.

## Je li moguće pokretati POSIX kompatibilne programe i pod primjerice Microsoft Windowsima?

Kratki odgovor je ne direktno jer ***Windowsi*** nisu ***POSIX*** kompatibilni. Međutim postoji programska komponenta naziva ***Cygwin*** koja daje ***POSIX*** kompatibilno okruženje za programiranje i izvršavanje programa koje se izvorno mogu izvršavati na operativnom sustavu ***Microsoft Windows***.
Naime pod ***Cygwinom***, izvorni programski kôd programa dizajniranih za operativne sustave slične Unixu to jest Linuxu, može se kompilirati uz minimalne izmjene te potom i izvršiti.
Instalacijski direktorij programa ***Cygwin*** ima izgled direktorija koji je sličan korijenskom datotečnom sustavu sustava sličnih Unixu, s poznatim direktorijima, kao što su `/bin`, `/home`, `/etc`, `/usr` i `/var`.

***Cygwin*** se instalira sa stotinama alata naredbenog retka i drugih programa koji se obično nalaze na sustavima sličnim Unixu odnosno Linuxu. Osim toga, mnoge aplikacije mogu se instalirati i iz ***Cygwinovog*** paketnog sustava.

***Cygwin*** se sastoji od dva dijela: programskih biblioteka za *Windowse* (tzv. *DLL* datoteke) kao sloja kompatibilnosti API-ja u obliku standardnih C biblioteka koje pružaju značajan dio funkcionalnosti **POSIX** API-ja te opsežnih zbirki softverskih alata i aplikacija koje pružaju izgled i osjećaj rada na Unixu odnosno Linuxu.
Dakle *Cygwin* je implementirao ***POSIX*** kompatibilno okruženje u obliku programskih biblioteka (*DLL*).

**Izvori informacija: (1031),(1061),(1098),(1198),(K-5), `man locale.conf`,`man localectl`,`man locale`, `man posixoptions`,`man sysconf`,`man 7 pthreads`,`man 7 signal`.**

### 10.4.4.2. API i ABI

**Aplikacijsko programsko sučelje (API)**
Aplikacijsko programsko sučelje (engl. *application programming interface)* (**API**) je veza između računala ili između računalnih programa i operativnog sustava. Ono čini skup određenih pravila i specifikacija koje se koriste u programiranju kako bismo se mogli koristiti uslugama ili resursima operativnog sustava ili nekog drugog (složenog) programa, uporabom standardnih biblioteka s rutinama (primjerice: funkcijama, procedurama, metodama i slično).
***API*** možemo promatrati kao vrstu softverskog sučelja koje nudi usluge drugim dijelovima softvera.
Dokument ili standard koji opisuje kako izgraditi ili koristiti takvu vezu ili sučelje naziva se ***API*** specifikacija. Za računalni sustav koji zadovoljava ovaj standard se kaže da implementira ***API***. Pojam ***API*** može se odnositi ili na specifikaciju ili na implementaciju. Za razliku od korisničkog sučelja, koje povezuje računalo s korisnikom, aplikacijsko programsko sučelje povezuje računalo ili dijelove softvera jedno s drugim i/ili s operativnim sustavom. Jedna od svrha ***API***-ja je skrivanje internih detalja o tome kako određeni sustav radi, otkrivajući samo one dijelove koje će programer smatrati korisnima, održavajući ih dosljednima čak i ako se interni detalji implementacije kasnije promijene.
Princip skrivanja informacija se radi i stoga kako bi omogućili modularno programiranja, konkretno skrivanjem detalja implementacije određenog programskog modula tako da korisnici modula ne moraju razumjeti složenost sustava unutar modula.
Dakle, dizajn API-ja pokušava pružiti programeru samo one značajke koje bi očekivao, odnosno koje su mu potrebne.
***API*** može biti prilagođen samo za određene sustave ili može biti zajednički standard koji omogućuje interoperabilnost među mnogim sustavima. ***API*** može definirati i sučelje između aplikacije i operativnog sustava.
**POSIX**, na primjer, specificira skup uobičajenih ***API***-ja koji imaju za cilj omogućiti da se aplikacija napisana za **POSIX** kompatibilni operativni sustav prenese na drugi **POSIX** kompatibilni operativni sustav.
Tako su primjerice *Linux* i *Berkeley Software Distribution* (***BSD Unix***) operativni sustavi koji implementiraju ***POSIX API***.
To znači da, ako se program drži ***POSIX API***-ja, da se može izvršiti i na drugom operativnom sustavu od onoga na kojem je razvijen, a koji koristi identični ***POSIX API***. Za detalje o **POSIX** standardu, pogledajte prethodno poglavlje (**10.4.4.1.**).
Osim **POSIX** standarda, koji nudi **API** programerima, postoje i mnogi drugi primjeri korištenja **API**-ja, poput: ***DirectX*** za *Microsoft Windows*, ***JAVA API***-ja, ***OpenCL***, ali i mnogih drugih. Primjerice navedeni ***OpenCL*** (*Open Computing Language*) je programski okvir (engl. *framework*) za pisanje programa koji se izvode na heterogenim platformama koje se sastoje od središnjih procesorskih jedinica (CPU), grafičkih procesorskih jedinica (GPU), digitalnih procesora signala (DSP), polja programabilnih vrata (FPGA) i drugih procesora ili hardverskih akceleratora. Tako će primjerice program napisan uporabom ovog **API**-ja raditi na bilo kojem operativnom sustavu, koji podržava i koristi taj **API**.

**Aplikacijsko binarno sučelje (ABI)**
U računalnom softveru, binarno sučelje aplikacije (engl. *application binary interface)* (**ABI**) je sučelje između dvaju komponenti binarnog programa. Često je jedna od tih komponenti sistemska biblioteka ili dio operativnog sustava, a druga je program koji koristi korisnik. ***ABI*** definira kako se strukturama podataka ili programskim rutinama pristupa u strojnom kôdu, što je format niske razine koji je ovisan o hardveru. **ABI** definira binarno sučelje i način interakcije između: programa, biblioteka i sustava. Nasuprot tome, **API** definira ovaj pristup u izvornom programskom kôdu, koji je relativno visoke razine, ali je i format neovisan o hardveru i lako čitljiv odnosno on je pisan u programskom jeziku koji je lako čitljiv i razumljiv.
***ABI*** se bavi detaljima poput instrukcija procesora koje će se koristiti tijekom pokretanja programa, načina kako bi aplikacija trebala upućivati sistemske pozive operativnom sustavu i sličnih operacija na najnižoj razini.
Međutim, čak i ***ABI***, pod uvjetom da su prisutne potrebne dijeljene biblioteke i ispunjeni drugi slični preduvjeti, dopušta programu iz jednog operativnog sustava koji podržava taj (identičan) ***ABI,*** da se izvodi bez izmjena na bilo kojem drugom kompatibilnom operativnom sustavu. To se zove i kompatibilnost binarnog kôda (engl. ***Binary-code compatibility***), što znači da je program kompatibilan s binarnim kodom ili kompatibilan s objektnim kôdom, a ona je svojstvo računalnih sustava da mogu izvoditi isti izvršni kôd (strojni kôd) to jest binarni izvršni program. Primjer ovakvog binarno kompatibilnog programskog kôda odnosno binarnog izvršnog programa su programi koji se mogu izvršavati unutar različitih generacija istog operativnog sustava. Primjerice programi pisani za ***Windows XP***, u pravilu se mogu izvršavati na ***Windows 7***, ili ***Windows 10*** operativnim sustavima, kao što se izvršni binarni programi (tzv. **ELF**), mogu izvršavati na bilo kojoj distribuciji Linuxa.



U Linux sustavu se izvršne binarne datoteke zovu "***ELF***" datoteke (engl. *Executable and Linkable Format*).
Za više detalja otvorite poveznicu: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format **(1106)**.

**Binarno (ne)kompatibilni operativni sustavi**
Binarno nekompatibilni operativni sustavi su primjerice Linux i *Windows* koji osim što imaju različiti **API**, imaju i različiti **ABI**. S druge strane, **FreeBSD** i drugi članovi **BSD** obitelji Unixa imaju binarnu kompatibilnost s Linux kernelom, tako da se pozivi sustavu koje koristi svaki Linux binarni (izvršni) program u radu, automatski prevode u pozive **BSD** sustava.
To u konačnici omogućuje da se programi (aplikacije) i biblioteke koje su izvorno razvijene za Linux, također mogu izvoditi na **BSD**-u (pr. ***FreeBSD***-u), bez promjena u njihovom izvornom (*source*) ili binarnom (izvršnom) programskog kôdu.
Na sličan način radi i program ***Wine*** *(Wine **Is** Not an **E**mulator)*; koji radi pod Linuxom i drugim **POSIX** kompatibilnim operativnim sustavim (pr. *MacOS* i *BSD*). On radi tako da se preko njega mogu pokretati izvršni (binarni) programi pisani za *Windows* OS. On prevodi *Windows* ***ABI*** te sve *Windows* **API** pozive izvornih *Windows* binarnih programa i biblioteka, u **POSIX API** pozive, te u konačnici *Windows* programe pokreće pod Linuxom.

*Što se tiče binarne nekompatibilnosti softvera, znatno složenija opcija bi bila pokretanje emulatora, koji emuliraju neki drugi operativni sustav, pa omogućavaju izvršavanje programa namijenjenih njemu, ili uporaba virtualizacije.*

**Izvori informacija: (1099),(1100),(1101),(1102),(1103),(1104),(1105),(1106),(K-4).** `man 7 libabigail`**.**

# 10.5. IRQ (*Interrupt request*) i DMA (*Direct Memory Access*)

***Slijedi napredno poglavlje (10.5.x)!***
U ovom poglavlju obradit ćemo napredna poglavlja koja se tiču signala prekida (***IRQ***) znanih kao *interrupt requesti* te metode direktnog pristupa memoriji (***DMA***), znane kao *direct memory access* metode.

## 10.5.1. IRQ (*Interrupt request*)

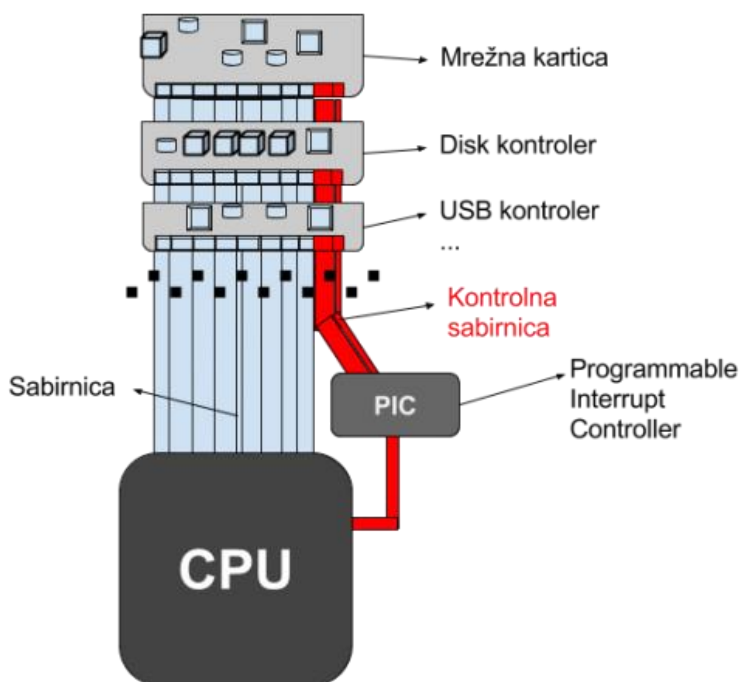
Kada neka hardverska komponenta (mrežna, grafička, zvučna ili neka druga kartica) odnosno uređaj zatraži pristup procesoru (*CPU*), taj pristup se zatraži kreiranjem signala prekida (engl. *Interrupt request*). Upravo signalom prekida (*IRQ*) uređaj upozorava procesor da je došlo do događaja zbog kojeg se traži malo vremena od procesora, kako bi se odradilo ono što nam poručuje uređaj koji je kreirao signal prekida. Ovi događaji su primjerice zaprimljeni paketi (podaci) na mrežnoj kartici, podaci koje treba snimiti na disk i slično. Operativni sustav tada prvo otkriva od koje komponente je kreiran signal prekida, te poziva posebnu proceduru koja se nalazi unutar upravljačkog programa konkretnog uređaja koji je kreirao signal prekida.
Ova procedura, koja se mora nalaziti unutar svakog upravljačkog programa za svaki pojedini uređaj (hardver), zove se *interrupt handler* procedura. Pomoću ove procedure se kontaktira komponenta unutar operativnog sustava koja je zadužena za prebacivanja između programa (procesa), a koja se zove **Linux *Process Scheduler***, jer se od tog trenutka operativni sustav mora pobrinuti da se signal prekida prihvati i počne obrađivati, za što je potrebno prebacivanje procesora na potprogram zadužen za tu namjenu (*interrupt handler*). Naime ovakvi događaji, aktivirani signalima prekida (*interruptima)* imaju vrlo visoki prioritet u odnosu na druge programe, koje trenutno obrađuje sâm procesor (CPU). U tom trenutku Linux *Process Scheduler* privremeno zaustavlja program/aplikaciju odnosno proces koji je na obradi te kreće s obradom podataka, koju je zatražio hardver koji je i kreirao signal prekida. Dakle procesor pokreće ovaj posao i odrađuje ga, nakon čega se proces koji je bio pauziran, ponovno učitava i pokreće.



Za više informacija o ***task*** odnosno ***proces scheduler*** mehanizmu, pogledajte poglavlje:
**9.3 Task/process scheduler**.

Tijekom obrade tog odnosno svakog zahtjeva, rade se mnoge operacije čitanja ili pisanja u i prema RAM memoriji, kao i diskovnom podsustavu, za koje je također zadužen centralni procesor (CPU).
Hardverski signali prekida (*Interrupt-i [IRQ]*) se koriste kod svake pojedine aktivnosti nekog uređaja: poput mrežne i zvučne kartice, modema, USB uređaja, uređaja na serijskom ili paralelnom sučelju (portu), disk kontrolera (SATA/ATA/SCSI) i slično. Čak i svako pomicanja miša ili stiskanje tipki na tipkovnici generira signale prekida (*IRQ)*.
Kako bi CPU mogao reagirati na ove signale prekida, potrebna je komponenta koja se brine za signale prekida, a zove se **PIC** (*Programmable Interrupt Controller*). **PIC** je prije bilo spojen na sve uređaje na sabirnici, preko posebnih linija odnosno vodova koji su zaduženi za *signale prekida*. A on je komunicirao s *CPU*om i prosljeđivao ove signale preko tih vodova/linija prema procesoru. *PIC* se fizički nalazio unutar takozvanog *sjevernog mosta* (*Southbridge chipseta)*, a logički spojen, kao na slici 51.
Pojednostavljeno, navedena arhitektura računala, koja koristi signale prekida (*IRQ*) izgleda okvirno kao na slici 51.

**Slika 51. Upotreba *PIC* kontrolera.**



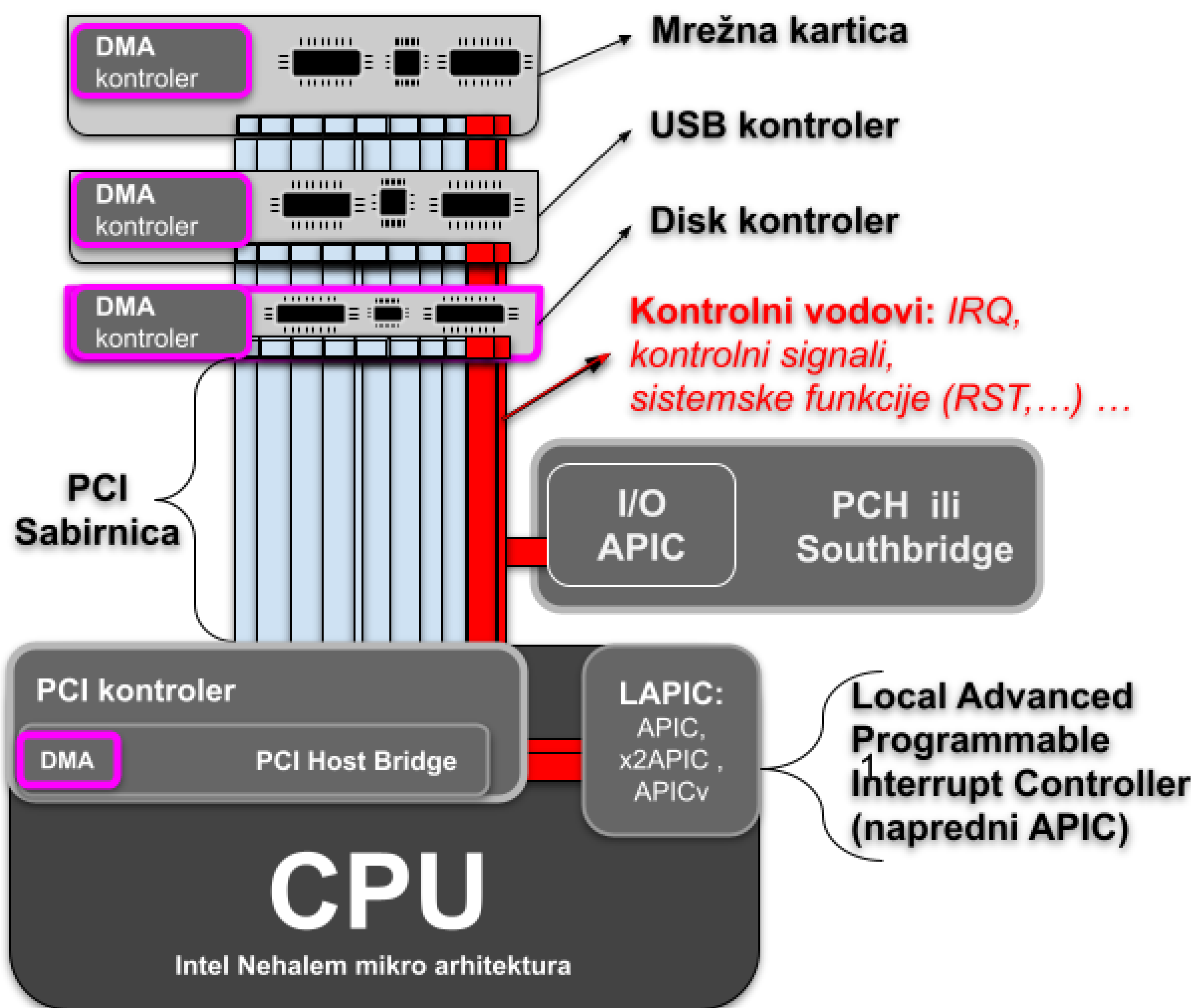

Za upotrebu i kreiranje signala prekida bio je nužan ***PIC*** kontroler. **PIC** (engl. *Programmable Interrupt Controller***)** je podržavao do 16 signala prekida (*interrupta*) i to IRQ: 0 – 15, te samo jedan fizički procesor (CPU). *PIC* je podržavao i mogućnost dijeljenja signala prekida između više uređaja.

Dijeljenje signala prekida između više uređaja je povećalo broj uređaja koji se mogao koristiti na jednom računalu, ali je za uređaje koji su zahtijevali bržu komunikaciju uveo usporavanja, a ponekad i probleme u radu.

U novije vrijeme ***PIC*** je zamijenjen s **APIC** (engl. *Advanced Programmable Interrupt Controller*) sklopom, koji je omogućio upotrebu više procesorskih jezgri unutar fizičkog procesora istovremeno, na jednoj matičnoj ploči. Ova metoda upotrebe više jezgri procesora se zove **SMP** (*Symmetric multiprocessing*).

Upotreba *APIC*a je dovela do toga da se dio ovih funkcionalnosti prebacio unutar sâmog procesora (*CPU*a), unutar kojeg je sada integriran takozvani **LAPIC**, a jedan dio je ostao kao vanjska komponenta, koja se naziva **I/O APIC**.

**Slika 51.1. Upotreba I/O APIC sa LAPIC dijelom.**




## Pogledajmo i I/O APIC + LAPIC arhitekturu

**I/O APIC** obrađuje signale prekida (*IRQ*) sa svih uređaja na glavnoj sabirnici te ih prosljeđuje prema ***LAPIC*** odnosno lokalnom APIC-u unutar procesora (CPU-a), prema svim postojećim procesorskim jezgrama. Osim toga, *APIC* podržava do 224 iskoristiva signala prekida (u teoriji) te sâmim time i veći broj uređaja koji se mogu koristiti unutar jednog računala.
U praksi broj signala prekida za primjerice ***PIIX3*** integrirani sklop (*chipset)* je bio do maksimalno 24.

## PCI sabirnica

Važno je razumjeti da na PCI sabirnici svaki uređaj generira signal prekida (IRQ) putem posebne linije (voda).
Taj signal tada dolazi do ***PIC/APIC/LAPIC*** koji preusmjeravaju signal prekida prema procesoru koji ga dalje obrađuje. Pojednostavljena shema ovakvog rada je vidljiva na slici **51.1.**
Na PCI sabirnici, ***PIC/APIC/LAPIC*** upravljaju fizičkim linijama za hardverske signale prekida (IRQ) te njihovom dodjelom procesoru.

## Novije inačice PCI sabirnice te PCI Express sabirnica

U novije vrijeme, nakon uvođenja standarda PCI v.2.2 za PCI sabirnice, kao i na svim inačicama ***PCI Express*** sabirnice, omogućena je i upotreba **MSI** (*Message Signaled Interrupts*) i/ili novijeg **MSI-X** standarda za rad s hardverskim signalima prekida (IRQ), na potpuno drugačiji način.
Naime *MSI* i *MSI-X* signali prekida ne šalju se preko posebne linije (voda) na sabirnici, već koristeći sabirnicu kao medij za komunikaciju generiranjem posebnog zapisa u takozvani *memorijski mapirani registar* (*MMIO*) s informacijama o signalu prekida. Važno je znati da *PCI Express* (**PCIe**) sabirnica uopće nema posebne linije (vodove) za slanje signala prekida jer se na njoj sva komunikacija odvija samo i isključivo slanjem posebnih komunikacijskih poruka (poput mrežnih paketa).
Ove poruke mogu sadržavati signale prekida (**IRQ**), **DMA** zahtjeve ili podatke. Stoga na ***PCIe*** sabirnici, *MSI/MSI-X* paketi eliminiraju potrebu za fizičkim linijama za signale prekida, dok **APIC/LAPIC** i dalje rukuje paketima za signale prekida, omogućujući skalabilniju i učinkovitiju komunikaciju.

## MSI (*Message Signaled Interrupts*) - o čemu se radi?

Tradicionalno, svaki uređaj spojen na **PCI** sabirnicu ima takozvani *interrupt line* odnosno vodič (*pin*) na koji se šalje signal svaki puta kada se želi poslati signal prekida prema procesoru. Dok **MSI** donosi potpuno drugačiji način rada, tako što se više ne moraju koristiti zasebni vodovi kroz koje se šalju signali prekida, već se oni šalju kroz podatkovni (normalni) dio sabirnice na posebnu memorijsku adresu, koja se naziva memorijski preslikana adresa (engl. *Memory-mapped I/O [**MMIO**]*).

Potom se prosljeđuje signal prekida prema procesoru. Važno je znati da ***PCI Express*** sabirnica uopće niti nema zasebne linije odnosno vodove za signale prekida, već se kod nje koriste samo **MSI** metoda ili novija ***MSI-X*** metoda rada sa signalima prekida. U današnje vrijeme najčešća je upotreba **MSI** metode za rad sa signalima prekida, koja je u prosjeku ubrzala obradu ovih signala za oko tri (3) puta. **MSI** omogućava maksimalno 32 signala prekida.
***MSI*** radi tako što se takozvani *interrupt vector* zapisuje direktno u *LAPIC*, a koji se danas nalazi unutar sâmog procesora, koji onda dalje odrađuje sve što je potrebno. Na ovaj način se još više ubrzao rad.
Od 2009 godine sve *Intelove* implementacije *LAPIC* unutar *CPU-a* podržavaju do 224 signala prekida.

## MSI-X

Noviji standard **MSI-X** (definiran u standardu PCI v.3.0) omogućava alociranje do 2048 signala prekida.
U praksi to ne znači kako su svi implementirali toliki broj, a prema standardu je minimalan broj njih 64.
Sve više od toga ovisi o proizvođaču i implementaciji.

Važno je znati i to da uređaji na **PCI** ili **PCI Express (PCIe)** sabirnici zapravo nisu spojeni direktno na ***PCI/PCIe*** sabirnicu već su spojeni preko posebnog ***PCI bridge*** uređaja (tj. ***PCIe Switch ili Bridge*** uređaja za **PCIe** sabirnicu).
***PCI bridge*** ili ***PCIe Switch*** služe za proširivanje kapaciteta sabirnice zbog spajanja više *PCI/PCIe* uređaja na nju, zbog slabljenja električnih signala. Odnosno ***PCI bridge*** je međuveza između procesora i sâme sabirnice, pa dodatno možemo vidjeti i: **PCI na PCI *bridge*** ili **PCI na ISA *bridge***, jer je moguća i veza s PCI na ISA sabirnicu koju ovakvi sklopovi odrađuju.

**PCIe** sabirnica radi potpuno drugačije, pa ona između procesora i ***PCIe Switch*** ili ***Bridge*** sklopa ima poseban centralni sklop koji se naziva ***Root complex*** koji omogućuje simultani prijenos podataka (pa tako i ***DMA*** i ***IRQ***) bez konkurencije između uređaja na sabirnici, kao što je slučaj kod PCI sabirnice.



Za više informacija o **PCI** i ***PCI Express*** (**PCIe**) sabirnicama i DMA pogledajte poglavlja:
**10.1.1. Matična ploča, CPU, *chipset* i sabirnica(e).**
**10.6. Direktan pristup memoriji – DMA.**

Želimo li vidjeti fizičku hijerarhijsku međuvezu između raznih PCI uređaja i ***PCI bridge*** uređaja, to možemo vidjeti s naredbom:
**`lspci -t`**

```
-+-[0000:00]-+-00.0
 |           +-01.0-[11]--
 |           +-01.1-[02]--
 |           +-02.0-[04]--+-00.0
 |           |            +-00.1
 |           |            +-00.2
 |           |            \-00.3
 |           +-02.1-[12]--
 |           +-02.2-[03]----00.0
 |           +-02.3-[13]--
 |           \-1f.0
 +-[0000:1f]-+-08.0
 |            +-08.3
 |            +-08.4
 |            \-13.6
 +-[0000:20]-+-00.0-[2d]--
 |            +-01.0-[24]—
…
```

Odnosno za listu svih PCI *bridge* uređaja možemo koristiti naredbu:
**`lspci | grep -i bridge`**

Provjerimo je li naš kernel uopće kompiliran s podrškom za ***MSI*** ili ***MSI-X*** signale prekida (očekujemo vrijednost `=y`)

```
grep CONFIG_PCI_MSI /boot/config-`uname -r`
```

**Za više detalja o MSI signalima prekida pogledajte izvore informacija: (193),(193.1),(193.2)**

### Virtualizacija signala prekida

S obzirom na to kako se razvijao svaki segment virtualizacije, postalo je potrebno imati i podršku za virtualizaciju signala prekida. Ova tehnologija se pojavila u novijim ***Intel*** i ***AMD*** procesorima. *Intelov* naziv za tu tehnologiju je **`APICv`**, a *AMD*: *Advanced Virtual Interrupt Controller* (**`AVIC`**). Ova tehnologija je dostupna od pojave *Intelovih* procesora "*Ivy Bridge EP*", negdje od kraja 2013. godine. Ona ubrzava sve radnje vezane za signale prekida u virtualizaciji (na virtualnim računalima).

### Što je IPI (Inter-Processor Interrupt) ?

**IPI** je posebna vrsta signala prekida pomoću kojega u *NUMA* konfiguraciji računala s više fizičkih procesora na jednoj matičnoj ploči, jedan fizički procesor može poslati signal prekida drugom procesoru.
Ovo se događa, ako jedan procesor treba od drugog procesora neku posebnu radnju, koja može biti:

- Zahtjev prema *memorijskom kontroleru* drugog procesora. Ovo se događa kod dohvaćanja memorije koja ne pripada prvom već drugom fizičkom procesoru. Recimo da CPU 1 želi pristupiti RAM memoriji od CPU 2 ili obratno, odnosno prema njegovom *memorijskom kontroleru* i u konačnici RAM memoriji. **Pogledajte poglavlje: 10.7.2.2.**
- Zahtjev za zaustavljanjem rada prema drugom procesoru; primjerice kada se gasi računalo.



Signali prekida o kojima smo do sada pričali se nazivaju hardverski signali prekida odnosno standardni signali prekida (**IRQ/*interrupti***)

## Posebni signali prekida

### NMI — nemaskirajući signali porekida (engl. *Non-maskable interrupt*)

Postoji i posebna kategorija definiranih signala prekida, koji imaju prioritet izvršavanja ispred svih ostalih. To su takozvani ne maskirajući signali prekida **NMI** (engl. *Non-maskable interrupt*).
**NMI** se koristi kada je vrijeme odziva kritično te se ovaj signal prekida mora odraditi u što kraćem roku.
Ovi signali prekida se koriste kada je potrebno obavijestiti sustav o nekoj kritičnoj grešci na hardveru, za debuggiranje ili za resetiranje sustava.

### Softverski signali prekida

Softverski signali prekida (*Soft. interrupti)* se događaju kada se dogodi neki poseban uvjet (engl. *Exception*) unutar softvera. Kada se recimo u programu naloži dijeljenje s nulom, ova nemoguća operacija će izazvati poruku koju će nam vratiti sustav: *divide-by-zero exception (dijeljenje s nulom)*, pri čemu će se ili odustati od operacije ili će doći do greške. U svakom slučaju pojavit će se softverski signal prekida pri ovom (i svakom ovakvom) pokušaju. Drugi primjer je komunikacija s diskovnim (*I/O)* sustavom tijekom koje se kreira softverski signal prekida kod zahtjeva za operacijom čitanja ili pisanja, kao i kod rada s mrežnim podsustavom. Svaki od tih softverskih signala prekida ima svoju rutinu za obradu signala prekida, koja se naziva *interrupt handler*, a koja vrijedi i za hardverske signale prekida. S time da softverskih signala prekida može biti definiranih na stotine ili više, te se oni mogu znatno brže generirati od hardverskih signala prekida. Dakle možemo reći kako su softverski signali prekida namijenjeni za ulazno/izlazne (I/O) operacije, kao i za komunikaciju između procesa tj. pokrenutih programa.
Međutim, moguća je i događa se i veza između hardverskih i softverskih signala prekida. Naime neki uređaj, poput mrežne kartice[(195)], prvo će preko hardverskog signala prekida zatražiti od CPUa vrijeme za obradu mrežnih paketa, na što će reagirati upravljački program za tu mrežnu karticu. Zapravo mrežna kartica preko upravljačkog programa, jer ona i njen upravljači program podržavaju **DMA** prijenos podataka, bez posredovanja CPUa, dobiva pristup u posebnu regiju RAM memorije koja se često naziva prstenasta memorija (*ring buffer)*, pohranjuje primljene mrežne pakete. Potom ona kreira hardverski signal prekida preko svog upravljačkog programa. Tada kernel operativnog sustava odgovara na ovaj hardverski signal prekida te preuzima mrežne pakete iz ove prstenaste memorije te ih privremeno sprema u svoju među memoriju odnosno mrežni nîz za obradu, koji se naziva *Network queue*.
Tada se generira softverski signal prekida kojim se dalje nastavlja dio primanja paketa, koji će obraditi proces (program) koji je zadužen za obradu mrežnih paketa, a koji obično te pakete počinje spremati u svoje međumemorije, koje se nazivaju *queue* ili nîzovi za primanje (ili za slanje paketa). Važno je znati kako i za primanje i slanje mrežnih paketa, postoji više mrežnih međumemorija, na razini operativnog sustava. Dalje ponašanje ovisi o implementaciji upravljačkog programa i hardvera. Konkretno za Linux, pitanje je podržavaju li ***Pooling*** i ***Interrupt moderation*** funkcionalnosti preko **NAPI** frameworka. Nadalje, proces (program) zadužen za obradu mrežnih paketa, može kreirati nove softverske signale prekida kada treba komunicirati s drugim programima, koji su zaduženi za specifične mrežne ili neke druge funkcionalnosti.



Za više informacija o ***softverskim signalima prekida*** pogledajte poglavlje:
**10.5.1.3. Servis *ksoftirqd i softverski signali prekida*.**

**Izvori informacija: (81),(82),(83), (194),(195),(196),(197),(K-2),(K-5).**

#### 10.5.1.1. IRQ i Linux

U ovoj cjelini vidjet ćemo vezu između signala prekida (*IRQ*) i Linuxa.
Za praćenje detalja rada signala prekida, naš kernel mora biti kompiliran s opcijom `CONFIG_IRQ_TIME_ACCOUNTING`, što možemo provjeriti sa sljedećim nîzom naredbi:

```
grep CONFIG_IRQ_TIME_ACCOUNTING /boot/config-`uname -r`
```

Ako je ona postavljena na `=Y`, to znači da će programi poput: `top`, `mpstat` i drugih, uredno prikazivati sve napredne statistike vezane za procesiranje odnosno obradu hardverskih signala prekida. Primarno se statistike signala prekida zapisuju u datoteku: `/proc/stat`, u retku odnosno statistici: ***intr***. Statistike na razini cijelog Linuxa, za svaki pojedini hardverski signal prekida, prema uređaju koji ga kreira, moguće je pratiti pogledom na posebnu datoteku `/proc/interrupts`.

Pogledajmo hardverske signale prekida (*interrupte)* na našem Linuxu i to na starijem *Intel Xeon* procesoru (ispis je skraćen):

```
cat /proc/interrupts

           CPU0       CPU1
  0:      18667       0          IR-IO-APIC-edge      timer
  1:          2       0          IR-IO-APIC-edge      i8042
  8:          1       0          IR-IO-APIC-edge      rtc0
  9:          0       0          IR-IO-APIC-fasteoi   acpi
 12:          4       0          IR-IO-APIC-edge      i8042
 17:        158       0          IR-IO-APIC-fasteoi   ata_piix
 20:       1241       0          IR-IO-APIC-fasteoi   ehci_hcd:usb1, uhci_hcd:usb2
 50:  106628875       0          IR-PCI-MSI-edge      hpsa0
 54:  306816715       0          IR-PCI-MSI-edge      eth0
 55:  145395108       0          IR-PCI-MSI-edge      eth1
```

**Pogledajmo i druge detalje našeg ispisa:**
Prvi (zatamnjeni) stupac označava konkretan broj signala prekida (`0`, `1`, `8`,…), potom slijede stupci `CPU0` i `CPU1` gdje se vidi koliko je hardverskih signala prekida obradila koja *CPU* jezgra (u primjeru imamo procesor sa dvije CPU jezgre: CPU0 i CPU1). Statistike su iskazane u broju signala prekida u svakoj sekundi (za svaki CPU [0 i 1] jer imamo 2 CPUa).
U našem slučaju sve signale prekida odrađuje prvi CPU (`CPU0`) što je u redu.

Zatim vidimo vrstu signala prekida (u četvrtom stupcu):
- Prvi nîz uređaja koristi ***I/O APIC*** (`IR-IO-APIC-`), a to su standardni signali prekida.
- Drugi nîz uređaja koristi ***MSI (***`IR-PCI-MSI-edge`***)*** odnosno novije *MSI* signale prekida.

Vezano za vrstu signala prekida (četvrti stupac) ovdje vidimo sljedeće:
- Prisjetite se **I/O APICa**. Naime *I/O APIC* je napredni kontroler signala prekida koji je zadužen za prosljeđivanje signala prekida na druge *CPU* jezgre ili *CPU* grupe. Naime bez njega, signal prekida koji je kreirao određeni uređaj (hardver) i zaprimila ga *CPU* jezgra koja je i pokrenula operativni sustav, ona bi ga morala i obraditi. Međutim, pošto se koristi **I/O APIC**, *CPU* jezgra koja je zaprimila signal prekida od uređaja koji ga je kreirao, može taj zahtjev proslijediti na neku drugu *CPU* jezgru. Ovakvi signali prekida su vidljivi kao `IR-IO-APIC`.
- Druga vrsta su **MSI** signali prekida, o kojima smo također govorili. Za uređaje koji koriste ovu vrstu signala prekida, to je vidljivo kao `IR-PCI-MSI`.

I u zadnjem stupcu je naveden konkretan uređaj koji koristi određeni signal prekida (objasnit ćemo ih sâmo nekoliko):
- `timer` – poznat je kao "*timer interrupt*" i vezan je za IRQ 0. Njega koristi operativni sustav za održavanje sustava u sinkronizaciji te za izvršavanje zadataka poput generiranja signala prekida, prebacivanje konteksta i slično.
- `rtc0` – *Real Time Clock* (***RTC)*** - ovo je interni sat na matičnoj ploči računala; on ovdje koristi IRQ 8.
- `acpi` – *ACPI* (engl. *Advanced Configuration and Power Interface*); koristi se za prepoznavanje i konfiguraciju hardvera, upravljanje napajanjem, praćenjem komponenti sustava i drugo, on ovdje koristi IRQ 9.
- `ata_piix` – ATA kontroler na koji je spojen samo CD/DVD-ROM, koji ovdje koristi IRQ 17.
- `hpsa` je *HP Smart Array RAID* kontroler na kojem su sistemski diskovi u **RAID 1** polju, on ovdje koristi IRQ 50.
- `eth0` i `eht1` – mrežne kartice, one ovdje koriste IRQ 54 i IRQ 55.

Vidimo kako je najveći broj signala prekida (*IRQ*) prema disk kontroleru (disku) i prema mrežnim karticama, koje su obje stalno aktivne, što i odgovara stvarnosti. Sada postaje jasno koliko se signala prekida kreira u svakoj sekundi.

Za sve mrežne kartice, između ostalih podataka i podatak koji ***IRQ*** koristi koja mrežna kartica, možete vidjeti s naredbom:

```
netstat -ie
```

Dodatne detalje moguće je vidjeti na razini svakog signala prekida.

Pogledajmo *IRQ 54*, a njega u našem slučaju koristi `eth0` mrežna kartica:

```
ls -a /proc/irq/54/

affinity_hint   /eth0     node      smp_affinity    smp_affinity_list    spurious
```

Naime svaka od ovih datoteka sadrži određena očitanja i parametre.
Recimo datoteka `smp_affinity` sadrži definiciju može li uređaj koji koristi ovaj signal prekida koristiti bilo koju jezgru procesora (CPU) za baratanje s ovim signalom prekida ili je "zaključan" na točno određenu *CPU* jezgru (Pr. CPU br.0).



Vezano za ***SMP*** sustave, pogledajte poglavlje:
**10.7.2.1. SMP.**

## Afinitet signala prekida prema CPU jezgrama (*IRQ affinity*)

Pogledajmo naš trenutni slučaj za signal prekida broj `54`, koji sada konkretno koristi mrežna kartica `eth0`:

```
cat /proc/irq/54/smp_affinity

000000ff
```

Ovdje `ff` znači kako bilo koja *CPU* jezgra može odrađivati signal prekida broj `54`. Na našem Linux sustavu smo vidjeli kako je sve signale prekida odradio CPU0, što je u redu, ali u nekim slučajevima postoji potreba da se oni (IRQ) ipak raspoređuju na više procesorskih jezgri podjednako. Pri tome ovakav rad može i uzrokovati usporavanja, ali postoje slučajevi kada ipak želimo ovakav rad. Dakle za neke primjene želimo da se neki signali prekida izvršavaju samo i uvijek na jednoj jezgri procesora.

Ako primjerice želimo da mrežne kartice `eth0` i `eth1` obrađuje samo jezgra CPU0, kao u našem slučaju, a na drugim *CPU* jezgrama da se odrađuje nešto drugo, takav primjer ćemo opisati dalje u tekstu. Ali prvo moramo razumjeti na koji način sustav označava jezgre procesora. Pogledajmo i reprezentaciju oznaka prema jezgrama *CPUa* (radi se o takozvanoj *bît maski*).

Maksimalna standardna bît maska je 32 bitna vrijednost odvojena s po četiri bita za po četiri jezgre, pa se prema tome može definirati do 32 CPU jezgre jer ukupno imamo 8 segmenata po četiri bita, kako je vidljivo u tablici dolje.

| Koja CPU jezgra se može koristiti | Binarna oznaka | Heksadecimalna oznaka |
|---|---|---|
| CPU 0 | 0001 | 1 |
| CPU 1 | 0010 | 2 |
| CPU 2 | 0100 | 4 |
| CPU 3 | 1000 | 8 |
| | | |
| CPU 4 | 0001 0000 | 10 |
| CPU 5 | 0010 0000 | 20 |
| CPU 6 | 0100 0000 | 40 |
| CPU 7 | 1000 0000 | 80 |

| Koja CPU jezgra se može koristiti | Heksadecimalna oznaka |
|---|---|
| CPU 8 | 100 |
| CPU 9 | 200 |
| CPU 10 | 400 |
| CPU 11 | 800 |

U slučaju kada želimo dati mogućnosti izbora korištenja više CPU jezgri istovremeno, tada se vrijednosti zbrajaju:

| Koja CPU jezgra se može koristiti | Binarna oznaka | Heksadecimalna oznaka |
|---|---|---|
| CPU 0 ili CPU 1 | 0011 | 3 |
| CPU 2 ili CPU 0 | 0101 | 5 |
| CPU 2 ili CPU 1 | 0110 | 6 |
| CPU 0 ili CPU 1 ili CPU 2 | 0111 | 7 |
| Sve jezgre od ponuđenih | 1111 | F |

Kada imamo računalo s više od četiri CPU jezgre, dodaju se vrijednosti prema sljedećem principu u tablici.

Uz napomenu kako je zadnja oznaka uvijek `F`. Svaki nîz od 4 bita označava nîz od četiri CPU jezgre, s time da se heksadecimalne oznake popunjavaju od desno i to od zadnjeg slova `F`, na lijevo:

| Nîz od po četiri CPU-a | Heksadecimalna oznaka |
|---|---|
| Sva četiri prva CPU-a (CPU0-3) | 000000FF |
| Sve četiri prva grupe (CPU0-3) i treće grupe (CPU8-11) | 00000F0F |
| Sve četiri prve i sve četiri druge i treće grupe CPU-a (CPU0-11) | 00000FFF |
| … | … |
| Sve prve četiri grupe CPU-a (CPU0-15) | 0000FFFF |
| . . . | . . . |
| Svih osam grupa CPU-a (CPU0-31) | FFFFFFFF |

Ako imamo sustav koji ima do 64 CPU jezgre, onda se koristi drugačije označavanje: *DRUGI 32 bitni segment, PRVI 32 bitni segment*, što izgleda ovako: `FFFFFFFF,00000000` → ovo bi značilo sve gornje 32 CPU jezgre (*CPU32-63*)
Ili na primjer: `FFFFFFFF,FFFFFFFF` → ovo bi značilo sve 64 CPU jezgre (*CPU0-63*)

U slučaju prema prvoj tablici, ako želimo da se mrežna kartica `eth0`, koja koristi `CPU 0` na `IRQ 54`, prebaci na `CPU 1`, tada treba napraviti sljedeće (ovo definiranje se broji od **1**, tako da **2** predstavlja CPU jezgru **1**):

```
echo 2 > /proc/irq/54/smp_affinity
```

I sada vidimo kako se signal prekida broj `54` odrađuje samo na CPU 1 jezgri procesora (skratili smo izlistanje):

```
cat /proc/interrupts

           CPU0       CPU1
54:   3 06816715       187        IR-PCI-MSI-edge        eth0
```

Dakle u prvom stupcu je vidljiv broj signala prekida (`54`) u stupcima za CPU0 i CPU1 su brojači broja hardverskih signala prekida za pojedinu CPU jezgru (0 ili 1 u ovom slučaju).

Potom slijedi oznaka vrste signala prekida (`IR-PCI-MSI-edge`) te uređaj na koji se signal prekida odnosi (`eth0`).
Stoga vidimo kako se `IRQ 54` preselio na `CPU jezgru 1` (`CPU1`) jer je brojač na `CPU1` za taj `IRQ` počeo rasti.

Konkretno, a najviše vezano za mrežne kartice, kod vrlo velikih brzina mreže (10/20/50/100+Gbps), i to kod primjena u kojima je potrebna iznimno velika brzina obrade mrežnih paketa, vrlo često se ručno konfigurira pripadnost signala prekida mrežne kartice točno i sâmo određenoj jezgri (ili jezgrama) procesora (CPUa).



Za dodatne primjere analize upotrebe signala prekida, pogledajte poglavlja:
**10.5.1.2. Servis *irqbalance*.**
**10.7.2.3.6 Naredba mpstat za CPU.**

Za optimizaciju rada mrežnih kartica vezanih za signale prekida (**IRQ**) pogledajte slijedeća poglavlja te poglavlje:
**25.2. Dodatne mogućnosti linuxa.**

Za praćenje hardverskih signala prekida (IRQ) proučite i naredbe:

- `lsirq` – za izlistanje signala perkida.
- `irqtop` – za ispis statistika svih signala prekida na sustavu, u stvarnom vremenu.

**Izvori informacija: (42),(43),(84) , `man netstat`,`man lsirq`,`man irqtop`,`man top`,`man 5 proc`.**

### 10.5.1.2. Servis *irqbalance*

U većini distribucija Linuxa postoji poseban Linux program (engl. Linux *thread)* odnosno servis koji radi na vrlo niskoj razini, a pokreće ga `kthreadd` servis. On se zove `irqbalance` i njegov zadatak je raspoređivanje hardverskih signala prekida između svih dostupnih jezgri procesora i to jednoliko. Dakle ovaj servis svako malo preraspodjeljuje signale prekida s jedne na drugu jezgru procesora, ne bi li sve jezgre procesora jednoliko bile opterećene obradom signala prekida. Međutim, zbog toga se sustav može i usporiti zbog stalnog prebacivanja signala prekida između raznih jezgri procesora na obradu. Pogotovo u **NUMA** sustavima u kojima se nova jezgra procesora može nalaziti na drugom fizičkom procesoru što u određenim slučajevima može usporiti komunikaciju. ***Ipak, novije implementacije ovog servisa dobro barataju i s NUMA topologijom***.



Za više detalja o **NUMA** arhitekturi pogledajte poglavlje:
**10.7.2.2 NUMA.**

Naime ovaj servis stalno prati koliko je signala prekida raspoređeno između svih jezgri procesora i ako u nekom trenutku signali prekida nisu ravnomjerno raspoređeni on ih ponovno preraspodjeljuje. To u nekim slučajevima kao kod vrlo brzih mreža (10++Gbps), u slučaju obrađivanja ogromnog broja mrežnih paketa (za usmjerivače i slične uređaje) nije dobro rješenje.
Ovaj servis je moguće i dodatno optimizirati, a njegova inicijalna konfiguracija se nalazi u datoteci: `/etc/sysconfig/irqbalance`. Dok se sve opcije i parametri rada ovog servisa, poput isključivanja CPU jezgri koje *Irqbalance* servis smije koristiti, definiraju u datoteci: `/etc/sysconfig/irqbalance`.



AKO NISTE SIGURNI ŠTO RADITE, NAJBOLJE NE DIRAJTE NIŠTA!.

Kako trenutno zaustaviti ovaj servis (***RedHat/CentOS*** 6.x), s naredbom `service` na sljedeći način:

```
service irqbalance stop
```

Odnosno za ***RedHat/CentOS*** 7.x/8.x., bi ga zaustavili sa sljedećom naredbom:

```
systemctl stop irqbalance
```

Kako trajno (permanentno) zaustaviti ovaj servis (***RedHat/CentOS*** 6.x), s naredbom `chkconfig` na sljedeći način:

```
chkconfig irqbalance off
```

Odnosno na ***RedHat/CentOS*** 7.x/8.x, ga trajno možemo zaustaviti s naredbom `systemctl` na sljedeći način:

```
systemctl disable irqbalance
```

**Sada pogledajmo i drugo Linux računalo** s *Intel Core 2 Duo E8500* procesorom, koji ima dvije CPU jezgre te aktivan proces `irqbalance`. Provjerimo je li stvarno i aktivan `irqbalance` proces:

```
ps -ef |grep irqbalance | grep -v grep

root      1820    1  0 Nov24 ?       00:00:13 irqbalance —pid=/var/run/irqbalance.pid
```

Vidimo kako je ovaj proces stvarno aktivan jer ga imamo vidljivog u ovom izlistanju, pod: `irqbalance`.

Sada pogledajmo kako su raspodijeljeni signali prekida (skratili smo ispis), zahvaljujući ovom procesu/servisu: `irqbalance`:

```
cat /proc/interrupts

           CPU0     CPU1
  0:        845      137   IO-APIC-edge      timer
  1:          6        2   IO-APIC-edge      i8042
  8:          1        0   IO-APIC-edge      rtc0
  9:          0        3   IO-APIC-fasteoi   acpi
 12:         98       54   IO-APIC-edge      i8042
 14:      72239    69192   IO-APIC-edge      ata_piix
 18:          0        0   IO-APIC-fasteoi   ehci_hcd:usb1,uhci_hcd:usb4,ata_generic
 29:         21        6   PCI-MSI-edge      i915
 30:      35447   411265   PCI-MSI-edge      eth0
 31:        294      280   PCI-MSI-edge      snd_hda_intel
```

Vidljiva je slična lista uređaja koji koriste hardverske signale prekida (*IRQ*), ali su oni zahvaljujući `irqbalance` raspodijeljeni između obje jezgre procesora. Moguće je u slučaju kada određeni *IRQ* želimo ručno *zalijepiti* na određenu CPU jezgru, taj IRQ isključiti iz dostupnih signala prekida (*IRQ*) s kojima barata `irqbalance`. Tako možemo dobiti mogućnost da određene uređaje vezane za određeni *IRQ* uvijek vežemo za željenu CPU jezgru, ali za sve ostale ipak želimo koristi `irqbalance` mehanizam za raspodjelu *IRQa*, prema CPU jezgrama, kako ih on već automatski raspodjeljuje.
Uzmimo slučaj naše mrežne kartice `eth0` koja koristi *IRQ* `30` te ju vežimo za CPU jezgru **0** (ovo definiranje se broji od **1**).

```
echo 1 > /proc/irq/30/smp_affinity
```

Sada ovaj signal prekida (*IRQ*) mičemo iz `irqbalance`-a, tako da prvo zaustavimo `irqbalance` te ga pokrenemo na sljedeći način:

```
/usr/sbin/irqbalance --banirq=30
```

**Napredni dio**
Ako imamo poslužitelj koji koristi ***NUMA*** arhitekturu odnosno koji ima dva fizička procesora, svaki sa određenim brojem logičkih jezgri procesora, tada su optimizacije ovdje vrlo značajne. Isto je tako i ako imamo višestruka mrežna sučelja koja rade na minimalno 10 Gbps te je potrebna velika brzina obrade mrežnih paketa.



Prvo se dobro podsjetite **NUMA** arhitekture, pa pogledajte poglavlje:
**10.7.2.2 NUMA.**
Pogledajte i poglavlje vezano za **NUMA** arhitekturu i sabirnice koje ona koristi:
**10.7.2.2.1. NUMA I/O topologija.**

Kako smo već spomenuli sve opcije i parametri rada servisa *Irqbalance* definiraju se u datoteci: `/etc/sysconfig/irqbalance`. Sve ove opcije su opisane u uputama, koje možemo dobiti sa:

```
man irqbalance
```

Stoga pogledajmo neke od opcija servisa *irqbalance*, koje možemo postaviti u datoteku `/etc/sysconfig/irqbalance`. Pri tome su neke varijable već ugrađene, dok druge moramo navoditi kao argumente varijabli `IRQBALANCE_ARGS`:

- `IRQBALANCE_BANNED_CPULIST=`**`X`** – za definiranje CPU jezgri procesora na koje NE želimo da *Irqbalance* prebacuje hardverske signale prekida. Ovdje se CPU jezgre navode brojčano i to pojedinačno ili u nizu ili slijedu; poput primjerice: `1` ili `1,2,3` ili `1-5`). To je korisno kod visoko perfomantnih mrežnih kartica (**n** x **10+** Gbps).

  Servis *irqbalance* automatski izbjegava IRQ-ove na CPU jezgrama izoliranim putem `isolcpus=` kernel opcije.

  
  Za detalje o kernel opciji `isolcpus` pogledajte poglavlje: **10.7.2.4. Izolacija jezgri procesora .**
  Za **NUMA** arhitekturu i ograničavanje CPU jezgri, pogledajte: **10.7.2.2.2. NUMA CPU i RAM afinitet**.

- `--banirq= X` – za postavljanje hardverskih signala prekida za koje ne želimo da se balansiraju odnosno koriste od strane ovog servisa. Dakle ovo je implicitno isključivanje pojedinih IRQ-a od ovog servisa. Primjerice, ako želimo isključiti IRQ **27** i **29** od *irqbalance* servisa, tada dodajmo sljedeći redak na kraj navedene datoteke, kao *ARGS*:

```
IRQBALANCE_ARGS= --banirq=27 --banirq=29
```

- `--policyscript=<skripta>` – moguće je definirati i *shell* skriptu s varijablama i argumentima, koja će biti učitana za svaki novi IRQ prilikom inicijalizacije ovog servisa. Osim navedenih, dostupno je i desetak drugih opcija.

**Nakon obih promjena u konfiguraciji *irqbalance* servisa, potrebno ga je restartati:**

```
systemctl restart irqbalance
```

**Dodatno o hardverskim signalima prekida (IRQ)**
Za statistike ispisa upotrebe svih hardverskih signala prekida, možemo koristiti i dvije naredbe `lsirq` i `irqtop`, koje dolaze u softverskom paketu `util-linux`. Naredba `lsirq` daje nam popis hardverskih signala prekida (IRQ), ali ona može prikazivati i softverske signale prekida, ako koristimo prekidač `-S`. Pogledajmo primjer ispisa hardverskih signala prekida:

```
lsirq -s IRQ
IRQ  TOTAL      NAME
  8      0      IR-IO-APIC 8-edge rtc0
  9      4      IR-IO-APIC 9-fasteoi acpi
 16     29      IR-IO-APIC 16-fasteoi ehci_hcd:usb1
 27  16546      IR-PCI-MSI-0000:00:19.0 0-edge eno1
 29    136      IR-PCI-MSI-0000:00:02.0 0-edge i915
```

Pogledajmo i naredbu `irqtop`, koja nam statistike upotrebe hardverskih signala prekida prikazuje u stvarnom vremenu:

```
irqtop
        cpu0 cpu1 cpu2 cpu3 cpu4 cpu5 cpu6 cpu7
  %irq: 18.5 11.7 11.4  9.2  9.1  9.1  9.8 21.1
%delta: 13.3 14.8  6.7  3.7  0.7  7.4  3.7 49.6

IRQ       TOTAL        DELTA     NAME
CAL      141878            3     Function call interrupts
LOC      124356          109     Local timer interrupts
27        16807           22     IR-PCI-MSI-0000:00:19.0 0-edge eno1
26          280            0     IR-PCI-MSI-0000:00:14.0 0-edge xhci_hcd
29          136            0     IR-PCI-MSI-0000:00:02.0 0-edge i915
```

← Ovdje je osim sâmog osvježavanja u stvarnom vremenu, prema svakom hardverskom signalu prekida (IRQ) brojčano, dobro pratiti i vršnu preraspodjelu signala prekida prema procesorskim jezgrama, kako je vidljivo u prva tri gornja stupca.

**Izvor informacija: `man irqbalance`,`man lsirq`,`man irqtop`,`man ps`,`man systemctl`, `man service`,`man chkconfig`,`man proc`, (1576), (1582).**

#### 10.5.1.3. Servis *ksoftirqd i softverski signali prekida*

S*oftverski signali prekida* koriste se za iniciranje nekih događaja, koji mogu i ne moraju biti u direktnoj vezi s hardverskim signalima prekida (*IRQ*). Uzmimo za primjer rad mrežne kartice. Naime nakon što je mrežna kartica primila paket s mreže te ga preko **DMA** komunikacije snimila u RAM memoriju (u tzv. *ring buffer*), ona generira *hardverski signal prekida*. Taj *hardverski signal prekida* koji aktivira *interrupt handler* rutinu unutar upravljačkog programa mrežne kartice tada kreira *softverski signal prekida* prema prvoj komponenti mrežnog podsustava, s kojim traži njegovu pažnju (to može biti *NAPI* komponenta mrežnog podsustava). Tada sustav reagira na taj *softverski signal prekida* i traži od upravljačkog programa mrežne kartice da dohvati taj mrežni paket iz prstenaste međumemorije (*ring buffer*). Ovo je način dohvaćanja mrežnih paketa znan i kao *"Pooling*".
Dalje se u komunikaciji između raznih komponenti mrežnog podsustava, intenzivno koriste *softverski signali prekida (interrupti)*, svaki puta kada se traži pažnja odnosno obrada od neke od potkomponenti mrežnog sustava.
Zbog toga što se *softverski signali prekida* intenzivno koriste, i njihovo generiranje odnosno broj je znatno veći u odnosu na hardverske signale prekida. Međutim oni zahtijevaju i više CPU vremena, što je normalno. Stoga se u Linuxu, na svakoj CPU jezgri pokreće po jedan servis koji barata sa *softverskim signalima prekida*. Naziv tog servisa je `ksoftirqd`.
Servis `ksoftirqd` je prema načinu rada tzv. *Linux thread* odnosno servis koji radi na vrlo niskoj razini operativnog sustava, a pokreće ga (ih) `kthreadd` servis. Softverski *signali prekida* se često pozivaju nakon hardverskih, a pošto je njih moguće kreirati znatno brže, odnosno u većem broju nego hardverskih, postoji i mogućnost da ih se ne stigne sve obraditi u određenom vremenskom okviru. Ako se softverski *signal prekida* ponovi, jer prvi puta nije bio odrađen, tada kontrolu nad njim preuzima `ksoftirqd` koji se onda i dalje brine za njega sve dok se ne izvrši odnosno proslijedi na obradu te u konačnici i obradi. Vezano sa ***softverske*** *signale prekida*, njihove statistike se u stvarnom (realnom) vremenu zapisuju u datoteku: `/proc/softirqs`.

Pogledajte i poglavlje: **10.7.2.3.6. Naredba *mpstat* za CPU.**



U slučajevima kada ***ksoftirqd*** počne zauzimati sve više procesorskog vremena, to je znak kako je računalo pod velikim opterećenjem, jer ne stiže odrađivati sve softverske signale prekida na vrijeme.

Pogledajmo kako softverski *signali prekida* izgledaju na računalu s dvije CPU jezgre, pogledom na `/proc/softirqs`:

```
cat /proc/softirqs
                    CPU0       CPU1
          HI:          0          0
       TIMER:    9127483     424642
      NET_TX:       6529          1
      NET_RX:     374696       6777
       BLOCK:      54645       1411
BLOCK_IOPOLL:          0          0
     TASKLET:          1          5
       SCHED:     467764     356815
     HRTIMER:         75         47
         RCU:     249426     157217
```

Svaki stupac predstavlja jednu CPU jezgru (imamo ih sâmo dvije), a redovi predstavljaju[198] sljedeće:

- `HI` — softverski *signali prekida* visokog prioriteta (engl. *High Priority*).
- `TIMER` — vremenski okidači vezani za softverske *signale prekida.*
- `NET_TX` — softverski *signal prekida* vezani za slanje mrežnih paketa na mrežnu karticu.
- `NET_RX` — softverski *signali prekida* vezani za primanje mrežnih paketa sa mrežne kartice.
- `BLOCK` — softverski *signali prekida* vezani za operacije nad blok uređajima (diskovima).
- `BLOCK_IOPOLL` —softverski *signali prekida* vezani za operacije nad blok uređajima (diskovima), gdje je moguće prebacivanje na drugu CPU jezgru.
- `TASKLET` — daju nam mogućnost da se određene funkcionalnosti odgode za kasnije izvršavanje
- `SCHED` — predstavlja mogućnost da se pokreće barem jedan ***kthread*** proces po CPU jezgri te da se on i izvršava (za razne funkcionalnosti).
- `HRTIMER` — daje mogućnost da se određene funkcionalnosti, koje bi se trebala odraditi (izvršiti) nakon isteka vremenskog okvira (*timera)*, mogle prebaciti na drugu CPU jezgru.
- `RCU` — **RCU** funkcionalnost (engl. *Read-copy update*) je sinkronizacijski mehanizam pomoću kojeg se operacije čitanja, kopiranja i osvježavanja podataka mogu bolje skalirati na veći broj istovremenih procesa/programa i na više paralelnih CPU jezgri.

Iste statistike možemo dobiti i s novijom naredbom `lsirq` (koja dolazi u programskom paketu `util-linux`):

```
lsirq -S
     IRQ TOTAL NAME
     NET_RX 15973 network receive softirq
     NET_TX     3 network transmit softirq
```

**Izvori informacija: (196),(197),(198),(200),(K-5), `man 5 proc`,`man lsirq`.**

#### 10.5.1.3.1. Softverski signali prekida detaljnije

Važno je razumjeti da su sistemski signali također softverski signali prekida koji pružaju mehanizam za rukovanje asinkronim događajima na sustavu. Ti događaji mogu biti uzrokovani od strane određene hardverske komponente računala kao što je mrežna kartica, diskovni kontroler i slično, ali i potjecati od strane korisnika; primjerice kada korisnik pritisne kombinaciju tipki poput **Ctrl-C** ili iz aktivnosti unutar programa ili kernela i slično.
Mi ćemo se ovdje fokusirati na softverske signale prekida uzrokovane hardverskom komponentom poput mrežne kartice.
Već smo naučili da nakon što mrežni paket dođe do mrežne kartica, ona ga pomoću **DMA** (bez uplitanja procesora) pohrani u RAM memoriju odnosno u međumemoriju koja se naziva *ring buffer* to jest prstenasta memorija. Tek potom mrežna kartica kreira hardverski signal prekida (IRQ). Potom se ovaj IRQ prema svom broju zapisuje u statistiku u datoteku: `/proc/interrupts`. Na taj hardverski signal prekida reagira upravljački program mrežne kartice, a sve dalje ovisi o konkretnoj implementaciji. Međutim u većini implementacija nakon toga se koristi softverski signal prekida čak i ako nemamo **NAPI** upravljački program za mrežnu karticu.



*New API* (**NAPI**) poboljšanje je standardnog mehanizma za obradu mrežnih paketa unutar upravljačkog programa mrežne kartice koje drastično poboljšava učinkovitost dohvaćanja dolaznihh mrežnih paketa. Upotrebom *NAPI* mehanizma, upravljački program može koristiti *Pooling* način dohvaćanja mrežnih paketa umjesto da se koriste hardveski signali prekida za za svaki primljeni paket. *Pooling* način dohvaćanja (ali malo manje učinkovit od *NAPI*) mogu koristiti i upravljački programi koji ne koriste *NAPI*, pa i oni mogu nastaviti dohvaćati mrežne pakete pomoću softverskih signala prekida (*asinkrono*).

Softverski signal prekida predstavlja mehanizam obavještavanja kernela o dostupnosti novih mrežnih paketa unutar prstenaste međumemorije. Servis `ksoftirqd` tada dohvaća mrežne pakete iz prstenaste međumemorije te ju prazni pripremajući ju za primitak novih mrežnih paketa. Statistike softverskih signala prekida se zapisuju u datoteku: `/proc/softirqs`.
Softverske signale prekida vezane za mrežu možemo pratiti u stvarnom vremeno s nizom naredbi:

```
watch -n1 'egrep "CPU|NET_RX|NET_TX" /proc/softirqs'
```

Iako su softverski signali prekida učinkovitiji od hardverskih u dohvaćanju mrežnih paketa, ako *SoftIRQ* ne radi dovoljno dugo ili ne dohvati dovoljno veliki broj paketa, brzina dolaznih paketa mogla bi premašiti sposobnost sustava da ih dohvati i obradi. Kao rezultat toga, mrežna kartica može prepunjavati međumemoriju ili odbacivati pakete. U svakom slučaju tada se paketi gube. Srećom povećamo li tada vrijednosti parametara kernela (*sysctl*): `net.core.netdev_budget` i `net.core.netdev_budget_usecs`, moguće je kontrolirati vrijeme i broj paketa koje `ksoftirqd` može obraditi psvakom pojedinom ciklusu dohvaćanja mrežnih paketa. Da bismo utvrdili je li potrebno podešavanje parametra `net.core.netdev_budget`, potrebno je pregledati brojače u datoteci `/proc/net/softnet_stat` i to ovako:

```
awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
```

← Izvršavanjem ovog niza naredbi u jednom retku dobit ćemo decimalni ispis za svaku CPU jezgru u novom retku:

```
30377454    0   15      0   0   0   0   0   0   0   0   0   3   0   0
23305601    0   45780   0   0   0   0   0   0   0   0   0   4   0   0
57164027    0   7       0   0   0   0   0   0   0   0   0   5   0   0
```

U ovoj sažetoj i skraćenoj statistici je važno pratiti:

- (**1**) stupac – u njemu se nalazi broj primljenih mrežnih paketa.
- (**2**) stupac [***dropped***] – broj odbačenih mrežnih paketa. Paketi su obično odbačeni zbog prepunjavanja tzv. ***backlog*** međumemorije koja se definira (i može povećati) u: `net.core.netdev_max_backlog`. Ovu statistiku je moguće dobiti i s naredbom `nstat`:

```
nstat -az TcpExtTCPBacklogDrop
```

- (**3**) stupac [***time_squeeze***] – broj puta koliko `ksoftirqd` proces nije mogao dohvatiti sve pakete sa sučelja u jednom *NAPI* ciklusu dohvaćanja paketa. Isto vrijedi i za upravljačke programe koji ne koriste *NAPI*, ali također koriste *Pooling* mehanizam dohvaćanja mrežnih paketa. Rješenje je povećanje: `net.core.netdev_budget` i `net.core.netdev_budget_usecs`.
- (**9**) stupac [***received_rps***] – prijemni broj paketa za **RPS**.
- (**10**) stupac [***flow_limit_count***], (**11**) [***input_qlen*** + ***process_qlen***], (**12**) [***seq***→***index***], (**13**) [***input_qlen***] – ulazni *queue* i (**14**) [**process_qlen**] – *qlen* od procesa.



Za detalje ovih optimizacija pogledajte i poglavlje (cjelinu **NAPI i Pooling**):
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

**Izvori informacija: (1647),(1648),(1649),(K-4),(K-5),** `man 5 proc`,`man nstat`,`man sysctl`.

## 10.5.2. Pooling i IRQ te Interrupt moderation

### 10.5.2.1. Pooling i IRQ

Mrežne kartice, kao i drugi uređaji, poput primjerice disk kontrolera, kada im je potrebna pažnja *CPU-a*, odnosno kada od CPU-a traže procesorsko vrijeme za obradu njihovih zahtjeva, generiraju signal prekida. U slučaju mrežne kartice, obično kod svakog novog mrežnog paketa koji je zaprimljen, mrežna kartica kreira novi hardverski signal prekida. CPU tada zaustavlja trenutni proces (pokrenuti program) s obradom, to jest pauzira ga, privremeno pohranjuje međurezultate, prazni memoriju i registre, te preuzima na obradu zadatak od mrežne kartice. Odnosno preuzima na obradu taj konkretni mrežni paket. Za gigabitne brzine (1 Gbps) to znači 1.48 milijuna paketa u sekundi za pakete veličine 64 bajta, koje mrežna kartica mora primiti ili poslati.



Važno je razumjeti, da se broj signala prekida kod ovakvog načina rada, također mora popeti na isti broj.
Pri tome je: **1 mrežni paket = 1 hardverski *signal prekida (interrupt)***.

Srećom mehanizmi implementirani u ***Linux***, ***FreeBSD*** i druge moderne operativne sustave u slučajevima kada broj mrežnih paketa i sâmim time hardverskih *signala prekida* postaje prevelik, te bi zagušio cijeli operacijski sustav i sve programe, prelazi u drugačiji režim rada. Međutim on ovisi o tome podržava li ga uopće upravljački program kao i sâm hardver mrežne kartice.
U tom trenutku se prelazi na rad u kojem se više ne generiraju hardverski *signali prekida* za svaki paket, već se sustav počinje brinuti o tome da svako malo, s posebnim mehanizmima, sâm dohvaćao pakete iz međumemorije (*ring buffer*) mrežne kartice. Ovakav način rada se naziva mehanizmom dohvaćanja odnosno naziva se *engl. Polling* mehanizam.
Provjerimo je li naš kernel kompiliran s podrškom za *Pooling* način dohvaćanja mrežnih paketa iz međumemorije mrežne kartice:

```
grep  CONFIG_IRQ_POLL /boot/config-`uname -r`
```

Ako imamo podršku za *pooling* na razini našeg kernela, ovdje ćemo vidjeti: `=y`.
Upotrebom *pooling-a* sâm sustav se počinje brinuti i o tome kako će opsluživati ostale procese odnosno programe na najnižoj razini rada operativnog sustava. Kod boljih implementacija ovdje se pazi na tzv. ***Context switching*** na najbolji mogući način i time se dodatno dobiva na performansama, jer se CPU prebacuje na obradu između procesa, na način koji je optimalan i ne troši nepotrebno resurse na sâmo prebacivanje (tzv. *context switch*) između programa.

***U konačnici, upotrebom NAPI modela***:

- Izbjeglo se zagušenje cijelog sustava zbog generiranja prevelikog broja hardverskih *signala prekida*.
- Paketi koji se dohvačaju (***polling***) s mrežne kartice na obradu, mogu se dohvaćati u nîzu, i to više njih odjednom.
- Kod najgorih slučajeva, kada se tolika količina paketa ne može obraditi niti ovom metodom, sustav ih odmah odbacuje na sâmoj mrežnoj kartici, bez nepotrebnog dodatnog procesiranja od strane mrežnog stôga (*network stack)* odnosno dijela kernela operativnog sustava zaduženog za mrežu.

***Pooling način rada mrežne kartice je moguć i bez NAPI, ali nije toliko efikasan jer znatnije opterečuje procesor!***

Osim toga, sâm prijenos podataka od mrežne kartice prema CPU, može se odraditi na dva načina:

- Pomoću *programmed input/output* metode u kojoj se CPU brine za svaki prijenos podataka od i prema mrežnoj kartici. Ovo naravno nije najbolja metoda jer se opterećuje CPU za svaku aktivnost mrežne kartice.
- Pomoću *Direct memory access* (**DMA**) metode, u kojoj mrežna kartica može preko ***DMA*** kanala pristupati direktno RAM memoriji (do tzv. *ring buffera*), bez posredovanja *CPU-a*. Ovo je danas standardna metoda rada, ali zahtjeva dodatnu logiku unutar mrežne kartice. Sve modernije mrežne kartice podržavaju ovakav (***DMA***) način rada.

**Može i bolje!**
Kako smo vidjeli, na ***SMP*** sustavima (s više procesorskih jezgri), moguće je odrediti CPU afinitet signala prekida odnosno takozvani (*SMP) IRQ affinity,* na određenu grupu CPU jezgri. Međutim čak i mogućnošću preraspodjele signala prekida između CPU jezgri iz ove grupe, u konačnici će se uvijek koristiti isti signal prekida. To znači da će primjerice mrežna kartica iz prijašnjeg primjera (`eth0`) koja koristi signal prekida broj `54` uvijek koristiti isti (jedan jedini) signal prekida koji će ga eventualno malo obrađivati jedna, a malo neka druga CPU jezgra unutar grupe definiranih CPU jezgri. Čak i ako ovaj signal prekida (i prihvaćanje mrežnih paketa) uvijek obrađuje uvijek ista CPU jezgra, vrlo brzo će se dogoditi da će ista jezgra u nekom trenutku izvršavati i neki program ili servis, te će zbog prebacivanja konteksta (*context switching*) često dolaziti do usporavanja mreže. Za male brzine mreže (čak do 1Gbps) to nije neki problem, ali kod iole većih brzina (1Gbps+) ovo može postati i vrlo često postaje usko grlo. Rješenje postoji i u hardveru mrežnih kartica i u softveru. Naprednije fizičke mrežne kartice (i *paravirtualne* poput ***VirtIO***) podržavaju takozvani ***Multiqueue*** način rada, u kojem mrežna kartica podržava višestruke nîzove (engl. *queue*) za dohvaćanje mrežnih paketa. Pri tome svaki nîz koristi drugi signal prekida, a koji opet može obrađivati druga jezgra procesora.
Softverska rješenja koja koriste sličnu tehniku, ali sa softverskim signalima prekida su: **RPS** i **RFS**.



Vezano za ***Multiqueue*** značajku mrežnih kartica pogledajte sljedeće poglavlje te poglavlje:
**25.2.2. Multiqueue funkcionalnost.**

**Izvori informacija: (42),(199),(752),(200),(1378).**

### 10.5.2.1.1. Kako to (NAPI) radi, još malo detaljnije

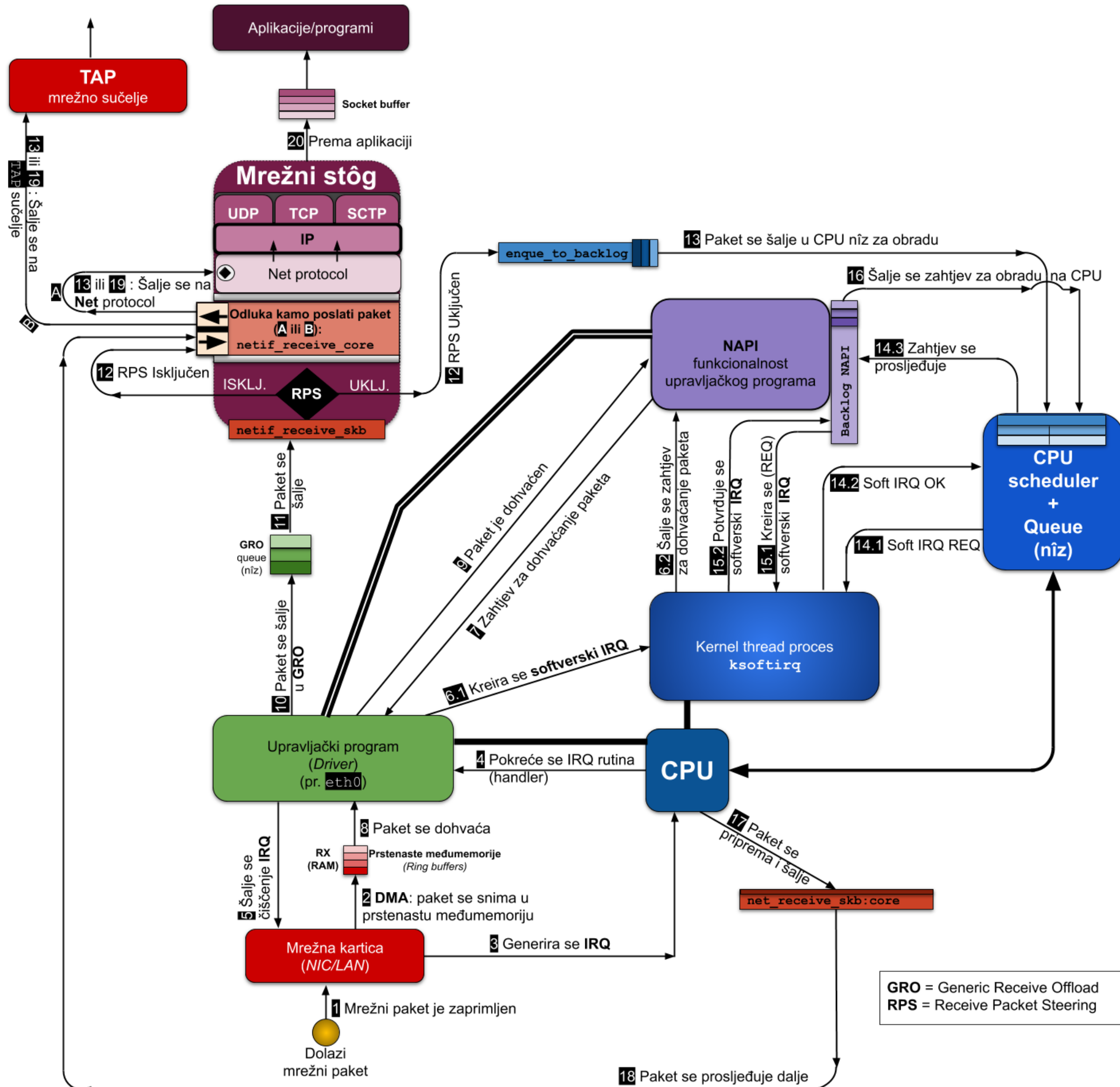
NAPI (engl. *New API*) je softverski okvir za razvoj upravljačkih programa mrežnih kartica, koji nam daje prethodno navedene napredne mogućnosti rada mrežnih kartica. Ako imamo mrežnu karticu i pripadajući *NAPI* upravljački program, ona se ponaša potpuno drugačije od mrežnih kartica koje koriste stare metode rada, što uzrokuje drastično brže dohvaćanje mrežnih paketa.
Pogledajmo što se događa, ako mrežna kartica i njen upravljački program podržavaju NAPI, kada se zaprimi prvi mrežni paket:

1. **NAPI** funkcionalnost unutar upravljačkog programa (tzv. ***kernel modula***) je inicijalizirana, ali je isključena.
2. Mrežna kartica paket upotrebom ***DMA*** prijenosa, a bez posredstva CPU-a, sprema u tzv. *ring buffer* međumemoriju.
3. Mrežna kartica generira hardverski signal prekida (**IRQ**) koji se aktivira unutar upravljačkog programa mrežne kartice.
4. Upravljački program aktivira *NAPI* funkcionalnost, slanjem **softverskog signala prekida** (**pogl. 10.5.1.3.**) a potom upravljački program aktivira ***Pooling*** mehanizam dohvaćanja mrežnih paketa u zasebnoj programskoj nîti kernela.
5. Upravljački program isključuje dalje slanje **hardverskih** signala prekida od strane mrežne kartice, kako bi potom primao sve mrežne pakete bez nepotrebnog kreiranja **hardverskih** signala prekida, a koji bi usporili rad računala odnosno CPUa. Od sada CPU odrađuje sve potrebno i sâm preuzima pakete (ovo je *Pooling metoda*) te provjerava je li došao novi nîz paketa koje treba preuzeti. Ako nije, odnosno ako je količina paketa premala, tada se ide na sljedeći korak.
6. Pošto/ako više nema velike količine paketa, upravljački program preko *NAPI-ja* ponovno uključuje **hardverske** signale prekida na mrežnoj kartici, a ona nastavlja raditi od točke **1**, kako je vidljivo na slici 52**.**

***Pooling način rada mrežne kartice je moguć i bez NAPI, ali nije toliko efikasan jer znatnije opterečuje procesor!***

Prema ***NAPI*** modelu, još detaljniji dijagram dohvaćanja mrežnih paketa je vidljiv na slici 52. dolje.

**Slika 52. Način rada NAPI mrežnog modela dohvaćanja i obrade mrežnih paketa.**




Kako provjeriti podržava li naš upravljački program (*kernel modul*) ***NAPI*** ? Primjerice za kernel modul imena `virtio_net`:

```
modinfo  virtio_net |  grep napi
```

← Ako se unutar upravljačkog programa unutar polja `parm:` nalazi ključna riječ `napi` to znači da podrška za *NAPI* postoji.

**Izvori informacija: (42),(199),(752),(200),(1378). Pogledajte i: 25.1.3.1. NAPI i Pooling.**

### 10.5.2.2. *Interrupt moderation*

U ovom naprednom poglavlju vidjet ćemo kako novije i snažnije mrežne kartice danas imaju i metodu ***Interrupt coalescing*** znanu i kao *interrupt moderation* mehanizam[43]. U slučaju upotrebe ovog mehanizma, uz pripadajući upravljački program, mrežna kartica ne generira hardverski signal prekida (***IRQ***) za svaki pojedini mrežni paket, već tek kada joj se međuspremnici napune većim brojem mrežnih paketa. Dakle signal prekida se generira za više paketa, a ne za svaki pojedinačno. Ova metoda značajno smanjuje broj signala prekida, te može ubrzati rad mreže. Stopa kojom mrežno sučelje (mrežna kartica) generira signale prekida prema procesoru može se kontrolirati podešavanjem parametra zvanog ***coalescing***. To se može postići pomoću naredbe `ethtool`, s prekidačem `-c` (engl. *Coalesce*). Pogledajmo našu `eth0` mrežnu karticu na sljedeći način (ispis je skraćen):

```
ethtool -c eth0

Coalesce parameters for eth0:
Adaptive RX: on  TX: on
stats-block-usecs: 0
sample-interval: 0
rx-usecs: 20
rx-frames: 5
rx-usecs-irq: 0
rx-frames-irq: 5
tx-usecs: 72
tx-frames: 53
tx-usecs-irq: 0
tx-frames-irq: 5
```

Objasnit ćemo opcije koje smo dobili u ispisu gore, u tablici:

| Opcija | Opis | Predložena vrijednost |
|---|---|---|
| **`adaptive-rx`** | Ovo je metoda dinamičke kontrole za smanjenje latencije za primljene pakete (RX) pri niskim brzinama paketa i povećanje propusnosti pri visokim brzinama paketa. | `On` |
| **`rx-usecs`** | Ovo je broj mikro sekundi (µs) koje se trebaju čekati prije podizanja hardverskog signala prekida (IRQ) za primljeni paket (RX), od trenutka kada je paket zaprimljen.<br>Kada je `rx-usecs` postavljen na `0` tada se koristi `rx-frames` vrijednost. | `5 – 10` |
| **`rx-frames`** | Ovo je broj mrežnih okvira (paketa) koji će se spremati u među memoriju mrežne kartice, prije nego se podigne hardverski signal prekida (IRQ), za primljene pakete (RX). | `1 – 250` |
| **`adaptive-tx`** | Ovo je metoda dinamičke kontrole za smanjenje latencije za slanje paketa (TX) pri niskim brzinama paketa i povećanje propusnosti pri visokim brzinama paketa. | `On` |
| **`tx-usecs`** | Ovo je broj mikro sekundi (µs) koje se trebaju čekati prije podizanja hardverskih signala prekida (IRQ) kada je mrežni paket poslan na mrežno sučelje (karticu); dakle za poslani paket (TX).<br>Kada je `tx-usecs` postavljen na `0` tada se koristi `tx-frames` vrijednost. | `5 – 10` |
| **`tx-frames`** | Ovo je broj mrežnih okvira (paketa) koji će se spremati u među memoriju mrežne kartice, prije nego se podigne hardverski signal prekida (IRQ), za poslane pakete (TX) prema mrežnom sučelju. | `1 – 250` |

Iz naše statistike iz prijašnjeg primjera za našu `eth0` mrežnu karticu, za primanje paketa (RX) vidimo sljedeće:

```
rx-usecs: 20
rx-frames: 5
```

To znači da se svakih `20` µs (mikro sekundi) ili za svakih `5` zaprimljenih mrežnih okvira (paketa) na mrežnoj kartici aktivira hardverski signal prekida (IRQ) i obavještava procesor na pristigle pakete.
Ako bi recimo ovo vrijeme (`rx-usecs`) u µs smanjili na 10, to bi značilo kako bi se hardverski signal prekida dvostruko češće aktivirao, ali i da bi se zbog toga dvostruko brže praznila **rx** međumemorija mrežne kartice *(rx ring buffer*).
U slučaju problema u prihvaćanju ili slanju mrežnih paketa, ovdje možemo napraviti optimizacije u nekoliko različitih smjerova:

- Povećati broj mrežnih okvira (paketa) koji će se spremati u među memoriju mrežne kartice, prije nego se podigne hardverski signal prekida ***IRQ*** (`rx-frames` i `tx-frames`) i samim time uvesti nešto veće kašnjenje odnosno latenciju u obradi paketa, ali smanjiti opterećenje CPU-a.
- Povećati broj mikro sekundi (µs) koje se trebaju čekati prije podizanja hardverskog signala prekida (IRQ) (`rx-usecs` i `tx-usecs`), s čime će također uvesti nešto veće kašnjenje odnosno latencija u obradi paketa, ali smanjiti opterećenje CPUa.
- Uključiti ***Pooling*** i/ili ***NAPI*** mehanizam (ako ga podržava hardver i upravljački program mrežne kartice).
- Ili čak smanjiti sva navedena vremena ili čak isključiti *Interrupt coalescing* s čime će se latencija (kašnjenje) smanjiti na najmanje moguće, ali će se opterećenje procesora (CPU-a) poprilično povećati.

Prema zadnjem navedenom primjeru odnosno prema nekim mjerenjima i preporukama[352],[353] u određenim ekstremnim situacijama, pogotovo kod mrežnih veza od 10Gbps ili znatno bržih (100+Gbps), kod kojih je vrlo važna mala latencija (kašnjenje), čak se preporuča isključiti ***Interrupt coalescing*** odnosno ovu mogućnost grupiranja više mrežnih paketa prije nego se za njih pošalje signal prekida. S druge strane za većinu primjena (također za brzine 10+Gbps) isključivanje nije preporučivo!

***Isključivanje Interrupt coalescing***
Ako želimo isključiti *Interrupt coalescing*, tada trebamo napraviti sljedeće:
```
ethtool -C eth0 rx-usecs 0 tx-usecs 0 rx-frames 1 tx-frames 1
```
S ovime smo sustavu naložili da se hardverski signal prekida generira kod svakog primljenog, ali i svakog poslanog mrežnog okvira (paketa), s čim smo smanjili latenciju primanja i slanja paketa, ali smo drastično više opteretili procesor (CPU).

Druga bolja opcija bi bila korištenje *POOLING* mehanizma, što zahtjeva mrežne kartice, ali i njihove upravljačke programe koji to moraju podržavati. Trajnu promjenu s kojom pozivamo naredbu `ethtool` tijekom pokretanja sustava, možemo zapisati u konfiguracijsku datoteku, koja postoji za svaku mrežnu karticu i s kojom se konfigurira svaka pojedina mrežna kartica na Linuxu. Ova datoteka (tzv. ***ifcfg***) je za `eth0` mrežnu karticu: `/etc/sysconfig/network-scripts/ifcfg-eth0`.
To bi postigli dodavanjem novog retka teksta, u gore navedenu datoteku:
```
ETHTOOL_OPTS="$ETHTOOL_OPTS ; -C eth0 rx-usecs 0 tx-usecs 0 rx-frames 1 tx-frames 1"
```


Za konfiguraciju ***ethtool*** opcija kod novijih sustava koji koriste ***NetworkManager*** (za **Red Hat 8+**), pogledajte: **26.1. Network Manager.**

**Adaptive moderation – dodatna metoda rada**
Postoji i još jedan mehanizam rada *Interrupt coalescing* koju također možemo koristiti.
Ovaj mehanizam se posebno primjenjuje za primanje (RX), a posebno za slanje mrežnih paketa (TX). Radi se o mehanizmu koji se zove adaptivna kontrola (engl. ***Adaptive moderation***) kod koje, ako ju uključimo (što je često i zadano), sustav stalno analizira veličinu mrežnih paketa i propusnost koju trenutno imamo, te prilagođava brzinu kreiranja signala prekida (`rx-usecs` i `tx-usecs`). Međutim ova funkcionalnost ne postiže uvijek najbolje rezultate kod velikih brzina mreža (10/25 Gbps+)!
Stoga je najbolje testirati njeno ponašanje (ili ju uopće ne koristiti).

Ova funkcionalnost se može uključiti na sljedeći način; za `eth0` mrežnu karticu, i za ***RX*** (dolazni) i ***TX*** (odlazni promet):
```
ethtool -C eth0 adaptive-rx on
ethtool -C eth0 adaptive-tx on
```

Odnosno ponovno ju za RX i TX prema potrebi možemo isključiti sa:
```
ethtool -C eth0 adaptive-rx off
ethtool -C eth0 adaptive-tx off
```

**Smanjenjem upotrebe *interrupt moderationa* (s manjim vrijednostima) povećavamo zauzeće procesora. Dok isključivanjem *interrupt moderationa* odnosno *interrupt coalescing* metode, smanjujemo latenciju, ali opterećujemo CPU (procesor).**

Ponekad je bolje koristiti ***Pooling/NAPI*** mehanizam koji iskorištava činjenicu da mrežni paketi dolaze u nîzu, pa se odmah nakon primitka prvog paketa generira signal prekida (IRQ), ali se odmah potom prelazi u *pooling* način rada u kojem se signali prekida više ne generiraju, već procesor (CPU) dohvaća cijeli nîz paketa koji su došli na mrežnu karticu:
- Ili ovisno o tome koliko ih je konfigurirano za dohvaćanje u *pooling* nîzu; primjerice 64.
- Ili unutar kojeg definiranog vremenskog razdoblja (**`tx-usecs`** ili **`rx-usecs`**).

Nakon toga ***NAPI*** se ponovno prebacuje u način rada sa signalima prekida (IRQ način rada) te sve kreće iz početka.


Pogledajte i poglavlje: **25.1 Linux mrežni model** te dodatne optimizacije mreže u poglavljima:
**25.5.6. Optimizacije mrežnih sučelja propusnosti veće od 10Gbps.**
**25.2. Dodatne mogućnosti linuxa.**
Za praćenje opterećenja sustava odnosno procesora (CPU) obradom signala prekida mrežne kartice, pogledajte poglavlje: **10.5.1.3 Servis *ksoftirqd i softverski signali prekida*** u kojem ćete vidjeti kako pratiti statistike korištenje softverskih signala prekida (*soft IRQ*) iz datoteke: `/proc/softirqs`, čijim statistikama, kao i statistikama standardnih hardverskih signala prekida (***IRQ***) se mogu primijetiti zagušenja sustava i/ili mreže.

Pod operativnim sustavom *Linux*, obje metode (***Pooling*** *i* ***Interrupt moderation***) implementirane su u novi mrežni *API* koji se zove ***NAPI***. Osim toga implementiran je i zaštitni mehanizam, koji se aktivira u slučajevima zagušenja sustava, poznatog pod nazivom ***Packet throttling***. On radi unutar upravljačkog programa (tzv. ***Network interface-level throttling***). Dok se *Throttling* na aplikacijskoj razini kontrolira primjerice pomoću naredbi `tc` ili `ethtool -C pkt-rate-high` **`XY`** (**`XY`** =*Mbps*).

*Kada je mrežna kartica potpuno zagušena, ona nove mrežne pakete neće niti pokušavati prosljeđivati dalje prema procesoru, uzrokujući dodatno zagušenje sustava, već će svi novo pristigli mrežni paketi tada biti* ***<u>odbačeni</u>*** *direktno od strane mrežne kartice, točnije od strane njenog upravljačkog programa odnosno mehanizma „****Network interface-level throttling“****.*

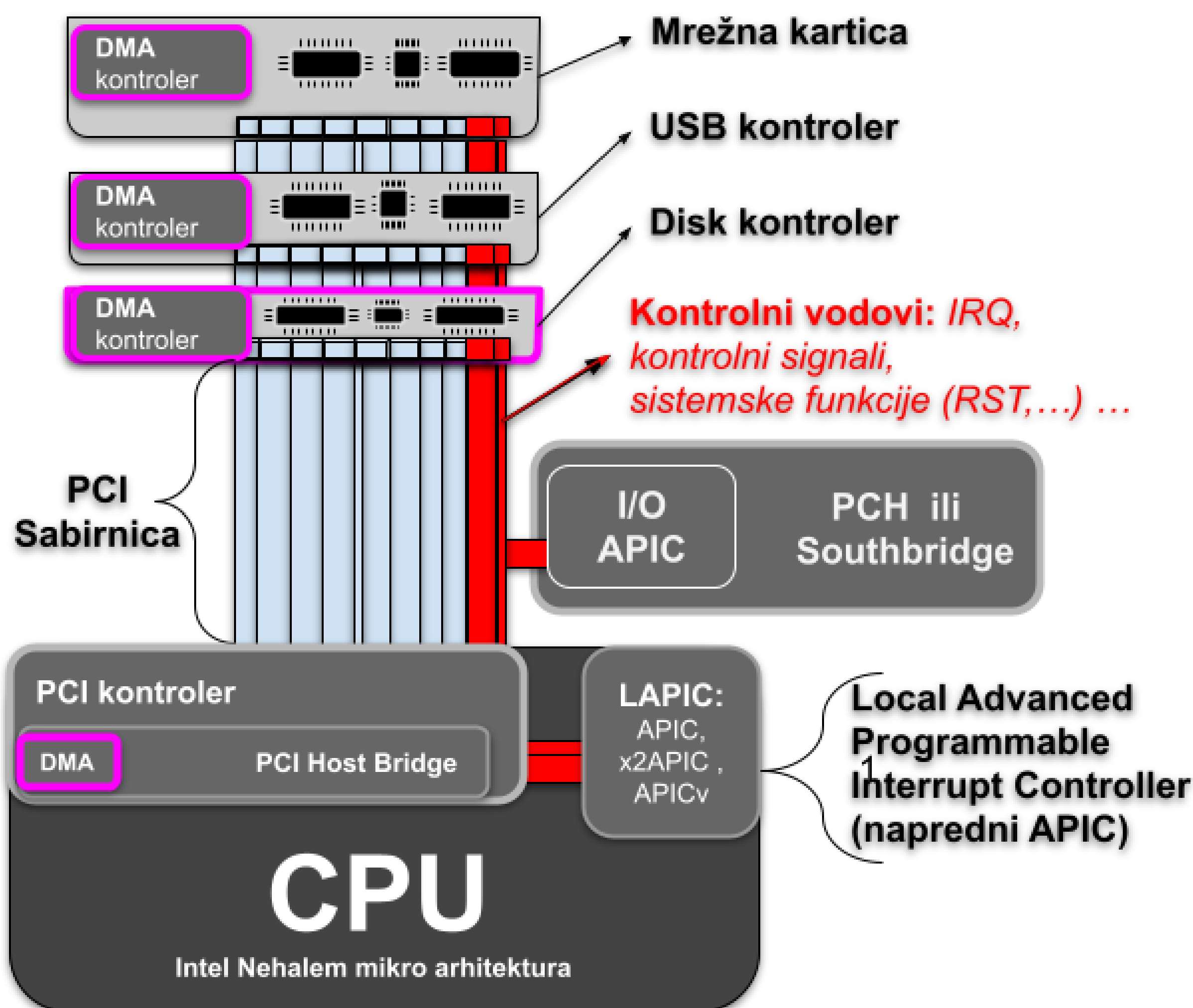

Za optimizacije mrežnih sučelja za velike propusnosti (*10+Gbps*), pogledajte poglavlje:
**25.5.6. Optimizacije mrežnih sučelja propusnosti veće od 10Gbps.**

**Izvori informacija: (43),(348),(349),(350),(351),(352),(353),(354),(1640),(1641),** **`man ethtool`**,**`man tc`**.

# 10.6. Direktan pristup memoriji – DMA

***Slijedi napredno poglavlje (10.6.x)!***
U ovom naprednom poglavlju objasnit ćemo što je metoda direktnog pristupa memoriji odnosno ***DMA*** (engl. *Direct Memory Access*) kako radi i za što se koristi. Naime kod klasičnoga korištenja signala prekida (*Interrupta*), u jednom trenutku, potreban je prijenos podataka od i prema memoriji sâmog uređaja, što predstavlja čitanje ili pisanje na taj uređaj, te dodatno i prijenos podataka od ili prema centralnoj RAM memoriji računala. Kod bilo koje od ovih operacija, potrebno je posredovanje CPU-a. Srećom, postoji i metoda koja može rasteretiti CPU od operacija pristupa RAM memoriji ili pristupa memoriji uređaja, odnosno omogućiti svakom uređaju direktan pristup RAM memoriji. Sâmim time se i drastično povećava brzina prijenosa podataka između (tog) uređaja i RAM memorije. Ova metoda se zove *Direct memory access* (***DMA***) odnosno metoda direktnog pristupa RAM memoriji bez potrebe za posredovanjem od strane *CPU*-a. Kod korištenja ***DMA***, ako ga uređaj podržava, sâm uređaj koji traži pristup procesoru (CPU) za obradu, prvo kreira hardverski signal prekida. Sustav tada privremeno zaustavlja pokrenuti proces (program) koji je na obradi odnosno događa se *context switch*. Potom se pokreće *interrupt handler* rutina (program) i kreće se s obradom (ovog) novog procesa odnosno sâmog zahtjeva uređaja koji je kreirao signal prekida. Ali od tog trenutka, preko *DMA* kanala, pristupa se RAM memoriji direktno bez posredovanja procesora, koji sada može nastaviti obradu podataka bez brige o pristupu RAM memoriji odnosno o operacijama čitanja ili pisanja u nju. Dakle sve sljedeće operacije rade se od strane uređaja koji upravo radi prijenos podataka u RAM memoriju, i to pomoću ***DMA*** pristupa, kako je vidljivo na slici 53.

**Slika 53. Logička arhitekturalna shema IRQ i DMA podsustava na PCI sabirnici.**

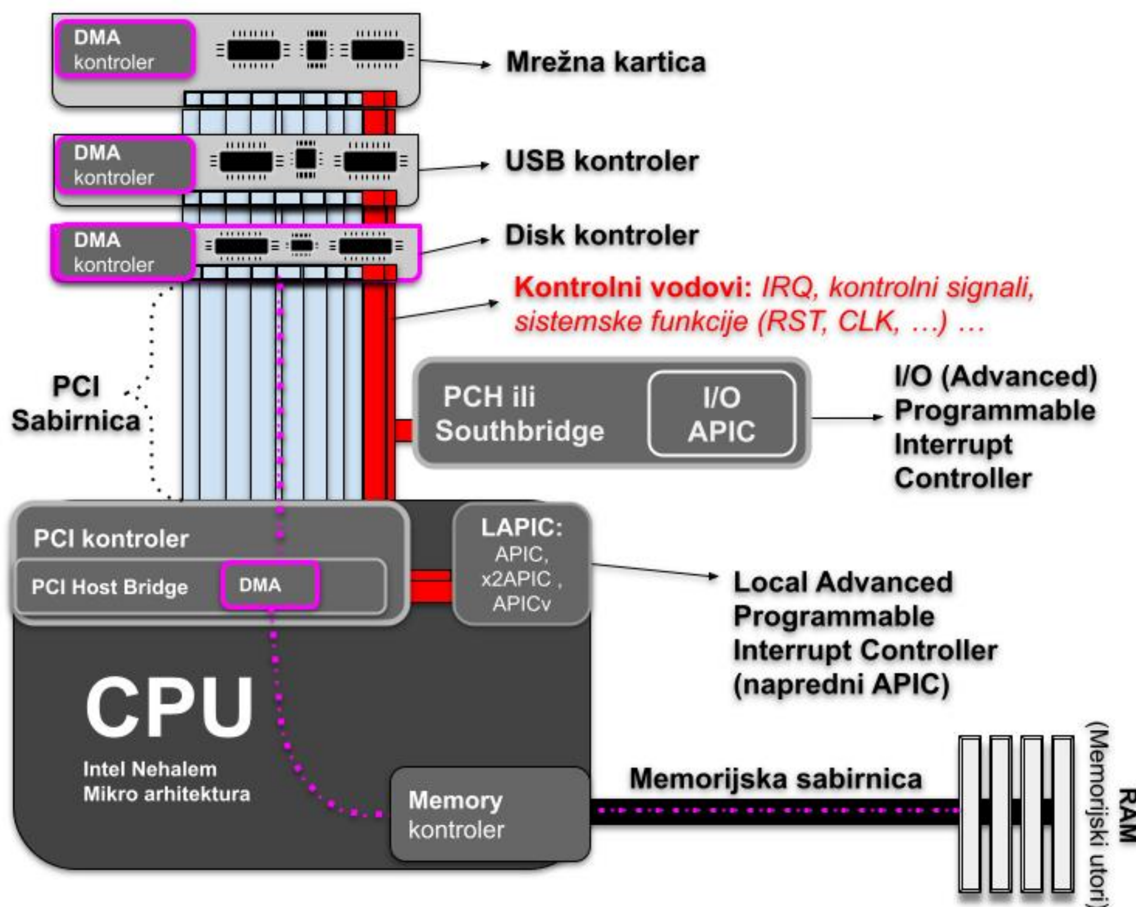



Dodatan problem kod klasičnoga rada (bez DMA) je bio i u tome što je procesor često čekao na neke ulazno izlazne operacije kao primjerice operacije prema disk kontroleru i sâmim time prema disku, grafičkoj ili mrežnoj kartici ili nekom drugom uređaju. Naime u trenutku kada se neki proces (program) obrađuje od strane CPU-a, a nakon primljenog hardverskog signala prekida, u životnom vijeku tog procesa (programa) konstantno se podaci moraju čitati ili zapisivati u RAM memoriju.
Tada je procesor morao čekati na uređaj koji je i inicirao signal prekida odnosno tražio svoje CPU vrijeme za obradu, kako bi ili iz njega ili prema njemu primao ili slao podatke. Ti podaci bi u konačnici završili u RAM memoriji pa je uvođenje ***DMA*** uvelike ubrzalo rad i obradu svakog programa (procesa).

Na kraju životnog vijeka ovog dijela procesa, DMA kontroler šalje signal za prekid kada su operacije prema RAM memoriji završile. Potom se nastavlja daljnja procedura identična za rad bez DMA odnosno pauziranje trenutnog procesa (programa) i prebacivanje (*context switch*) na novi proces odnosno program koji čeka u nîzu za obradu.

Zamislimo i sljedeću situaciju kada disk kontroler zatraži *DMA* komunikaciju: DISK → RAM memorija, kao na slici 54 dolje.

**Slika 54. Logička arhitekturalna shemu IRQ i DMA podsustava na PCI sabirnici: komunikacija**




Povijesno gledano, u originalnom **IBM PC XT** računalu, postojao je samo jedan DMA kontroler (*Intel 8237*) koji je imao 4 DMA kanala: DMA 0 — DMA 4, koji se koristio na starim **ISA** sabirnicama. Pojavom novije arhitekture ***IBM PC AT***, dodan je i drugi DMA kontroler (*Intel 8237*), tako se sada broj DMA kanala povećao na 8: DMA 0 — DMA 7. Pri tome je odabran DMA kanal 4 za međusobnu komunikaciju između ta dva DMA kontrolera. Do danas je zadržana ova kompatibilnost pa se oba *DMA* kontrolera nalaze integrirana unutar ***Super IO čipseta***, koji je zadužen za starije/sporije uređaje poput serijskih i paralelnih sučelja (portova), portova za miš i tipkovnicu i slično (*PS/2* i drugi).

Prema starijoj arhitekturi, ako se neki podaci trebaju čitati ili zapisivati na tvrdi disk, događa se sljedeće. Kada se računalo uključilo, disk kontroler je dobio svoj ***DMA*** kanal; primjerice ***DMA*** kanal broj pet (5) i tada se događa sljedeće:

- Disk kontroler je poslao procesoru hardverski signal prekida (*IRQ*) jer želi nešto čitati ili zapisivati na disk.
- Procesor pauzira trenutni proces (događa se ***context switch***) odnosno program koji je trenutno pokrenut.
- Disk kontroler traži pristup disku te pomoću DMA kontrolera, a preko DMA kanala broj 5 započinje prijenos podataka od ili prema RAM memoriji, bez posredovanja procesora, u posebnu regiju RAM memorije zaduženu za DMA prijenos.
- U međuvremenu procesor odrađuje sve ostale aktivnosti potrebne kako bi se ovaj proces odradio te čeka potvrdu od DMA kontrolera kada se prijenos podataka između diska i RAM memorije završio.
- Potom se kreira novi hardverski signal prekida (*IRQ*) kojim se potvrđuje da je sve završeno te se procesor vraća na izvršavanje prijašnjeg ili nekog drugog procesa odnosno programa koji čeka na izvršavanje: ponovno se događa *context switch*.

**PCI sabirnica i DMA**

DMA komunikacija na PCI sabirnici se odvija preko *DMA* kanala kojih ima ukupno osam. *DMA* kanali se ne mogu dijeliti između više uređaja poput *IRQ*-a već se svaki *DMA* kanal dodjeljuje pojedinom uređaju. Uređaji na **PCI** sabirnici sâmi mogu odabrati odnosno zauzeti prvi slobodan *DMA* kanal za komunikaciju, prema potrebi. Uređaj na PCI sabirnici (pr. mrežna kartica) putem (svog) DMA kontrolera inicira prijenos podataka između RAM memorije i svog internog međuspremnika (*buffer-a*). Važno je znati da je kod PCI sabirnice, kontroler za DMA prijenose integriran u svaki uređaj (karticu) dok je funkcija PCI *arbitra* integrirana u **PCI Host Bridge**. S obzirom na činjenicu da je PCI sabirnica dijeljena sabirnica (*shared bus*) to znači da više uređaja koristi istu komunikacijsku infrastrukturu unutar PCI sabirnice, pa samo jedan uređaj može prenositi podatke u određenom trenutku. Svaka kartica stoga prvo mora zatražiti kontrolu nad sabirnicom putem *arbitražnog* mehanizma. Taj mehanizam se zove "**PCI *Bus mastering***". *Bus Mastering* na PCI sabirnici omogućuje PCI uređajima (karticama) direktno upravljanje (kontrolu) nad sabirnicom bez intervencije centralnog procesora (CPU-a). Pri tome uređaj koji želi postati "*Bus Master*" prvo šalje zahtjev za kontrolu nad PCI sabirnicom. To signalizira *PCI Host Bridge-u* (*arbitru*) da određeni uređaj želi kontrolu nad sabirnicom. PCI *arbitar* analizira zahtjeve svih uređaja na sabirnici i kada je sabirnica slobodna, on odlučuje koji od njih će prvi dobiti kontrolu nad sabirnicom.

***Važno je znati da sâmo jedan uređaj može imati kontrolu nad PCI sabirnicom u bilo kojem trenutku.***

Potom se odabranom uređaju daje kontrola nad sabirnicom te on postaje tzv. *kontrolor sabirnice* (engl. *Bus Master)*. Tada on preko sabirnice i preko kontrolera sabirnice (***PCI Host Bridge***) komunicira s memorijskim kontrolerom (**MMU**), koji mu daje direktan pristup RAM memoriji bez posredovanja procesora. I od tada može krenuti s DMA prijenosom.

**PCI Express (PCIe) sabirnica i DMA prijenos**
Za razliku od ***PCI*** sabirnice, koja koristi zajedničku odnosno dijeljenu sabirnicu, ***PCIe*** koristi tzv. *point-to-point* vezu i serijsku komunikaciju između uređaja (kartica) na sabirnici i posebnog centralnog sklopa na vrhu hijerarhije *PCI Express* sabirnice koji se naziva ***Root Complex***. Naime uloga *Root Complex-a* jest povezivanje uređaja na *PCIe* sabirnici s procesorom i memorijskim podsustavom odnosno s RAM memorijom. U takvom radu (*point-to-point*) svaki *PCIe* uređaj ima svoju direktnu vezu s *Root complex-om* to jest korijenskim čvorištem *PCIe* sabirnice, omogućujući simultane prijenose podataka bez konkurencije između uređaja, kao što je bio slučaj kod PCI sabirnice.
Pogledajmo razlike u DMA načinu rada između *PCI* i *PCI Express* sabirnica:

| Značajka | PCI sabirnica | PCI Express sabirnica |
|---|---|---|
| **Propusnost** | Maksimalno 533 MB/s. | Skalabilna, od 1 GB/s (x1) do 121 GB/s (x16) na PCIe v.6.0. |
| **Potrebna arbitraža** | Da (***PCI Host Bridge***): dijeljena sabirnica. | Točkasto-točkasta veza (*bez konflikata ili dijeljenja sabirnice*) |
| **Adrese memorije** | Ograničeno na 32-bitne adrese. | Puni 64-bitni adresni prostor. |
| **Signali prekida (IRQ)** | Stari (hardverski) ili (rjeđe) MSI. | MSI i/il MSI-X (*bolja paralelizacija*). |
| **Scatter-Gather DMA** | Ograničeno. | Standardno podržano. |
| **IOMMU podrška** | Ograničeno (rijetko se koristi). | Standardno podržano. |

Kako provjeriti koji DMA kanali su u upotrebi na Linuxu (unutar datoteke: `/proc/dma` su popisani svi aktivni DMA kanali):

```
cat /proc/dma
 4: cascade
```

Vidljivo je kako je samo DMA kanal broj četiri aktivan. On je DMA kanal koji se koristi samo za komunikaciju između dva DMA kontrolera (unutar *Super IO* dijela čipseta). Međutim počevši od PCI sabirnice, DMA pristup se više ne oslanja na starinske DMA kontrolere (***ISA DMA***) jer se DMA u potpunosti implementira na razini samog uređaja (kartice) i kontrolera sabirnice, koristeći memorijski mapirane I/O (**MMIO**) registre i upravljanje putem upravljačkih programa i **IOMMU**. ***IOMMU*** upravlja pristupom memoriji za I/O uređaje, omogućavajući korištenje virtualnih adresa za **DMA**, koje se zatim mapiraju na fizičke adrese.

Za informacije o **DMA** i **IOMMU** (ako se koristi), pogledajte detalje za svaki ***PCI/PCIe*** uređaj na sabirnici s naredbom:

```
lspci -vvv
```



Za primjere upotrebe **DMA** pristupa pogledajte poglavlja:
**10.5.2.1.1. Kako to (NAPI) radi, još malo detaljnije.**
**10.6.1. Napredne DMA tehnologije.**
**10.7. CPU.**
**10.7.1.4. Ulazno-izlazni memorijski kontroler (IOMMU).**
**10.8. AHCI i ACPI.**
**12.3.5. Zone RAM memorije.**
**13.1.1. Podjela prema vrsti diskova.**
**13.2.1.7. Native Command Queuing (NCQ).**
**19.2.2. Mrežne kartice.**
**20.2. Usmjerivači koji rade na slojevima 3 i 4 te *Multilayer* preklopnici.**
**25.3. Statistike, analiza i praćenje mrežnih paketa.**
**25.5.4. *Scatter-gather*.**
**26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP + eBPF*).**

DMA za komunikaciju koriste uređaji koji zahtijevaju veliku brzinu komunikacije od ili prema RAM memoriji, poput:

- **Sâmih procesora** – primjerice DMA između jezgri procesora (tzv. *Intra-chip data transfer*) ili između fizičkih procesora; na poslužiteljskim sustavima s više fizičkih procesora (na **NUMA** arhitekturi računala).
- **Uređaji za pohranu** – DMA se koristi za prijenos podataka između RAM memorije i hard diskova, ***SSD*** ili ***NVMe*** diskova, optičkih pogona i drugih uređaja za pohranu. To omogućuje brže i efikasnije prijenose podataka u usporedbi s tradicionalnim tehnikama programiranog I/O-a (**PIO**) prijenosa podataka koji uključuju procesor.
- **Mrežne kartice** – DMA se koristi za prijenos podataka između RAM-a i mrežnih kartica. To omogućuje brze prijenose podataka preko mreže bez opterećivanja procesora.
- **Grafičke kartice** – DMA se koristi za prijenos podataka između RAM-a i grafičkih kartica. To omogućuje brzu obradu grafike i videa bez opterećivanja procesora.
- **USB uređaji** – DMA se može koristiti za prijenos podataka između RAM-a i USB uređaja, poput USB *stikova*, vanjskih USB hard diskova i skenera. To omogućuje brže prijenose podataka i manje opterećenje procesora.
- **Zvučne kartice** – DMA se koristi za prijenos podataka između RAM-a i zvučnih kartica. To omogućuje reprodukciju i snimanje zvuka bez opterećivanja procesora.

**Izvori informacija: (28),(87),(1773),(K-5),(K-9).**

### 10.6.1. Napredne DMA tehnologije

Jedna od naprednih DMA tehnologija, a koja je za sada dostupna samo na ***Intelovim*** snažnijim *Xeon* procesorima je tehnologija ubrzavanja rada DMA komunikacije između periferije (uglavnom mrežnih kartica), RAM memorije i procesora.
Ova tehnologija se naziva: ***I/O Acceleration Technology*** (*IOAT*). Naime ako postoji podrška za ovu tehnologiju na strani procesora i matične ploče te hardvera, recimo mrežne kartice, tada se sve kopiranje podataka koje bi se događalo u normalnoj DMA komunikaciji ubrzava. To radi tako što sada hardver (pr. mrežna kartica) ubrzano može komunicirati direktno s procesorom pomoću ove funkcionalnosti.
Ovaj ubrzani DMA mehanizam koristi se za mrežne kartice, ali i kod određenih disk/RAID kontrolera. Podrška za ***Intel IOAT*** mehanizam uključena je prvi puta u Linux kernel 2.6.18, ali je kasnije onemogućena u inačici 3.13.11.10 zbog nekih nesretnih *bugova* koji su uzrokovali korupciju podataka. U novijim kernelima 4.x ova funkcionalnost je znatnije optimizirana, mada je i dalje preporuka još ne koristiti ovu tehnologiju koja će vjerojatno biti optimizirana u nekoj od inačica kernela 5.10x ili novijih.

## ***Direct cache access*** (*DCA*)

Još jedna zanimljiva funkcionalnost uključena u paket s *Intel I/O AT* je i takozvani ***Direct Cache Access*** (***DCA***).
Ova značajka omogućuje mrežnim uređajima (preko njihovih upravljačkih programa) izravno postavljanje mrežnih podataka u predmemoriju CPUa. Kako to točno funkcionira, ovisi o konkretnoj implementaciji unutar upravljačkog programa mrežne kartice (kernel modula). Da biste koristili *DCA*, trebat ćete osigurati da je *DCA* omogućen u vašem BIOS-u, te da imate kernel modul `dca` kao i da vaša mrežna kartica i upravljački program podržavaju *DCA.*

Prvo moramo vidjeti je li naš kernel kompiliran s podrškom za ***ioat*** i ***dca*** *te* ***DMA*** (slijedi niz naredbi u jednom retku):

```
egrep -i „CONFIG_DCA=|CONFIG_INTEL_IOATDMA=|CONFIG_DMA_ENGINE=|CONFIG_DMADEVICES=“
/boot/config-`uname -r`

CONFIG_DMADEVICES=y
CONFIG_INTEL_IOATDMA=m
CONFIG_DMA_ENGINE=y
CONFIG_DCA=m
```

Vidimo da za obje funkcionalnosti imamo dostupne kernel module (*=m*), a ostale važne DMA funkcionalnosti su integrirane u kernel (*=y*). Sada pogledajmo je li ***ioat*** aktiviran u trenutku pokretanja sustava odnosno inicijalizacije kernela, upotrebom naredbe `dmesg` i malo filtriranja s naredbom `grep` na sljedeći način:

```
dmesg | grep ioat

[    5.360393] ioatdma: Intel(R) QuickData Technology Driver 4.00
[    5.360683] ioatdma 0000:00:04.0: irq 60 for MSI/MSI-X
[    5.361129] ioatdma 0000:00:04.1: irq 62 for MSI/MSI-X
[    5.361470] ioatdma 0000:00:04.2: irq 63 for MSI/MSI-X
[    5.361794] ioatdma 0000:00:04.3: irq 64 for MSI/MSI-X
[    5.362124] ioatdma 0000:00:04.4: irq 65 for MSI/MSI-X
```

Vidljivo je kako su uređaji koji koriste IRQ: `60,62,63,64` i `65` sposobni koristiti ***IOAT***, što se zapravo odnosi na DMA kontrolere unutar CPUa: vidljivo je kao „*`Intel Corporation Xeon E5/Core i7 DMA Channel 0/1/2/3/4`*“. To je jasnije, ako ih se pogleda s naredbom `lspci -vv`, iz koje se točno vidi kako ovi DMA kontroleri za navedene DMA kanale koriste *ioatdma* kernel modul za svoj rad. To znači kako imamo i učitane potrebne kernel module: `ioatdma` i `dca`

```
lsmod | grep  ioatdma
ioatdma                67758  0
dca                    15130  1 ioatdma
```

Koristili smo naredbu `lsmod` za listu kernel modula. Vidimo kako kernel modul: `dca` koristi: `ioatdma` kernel moduli, što je uredu.

Sada pogledajmo koristi li se ***DCA*** (koji mora biti uključen i u *BIOSu*, kao i *IOAT*):

```
dmesg | grep dca
[    5.222379] dca service started, version 1.12.1
```

Aktivan je i on, jer bi u suprotnom dobili poruku poput:

```
ioatdma 0000:00:08.0: DCA is disabled in BIOS
```

Na sljedeći način možemo provjeriti koristi li se IOAT DMA za neki od uređaja, preko bilo kojeg od IOAT DMA dostupnih DMA kanala:

```
cat /sys/class/dma/dma*/memcpy_count
```

Kako bismo vidjeli koliko podataka je preneseno pomoću IOAT DMA kanala, to možemo postići sa sljedećom naredbom:

```
cat /sys/class/dma/dma*/bytes_transferred
```

Ako imamo sve nule (0), to znači kako nema prijenosa podataka preko IOAT DMA kanala.
Sada nam je još potreban upravljački program za primjerice mrežnu karticu (uz samu naprednu mrežnu karticu koja to podržava), koji je kompiliran s podrškom za *ioatdma* i *dca*.

Određeni testovi upotrebe IOAT DMA i DCA na 10Gbps mrežama govore sljedeće (slika 55).

**Slika 55. Rezultati mjerenja propusnosti memorije (IOAT DMA i DCA)**

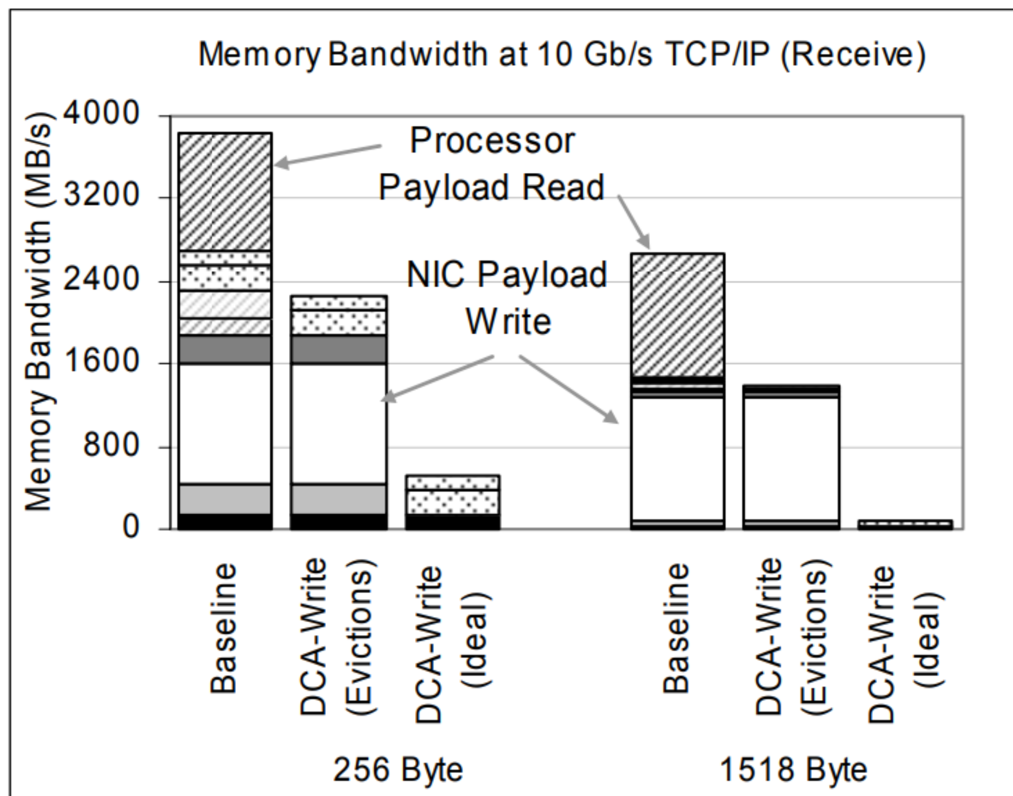



**Figure 4. Memory Bandwidth Reduction due to DCA Options (Receive Side, TCB, descriptor and header accesses not labeled)**

Izvor informacije odnosno ovih mjerenja je dokument: Direct Cache Access for High Bandwidth Network I/O: https://web.stanford.edu/group/comparch/papers/huggahalli05.pdf

Slika 55. prikazuje uštede na cjelokupnoj propusnosti memorije, ako se koristi (ili ne) ***DCA*** metoda rada u konkretnom primjeru upotrebe mrežne kartice.
Vidljivo je koliko se u gornjem dijelu grafa nadodaju (*Processor Payload Read*) CPU operacije koje naravno smanjuju efektivnu propusnost memorije i indirektno propusnost mreže. U lijevom dijelu vidimo utjecaj kada se koriste mrežni paketi veličine 256 bajta, a u desnom standardni (veći) paketi veličine 1518 bajta.

**Optimizacija IOAT DMA**

***IOAT DMA*** prijenos (***IOAT DMA Engine***) se koristi sâmo kada je veličina paketa iznad određenog praga. Taj se prag naziva *copybreak*. Ova provjera je u upotrebi jer se za kopiranje male količine odnosno malih blokova podataka, korištenja IOAT DMA mehanizma ne isplati odnosno neće se ubrzani prijenos.
Ova vrijednost je obično postavljena na 4096, a možemo ju mijenjati; u `/proc` direktoriju (mapi) na lokaciji: `/proc/sys/net/ipv4/tcp_dma_copybreak`

To je moguće promijeniti i sa `sysctl` varijablom: `net.ipv4.tcp_dma_copybreak`.

**Izvori informacija: (460),(461),(462),(463),(464),** **`man dmesg`**, **`man lsmod`**, **`man 5 sysfs`**, **`man proc`**.

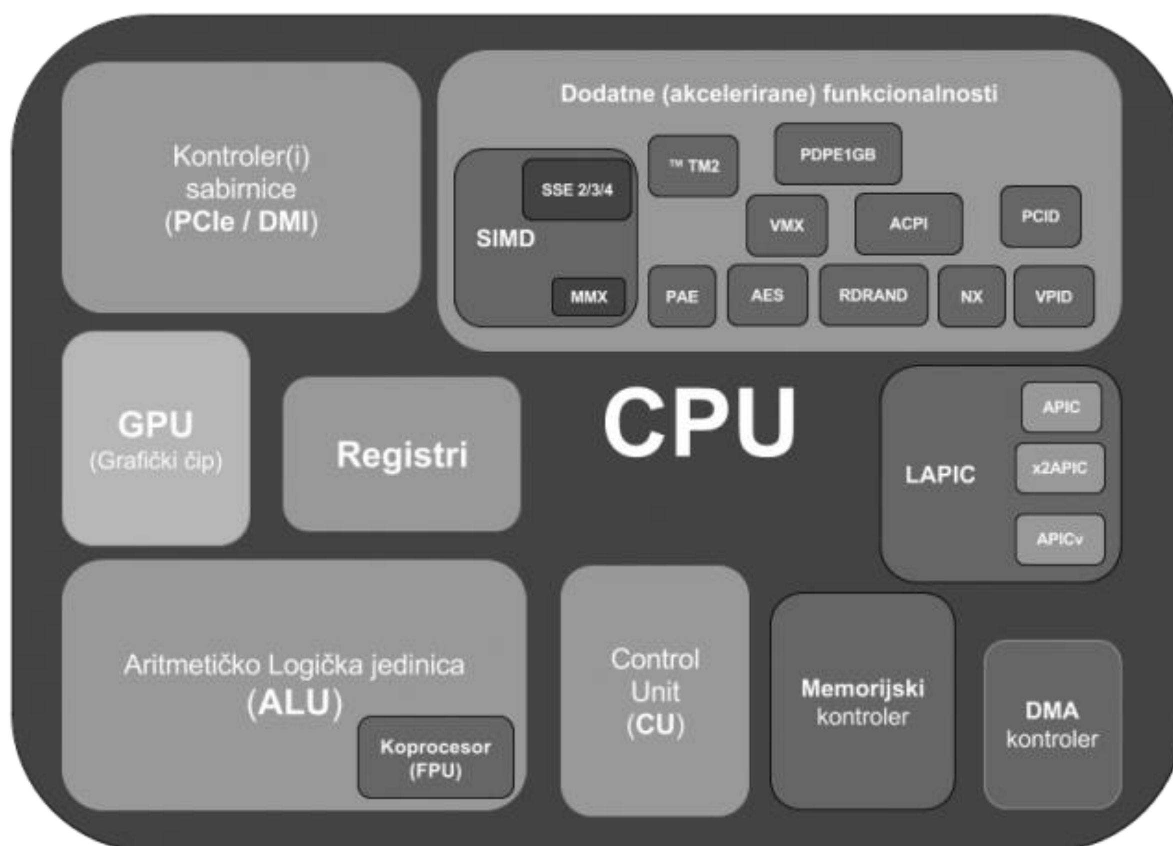
# 10.7. CPU

***Slijedi napredno poglavlje (10.7.x)!***
U ovom naprednom poglavlju detaljno ćemo analizirati rad centralnog procesora koji se naziva *CPU* (engl. *Central Processing Unit*). Kao što smo već prije naglasili, CPU odrađuje sve:

- Aritmetičke operacije poput: zbrajanja, oduzimanja, množenja, dijeljenja te drugih matematičkih operacija.
- Logičke operacije poput logičkih: **I** (AND), **ILI** (OR), **NE** (NOT), **NI** (NAND), **NILI** (NOR), **XILI** (XOR), **XNILI** (XNOR) i sličnih.
- Kontrolne operacije, za koje je zadužen dio koji se zove ***Control unit*** (**CO**) a koji na osnovi programskih instrukcija nalaže svim ostalim jedinicama: (aritmetičko-logičkoj, ulazno/izlaznoj i memoriji) što i kako da "odrade".
- Ulazno/izlazne operacije prema drugim komponentama računala.

Pogledajmo i blok dijagram modernih procesora na slici **56**.

**Slika 56. Logički blok dijagram procesora.**




Nadalje, na svim novijim generacijama procesora integrirane su i mnoge druge komponente poput:

- Memorijskog kontrolera za pristup RAM memoriji i svih njegovih komponenti, poput ***MMU dijela***.
- ***DMA*** kontrolera za direktan pristup memoriji, kao i kontrolera za signale prekida (***IRQ***).
- Ali i cijeli nîz dodatnih komponenti i procesorskih instrukcija koje ubrzavaju određene operacije:
    - Od matematičkih operacija koje su integrirane u **FPU** (Engl *Floating Point Unit*) tj. dijelu procesora koji odrađuje matematičke operacije s pomičnim zarezom poput; zbrajanja, oduzimanja, množenja, dijeljenja, korjenovanja, eksponencijalnih funkcija, logaritme, trigonometrijske funkcije i slično.
    - **SIMD** (engl. *Single instruction, multiple data*) skup instrukcija unutar kojih se nalazi i:
        - **MMX** nîz instrukcija zaduženih za multimedijalne zadatke (obrade slika i zvuka i sl.), ali samo s cjelobrojnim operacijama (bez "pomičnog zareza").
        - **SSE** (engl. *Streaming SIMD Extensions*) instrukcije (njih oko 70), koje su zadužene za digitalnu obradu signala i obradu grafike. Uslijedile su i dodatne funkcionalnosti koje su uvedene sa SSE2, SSE3, SSSE3 i SSE4. SSE je uvelo i računanje s pomičnim zarezom (decimalnim brojevima).
    - Podršku za virtualizaciju ***VT-x** (Intel)*, ***AMD-V** (AMD)*, …
    - Podršku za generator slučajnih brojeva (engl. *Random Number Generator*) (***RDRAND***).
    - Podršku za kriptiranje/dekriptiranje upotrebom ***AES*** algoritma (***AES-NI/AES***) i drugo.
- Ponekad i grafičkog integriranog sklopa (grafičke kartice).

Na ovaj način, implementacijom navedenih funkcionalnosti unutar samog procesora značajno su se ubrzale mnoge operacije koje je do tada morao odrađivati sam procesor (CPU). Ove funkcionalnosti promatrajmo kao da su zasebni integrirani sklopovi (*chipovi*) ugrađeni unutar sâ mog procesora, što zapravo i jesu.



Sve navedene funkcionalnosti su u Linuxu vidljive kao takozvane CPU zastavice "engl. ***CPU flags***".

Pogledajte listu većine tih funkcionalnosti (i/ili skupa instrukcija), uz vidljivi naziv takozvanih CPU zastavica pod *Linuxom*:

| Flag (zastavica) | Opis funkcionalnosti | Flag (zastavica) | Opis funkcionalnosti |
|---|---|---|---|
| 3dnow | 3Dnow! (AMD vektorske instrukcije, poput Intelovih **SSE1**) | mtrr | Memory Type Range Registers |
| 3dnowext | AMD 3Dnow! Extensions | nodeid_msr | NodeId MSR |
| 3dnowprefetch | 3Dnow prefetch instrukcije | nonstop_tsc_s3 | TSC doesn't stop in S3 state |
| abm | Advanced bit manipulation | nonstop_tsc | TSC does not stop in C states |
| acpi | ACPI via MSR (praćenje temperature i brzine rada CPUa) | nopl | NOPL (0F 1F) instrukcije |
| adx | ADCX i ADOX instrukcije | npt | AMD Nested Page Table support |
| aes / aes-ni | Hardverska podrška za **AES** (AES instrukcije) | nrip_save | AMD SVM next_rip save |
| amd_dcm | Multi-node processor | nx | Execute Disable |
| apic | Onboard APIC | osvw | OS Visible Workaround |
| arch_perfmon | Intel Architectural PerfMon | pae | Physical Address Extensions (Podrška za više od 4GB of RAM) |
| avx | Advanced Vector Extensions | pat | Page Attribute Table |
| avx2 | Proširenje AXV na 256 bitova. | pausefilter | AMD filtered pause intercept |
| AVX512**xy** | Proširenje AVX na 512bitova. | pbe | Pending Break Enable |
| bmi1 | 1st group bit manipulation extensions | pcid | Process Context Identifiers |
| bmi2 | 2nd group bit manipulation extensions | pclmulqdq | PCLMULQDQ instruction (carry-less multiplication — accelerator for GCM) |
| | | | |
| **Flag (zastavica)** | **Opis funkcionalnosti** | **Flag (zastavica)** | **Opis funkcionalnosti** |
| bts | Branch Trace Store | pdcm | Performance Capabilities |
| centaur_mcr | Centaur MCRs (= MTRRs) | pdpe1gb | 1GB memory pages |
| cid | Context ID | pebs | Precise-Event Based Sampling |
| clflush | CLFLUSH instrukcije | perfctr_core | Core performance counter extensions |
| cmov | CMOV instructions (conditional move) (također i FCMOV) | perfctr_l2 | L2 performance counter extensions |
| cmp_legacy | If yes ***HyperThreading*** not valid | perfctr_nb | NB performance counter extensions |
| constant_tsc | TSC ticks at a constant rate | pfthreshold | AMD pause filter threshold |
| cr8_legacy | CR8 in 32-bit mode | pge | Page Global Enable (global bit in PDEs and PTEs) |
| cx16 | CMPXCHG16B | pni | SSE-3 ("Prescott New Instructions") |
| cx8 | CMPXCHG8 instruction (64-bit compare-and-swap) | pn | Processor serial number |
| cxmmx | Cyrix MMX extensions | popcnt | **POPCNT** instruction (Hamming weight, i.e. bit count) |
| cyrix_arr | Cyrix ARRs (= MTRRs) | pse36 | 36-bit PSEs (4MB huge memory pages) |
| dca | Direct Cache Access | pse | Page Size Extensions (for 2MB memory pages) |
| decodeassists | AMD Decode Assists support | rdrand ili rng | RDRAND instrukcije (hardverski generator slučajnih brojeva) |
| de | Debugging Extensions (CR4.DE) | rdseed | RDSEED instrukcije vezane za rdrand |

| Flag (zastavica) | Opis funkcionalnosti | Flag (zastavica) | Opis funkcionalnosti |
|---|---|---|---|
| ds_cpl | CPL Qual. Debug Store | rdtscp | RDTSCP Read Time-Stamp Counter and Processor ID |
| dtes64 | 64-bit Debug Store | rep_good | rep microcode works well |
| dts | Debug Store (buffer for debugging and profiling instructions) | rtm | Restricted Transactional Memory |
| ept | Intel Extended Page Table ***[Second Level Address Translation (SLAT)]*** | rvi | AMD Rapid Virtualization Indexing [***Second Level Address Translation (SLAT)***] |
| erms | Enhanced REP MOVSB/STOSB | skinit | SKINIT/STGI instrukcije |
| est | Enhanced SpeedStep | smap | Supervisor Mode Access Prevention |
| extapic | Extended APIC space | smep | Supervisor Mode Execution Protection |
| extd_apicid | has extended APICID (8 bits) | smx | Safer mode — TXT (TPM support) |
| f16c | 16-bit fp conversions (CVT16) | ss | CPU self snoop |
| flexpriority | Intel FlexPriority | sse2 | SSE2 instrukcije |
| flushbyasid | AMD flush-by-ASID support | sse4_1 | SSE-4.1 instrukcije |
| fma4 | 4 operands MAC instrukcije | sse4_2 | SSE-4.2 instrukcije |
| fma | Fused multiply-add | sse4a | SSE-4A instrukcije |
| | | sse | Intel SSE vektorske instrukcije |

| Flag (zastavica) | Opis funkcionalnosti | Flag (zastavica) | Opis funkcionalnosti |
|---|---|---|---|
| fpu | Onboard FPU (floating point support) | ssse3 | Supplemental SSE-3 |
| fsgsbase | {RD/WR}{FS/GS}BASE instrukcije | svm_lock | AMD SVM locking MSR |
| fxsr | FXSAVE/FXRSTOR, CR4.OSFXSR | svm | Podrška za hardversku virtualizaciju ***AMD-v (AMD)*** |
| fxsr_opt | FXSAVE/FXRSTOR optimizacije | syscall | SYSCALL/SYSRET |
| hle | Hardware Lock Elision | tbm | trailing bit manipulations |
| ht | Podrška za Hyper-Threading | tce | translation cache extension |
| hypervisor | Pokrenut je na hipervizoru | tm2 | Thermal Monitor 2 |
| ia64 | IA-64 processor (not to be confused with x86-64 = amd64, indicated by lm) | tm | Automatic clock control (Thermal Monitor) |
| ibs | Instruction Based Sampling | topoext | topology extensions CPUID leafs |
| invpcid | Invalidate Processor Context ID | tpr_shadow | Intel TPR Shadow |
| k6_mtrr | AMD K6 nonstandard MTRRs | tsc_deadline_timer | Tsc deadline timer |
| lahf_lm | Load AH from Flags (LAHF) and Store AH into Flags (SAHF) in long mode | tsc_reliable | TSC is known to be reliable |
| lbrv | AMD LBR Virtualization support | tsc_scale | Podrška za AMD TSC skaliranje |
| lm | Long Mode (x86-64) — amd64 (Poznat i kao ***Intel 64***, tj. 64-bitni) | tsc | *Time Stamp Counter* (RDTSC) |

| Flag (zastavica) | Opis funkcionalnosti | Flag (zastavica) | Opis funkcionalnosti |
|---|---|---|---|
| lwp | Light Weight Profiling | up | SMP kernel running on up |
| mca | Machine Check Architecture | vmcb_clean | AMD VMCB clean bits support |
| mce | Machine Check Exception | vme | Virtual Mode Extensions (8086 mode) |
| misalignsse | Misaligned SSE mode | vmx | Podrška za hardversku virtualizaciju. ***VT-x (Intel)*** |
| mmxext | AMD MMX extensions | vnmi | Intel Virtual NMI |
| mmx | Multimedia Extensions | vpid | Intel Virtual Processor ID |
| monitor | Monitor/Mwait support (Intel SSE3 supplements) | wdt | Watchdog timer |
| movbe | Move to/from memory with byte order swap. | x2apic | x2APIC |
| mp | Multiprocessing mogućnost. | Xop | Proširene AVX instrukcije |
| mpx | Memory Protection ekstenzija | xsave | XSAVE/XRSTOR/XSETBV/XGETBV |
| msr | Model-Specific Registers (RDMSR, WRMSR) | xtopology | cpu topology enum ekstenzija |

Sada pogledajmo kako provjeriti koju od navedenih funkcionalnosti podržana naš procesor. U posebnom direktoriju `/proc/` se nalazi datoteka `cpuinfo` koja sadrži sve informacije o našem procesoru (CPU). Ispišimo ju te pogledajmo sve što je navedeno pod poljem: `flags`. Pogledajmo jedan *Intel Xeon* procesor (*E5-2650 v3* sa 40 CPU jezgri).
Iz primjera je izbačeno sve što nam trenutno nije zanimljivo:

```
cat /proc/cpuinfo

vendor_id       : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
cpu MHz         : 2300.210
cache size      : 25600 KB
fpu             : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf
cpuid_faulting pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16
xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx
f16c rdrand lahf_lm abm ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority ept
vpid fsgsbase bmi1 avx2 smep bmi2 erms invpcid xsaveopt
```

Vezano za zastavice, vidimo sve hardverske funkcionalnosti koje su ugrađene u naš konkretni procesor.

Više hardverskih informacija o našem procesoru, možemo dobiti pomoću naredbe `dmidecode` na sljedeći način:

```
dmidecode -t 4

# dmidecode 2.11
SMBIOS 2.7 present.
Handle 0x0400, DMI type 4, 42 bytes
Processor Information
        Socket Designation: Proc 1
        Type: Central Processor
        Family: Xeon
        Manufacturer: Intel
        ID: C2 06 02 00 FF FB EB BF
        Signature: Type 0, Family 6, Model 44, Stepping 2

Flags:
                FPU (Floating-point unit on-chip)
                PAE (Physical address extension)
                MCE (Machine check exception)
                CX8 (CMPXCHG8 instruction supported)
                ACPI (ACPI supported)
                MMX (MMX technology supported)
                SSE (Streaming SIMD extensions)
                HTT (Multi-threading)
        Version: Intel(R) Xeon(R) CPU E5606 @ 2.13GHz
        Voltage: 1.4 V
        External Clock: 133 MHz
        Max Speed: 4800 MHz
        Current Speed: 2133 MHz
        Upgrade: Socket LGA1366
        Core Count: 4
        Core Enabled: 4
        Thread Count: 4
```

Naime u Linuxu, većina informacija o procesoru se nalazi u nekoj od datoteka unutar */proc* ili */sys* direktorija koji predstavljaju posebni datotečni sustav kernela. Vezano za procesor podaci su unutar direktorija `/sys/devices/system/cpu/`, a osnovni podaci se nalaze i u datoteci `/proc/cpuinfo`. Važno je znati kako svaku od ovih funkcionalnosti po logici moramo promatrati kao da je integrirana na zasebnoj kartici ili integriranom sklopu (*čipu*). Stoga je potreban i upravljački program za svaku od njih. Upravljački program je obično ili već integriran u postojeći linux kernel ili je dostupnu kao (zasebni) linux kernel modul.



Više o kernel modulima u poglavlju: **11.1. Kernel.**



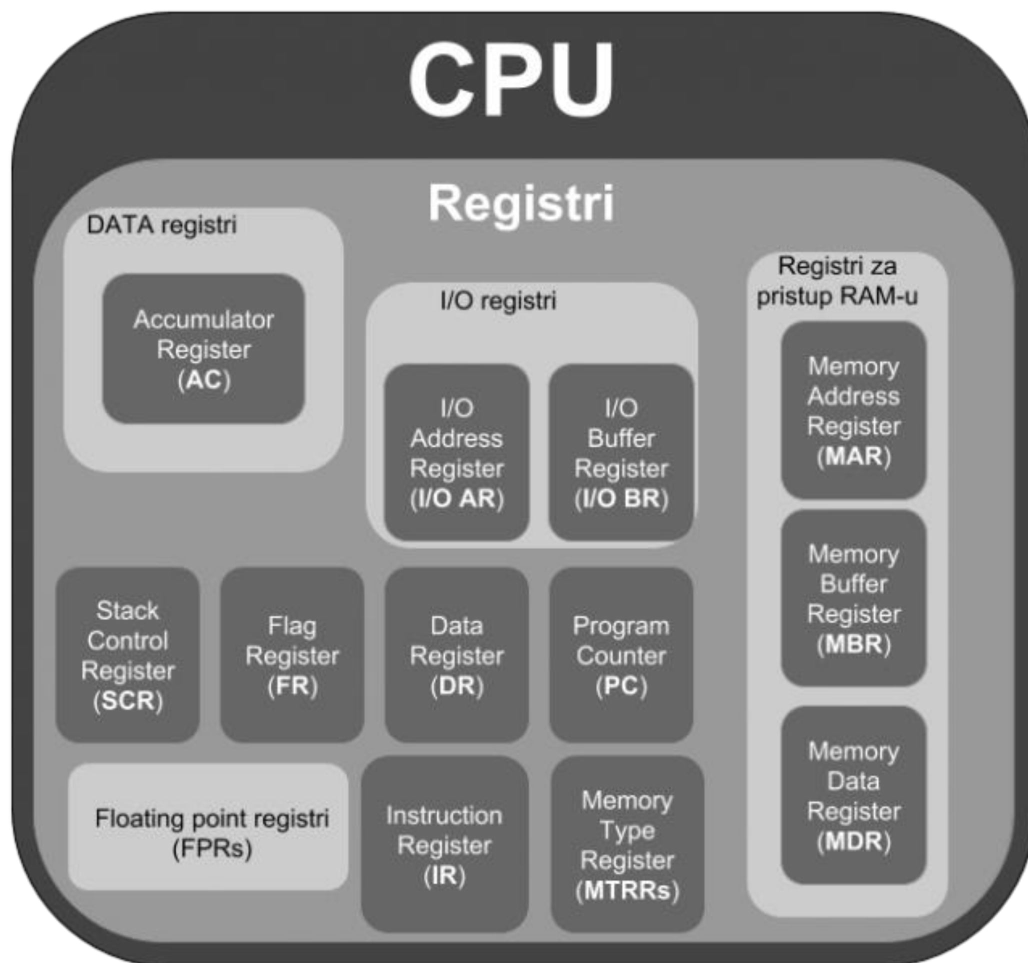
Ako smo i ispravno učitali upravljački program (*kernel modul*) za određenu CPU funkcionalnost, to ne znači kako će ona raditi sâma od sebe, već je za njenu upotrebu obično potreban i dodatan softver koji ju zna iskorištavati.

**Izvori informacija: (71), (K-9), `man dmidecode` , `man 5 proc`.**

## 10.7.1. CPU registri

**CPU** *registri* se koriste za brzo dohvaćanje, spremanje te prijenos podataka i *instrukcija* kojima CPU može trenutno pristupiti, bez potrebe za posredovanjem preko neke druge komponente. Registar može sadržavati instrukciju, adresu za pohranu ili bilo koju vrstu podataka. Neke instrukcije specificiraju registre kao dio instrukcija.
Na primjer, instrukcija može specificirati da se sadržaj dva definirana registra zbroji i zatim pohrani u neki treći registar.

**Slika 57. Pogled unutar arhitekture mikroprocesora; registri.**




Sâmi registri se nalaze između priručne memorije CPU-a odnosno takozvane:

- ***Level 1*** međumemorije: priručne male, ali ekstremno brze međumemorije prve razine (za instrukcije i podatke).
- ***Level 2*** međumemorije: priručne vrlo brze međumemorije druge razine.
- I/ili ***Level 3*** međumemorije

Te ostatka *CPU*a, s jedne strane, a s druge strane prema ostalim komponentama sustava.

Današnji procesori (CPU) imaju nekoliko osnovnih kategorija registara, poput ovih na slici (slika 57.).

CPU registri se koriste i za obavljanje specijaliziranih zadaća, svaki prema svojoj namjeni.
Na vrlo niskoj razini sve operacije (*instrukcije*) u konačnici završavaju u nekom od registara, koji ih dalje obrađuju.

Kada se odrađuje bilo koja radnja na računalu odnosno kada se izvršavaju procesorske **instrukcije**, svi podaci se pohranjuju u *registre*, kao i privremeni podaci u toku obrade podataka. Na kraju se i rezultat obrade podataka čita iz sâmih registara.

Pogledajmo i tablicu s popisom broja dostupnih registara ovisno o arhitekturi procesora:

| Arhitektura procesora | Broj registara opće namjene: *data* i adresnih registara | Broj registara s pomičnim zarezom (*Floating Point*) registara |
|---|---|---|
| **16 bitni x86** | 6 | 8 |
| **32 bitni x86** | 8 | 8 |
| **64 bitni x86** | 16 | 16 (32 ako se koriste AVX512 instrukcije) |
| **SPARC** | 31 | 32 |
| **Power** arhitektura | 32 | 32 |
| **32 bitni ARM** | 14 | varira do 32 |
| **64 bitni ARM** | 31 | 32 |
| **MIPS** | 31 | 32 |
| **Risc-V** | 31 | 32 |

**Izvori informacija: (1453).**

### 10.7.1.1. Malo detaljnije o registrima, instrukcijama i arhitekturi procesora

Registri koji su zaduženi za pristup RAM memoriji pohranjuju memorijske adrese s kojih CPU želi čitati ili pisati: ovo radi takozvani **MAR** registar. U ovom slučaju kada CPU želi nešto snimiti ili pročitati iz RAM memorije on sprema adrese potrebnih memorijskih lokacija u **MAR** registar. S druge strane **MBR** registar pohranjuje sadržaj (podatke ili instrukcije) koji se čita ili zapisuje u RAM memoriju. Sadržaj instrukcija koji se nalazi u ovom registru se prebacuje u **IR** registar (*Instruction Register*), dok se sadržaj (podaci) prebacuju u **I/O** registre ili **AC** registar. Drugim riječima **MBR** registar se koristi za pohranu podataka ili instrukcija koje dolaze ili odlaze prema RAM memoriji.

***I/O Adresni*** registar se koristi za adresiranje I/O (ulazno/izlaznih) uređaja, dok se ***I/O Buffer*** registar koristi za razmjenu podataka (engl. Data) između I/O (ulazno/izlaznih) komponenti i procesora.

***Program Counter*** (**PC**) registar koji se inače zove i *Instruction Pointer* odnosno **IP** registar se koristi za spremanje adrese sljedeće instrukcije koja je na redu za izvršavanje. Naime kada je prva instrukcija izvršena od strane procesora, vrijednost ovog registra se povećava, tako da on uvijek pokazuje na prvu sljedeću instrukciju koju procesor mora dohvatiti (engl. *Fetch operacija*) te potom i izvršiti. U tom procesu dohvaćanja (*Fetching*) instrukcije iz memorije, ona se prvo pohranjuje u *Instruction Registar* tj. **IR** registar. Tada *Control Unit* odnosno **CU** uzima instrukciju iz ovog registra i izvršava ju, slanjem signala drugoj komponenti koja ga zapravo može izvršiti (a ovisno o samoj instrukciji).

***Accumulator Register*** (**AC**) se nalazi unutar **ALU** jedinice te se koristi za sve aritmetičko logičke operacije *ALU*a (engl. *Arithmetic Logic Unit*). Njega koristi **CU** (*Control Unit*) tako što u njega sprema podatke koje je dohvatila iz RAM memorije za aritmetičko logičke operacije. Dakle ovaj registar drži ulazne podatke nâd kojima će se raditi neke od aritmetičko logičkih operacija, međurezultate i konačne rezultate ovih operacija. Konačni rezultati se tada prebacuju u RAM memoriju pomoću **MBR** registra.

**Stack Control Register** (**SCR**). Stog (engl. *Stack*) predstavlja grupu memorijskih blokova unutar kojih se trebaju pohraniti određeni podaci ili se trebaju pročitati u određenom nîzu. Podaci se čitaju ili zapisuju u *stôg* prema principu **FILO** (engl. *First In and Last Out*) odnosno prvi koji uđe zadnji izlazi. **SCR** registar se koristi za upravljanje stôgom (*Stack*) podataka u RAM memoriji. Veličina ovog registra je relativno malih 2 ili 4 bajta.

**Flag Register** (**FR**) se koristi za indikaciju određenih stanja ili uvjeta u radu procesora. Ovaj registar je prilično mali (1 ili 2 *bajta*) te se koristi za pohranjivanje određenih *alarma* ili posebnih *zastavica procesora*. Dakle u slučajevima kada se određene instrukcije izvršavaju te se dogodi neki uvjet, moguće je postavljanje tzv. zastavice unutar ovog registra, koja se provjerava, te ovisno o tome što ona signalizira, mogu se pokrenuti određene akcije/radnje unutar algoritma koji se izvršava.
**Data Register** (**DR**) se koristi za privremeno spremanje podataka koji će biti ili poslani ili učitani s perifernih uređaja.

## Što su instrukcije koje stalno spominjemo

*Instrukcije* o kojima smo govorili su zapravo naredbe koje podržava određeni procesor, odnosno arhitektura procesora, ali na vrlo niskoj razini. Ove naredbe su poput naredbi u višim programskim jezicima (**C/C++/JAVA/Scala/Python/Perl/PHP**…) kojima se indirektno daju instrukcije procesoru koji ih odrađuje. Ove instrukcije na najnižoj razini su ugrađene u sâm procesor i ovise o arhitekturi procesora. Kao što smo vidjeli i iz tablice s registrima, tako ovisno o arhitekturama procesora ovise i instrukcije koje postoje, kao i njihov broj. Važno je znati da postoji nekoliko međusobno nekompatibilnih osnovnih arhitektura procesora:

- **16. bitni x86** - proizvode ih tvrtke: *Intel* ili *AMD (i još neke druge)*.
- **32. bitni x86** - proizvode ih tvrtke: *Intel* ili *AMD (i još neke druge)*.
- **64. bitni x86** - proizvode ih tvrtke: *Intel* ili *AMD (i još neke druge)*.
- **Itanium** - proizvodi ga tvrtka *Intel*.
- **Motorola 6800** i **Motorola 68000** proizvodi ih tvrtka *Motorola*.
- **SPARC** - proizvode ga tvrtke *SUN/Oracle, Fujitsu, Panasonic* i neki drugi.
- **IBM power** - proizvodi ga tvrtka *IBM*.
- **Power** -- otvorena arhitektura — proizvode ga: *IBM, Freescale, AppliedMicro, LSI*, … .
- **ARM 32** ili **32/64.bitni** - licencira se dizajn, a proizvodi ga cijeli nîz tvrtki širom svijeta.
- **MIPS** - licencira se dizajn — *MIPS technologies/Imagination Technologies.*
- **RISC-V –** ovo je otvorena arhitektura — razvijaju ga i proizvode mnoge tvrtke, institucije i zaklade**.**

Vidljivo je da postoji cijeli nîz arhitektura procesora, od kojih svaka arhitektura ima točno određeni skup instrukcija.
To znači da prvo operativni sustav, a nadalje i programi koji su pisani za jednu arhitekturu procesora (govorimo o razini strojnog kôda), ne mogu raditi na drugoj arhitekturi procesora jer imaju potpuno različite instrukcije.

Međutim i unutar jedne arhitekture procesora kao što je recimo 64. bitna **x86** arhitektura (`x86-64`) postoje razlike: od proizvođača do proizvođača i od procesora do procesora. To je zbog dodatnih značajki za koje također postoje zasebne instrukcije. Primjerice iza svake od značajki (pr. **AES**) stoji cijeli skup novih instrukcija. Ako govorimo o ukupnom broju instrukcija unutar procesora, on ovisi o tome koje sve značajke procesor ima (tzv. *CPU flagove*), pa se ukupna brojka može popeti čak na preko **nekoliko tisuća** *x86* instrukcija (**izvor: 1457**). Pogledajmo tablicu s popisom samo nekoliko osnovnih skupova instrukcija:

| **Skup instrukcija** (*značajke*/***flags***) | ***MMX*** | ***MMX+ i SSE*** | ***MMX u SSE2*** | ***VT-x*** | ***SSE*** | ***SSE2*** | ***AVX*** | ***AVX2*** | ***AES*** | ***RDRAND i RDSEED*** | ***SHA*** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Broj instrukcija** | **~60** | **~14** | **~2** | **~17** | **~54** | **~12** | **~12** | **~30** | **~6** | **~6** | **~7** |

Naime, ako neki program koristi specifične značajke (*instrukcije*) jednog procesora, postoji velika vjerojatnost kako on neće raditi na drugom procesoru iako pripadaju istoj arhitekturi procesora (poput **x86**). Stoga se, osim kada to nije sa stvarnim razlogom, izbjegava korištenje poziva na vrlo niskoj razini koji koriste točno određene instrukcije procesora, već se najčešće taj posao prepušta (višem) programskom jeziku. To znači da tada programski jezik više razine, prema programeru radi apstrakciju svih detalja s niske razine, kojima često programer niti nema pristup.

Promatramo li još nižu razinu, koja je dostupna primjerice, ako programiramo u programskom jeziku poput ***Assemblera***, tek tada dajemo direktne (*strojne*) *instrukcije* procesoru, poput ovih u primjeru dolje, gdje je vidljiv *assemblerski* programski kôd.
U ovom primjeru *assemblerskog* programskog kôda, varijabli `var1` dodjeljujemo vrijednost `123`, a potom u sljedećem koraku *CPU* registru `EAX` postavljamo vrijednost te varijable:

```
var1 DWORD 123

mov ebx, var1
add eax, ebx
```

**Ovo je i najniža razina kojoj programeri mogu pristupiti**!
Popis svih instrukcija strojnog kôda programskog jezika *assembler*, što standardnih (osnovnih) a što onih koji su specifični za pojedine procesore, možete vidjeti na stranici/adresi: https://en.wikipedia.org/wiki/X86_instruction_listings

## Arhitektura skupa instrukcija (*instruction set architecture* [ISA])

U informatici, arhitektura skupa instrukcija (**ISA**), se naziva i računalna arhitektura. Općenito, **ISA** definira podržane instrukcije, vrste podataka, registre, hardversku podršku za upravljanje (**RAM**) memorijom, temeljne značajke i ulazno/izlazni model obitelji implementacije **ISA**-e. Ako operativni sustav održava standardno i kompatibilno binarno sučelje aplikacije (**ABI**) za određenu **ISA** arhitekturu, strojni kôd će se izvoditi na svim budućim implementacijama te **ISA** arhitekture i operativnog sustava. Međutim, ako **ISA** podržava pokretanje više operativnih sustava, to ne jamči da će se strojni kod za jedan operativni sustav moći izvoditi na drugom operativnom sustavu, osim ako prvi operativni sustav ne podržava izvršavanje strojnog kôda napisanog za drugi operativni sustav. Arhitektura skupa instrukcija razlikuje se od mikroarhitekture računala, koja je skup tehnika dizajna procesora koje se koriste, u određenom procesoru, za provedbu skupa instrukcija. Ipak, procesori s različitim *mikroarhitekturama* mogu dijeliti zajednički skup instrukcija. Na primjer, ***Intel Pentium*** i ***AMD Athlon*** implementiraju gotovo identične inačice skupa instrukcija **x86**, ali imaju radikalno različite unutarnje dizajne o kojima ćemo ubrzo pričati.

## Mikroarhitekture procesora i Linux

Tijekom 2020. godine tvrtke ***AMD***, ***Intel***, ***Red Hat*** i ***SUSE*** za osnovnu arhitekturu procesora **x86-64** definirale su nekoliko razina mikroarhitektura. U navedenim razinama mikroarhitektura grupirane su značajke procesora (tzv. *CPU flagovi*) odnosno skupovi instrukcija koje se mogu promatrati i na temelju datuma izdavanja procesora. Vezano za navedene arhitekture procesora, kod **x86-64** kompatibilnih arhitektura procesora razlikujemo nekoliko generacija mikroarhitekture, za koje se kompiliraju programski (softverski) paketi, ali i cijele distribucije Linuxa (počevši od kernela). Tako razlikujemo:

- `x86-64-v1` - prvu generaciju (mikro)arhitekture iz vremena ***AMD K8*** arhitekture to jest iz vremena **AMD Opteron** procesora iz 2003.g. Ovdje imamo **SSE**, **SSE2**, **CMOV**, **FPU**, **MMX** i druge skupove instrukcija.
- `x86-64-v2` - drugu generaciju (mikro)arhitekture za ***Intel Nehalem*** i novije procesore. Ova mikroarhitektura donosi podršku (između ostalog) za proširene vektorske instrukcije za *Streaming SIMD* 4.2 (tzv. **SSE4.2**) kao i proširene instrukcije streaming SIMD (tzv. **SSSE3**), **POPCNT** instrukcije (korisne za analizu podataka i manipulaciju bitovima u nekim strukturama podataka) te **CMPXCHG16B** instrukcije (instrukcije za usporedbu i zamjenu korisne za paralelne algoritme). Dakle radi se o procesorima proizvedenim od 2008,g, do 2011.g.
- `x86-64-v3` - treću generaciju (mikro)arhitekture za ***Intel Haswell*** i novije procesore, s kojima dolazi podrška (između ostalog) za vektorske instrukcije **AVX** i **AVX2 i MOVBE** (za *big-endian* pristup podacima) i dodatne instrukcije za manipulaciju bitovima. Ovdje se uglavnom radi o većini procesora proizvedenih od 2013.g. do 2017.g.
- `x86-64-v4` - četvrtu generaciju procesora koji podržavaju **AVX512** skupove instrukcija.

Naime svi moderni programi za kompiliranje (pr. **GCC** v.11 i **LLVM Clang** v.12.) nude prilikom kompiliranja mogućnost odabira *mikroarhitekture* procesora za koju želimo kompilirati programe. S time se dobivaju bolje performanse programa koji u konačnici kompiliramo. To primjerice znači da ako je određena distribucija Linuxa kompilirana za noviju generaciju **x86-64** mikroarhitekture procesora, da neće raditi na starijoj. Naime, s vremena na vrijeme najnovije distribucije Linuxa kompiliraju se za novije generacije mikroarhitektura procesora, kako bi radile što optimalnije (čitaj brže). Tako su primjerice **Red Hat 9** i **Fedora 32** kompilirani za `x86-64-v2` mikroarhitekturu procesora, što znači da neće raditi na starijoj (`x86-64-v1`) mikroarhitekturi procesora.

Na **Red Hat** kompatibilnim distribucijama Linuxa pomoću GNU C biblioteke (**glibc 2.33** ili novije) možemo provjeriti koje sve mikroarhitekture procesora podržava naš procesor. Ispis smo skratili samo na dio vezan za mikroarhitekturu procesora:

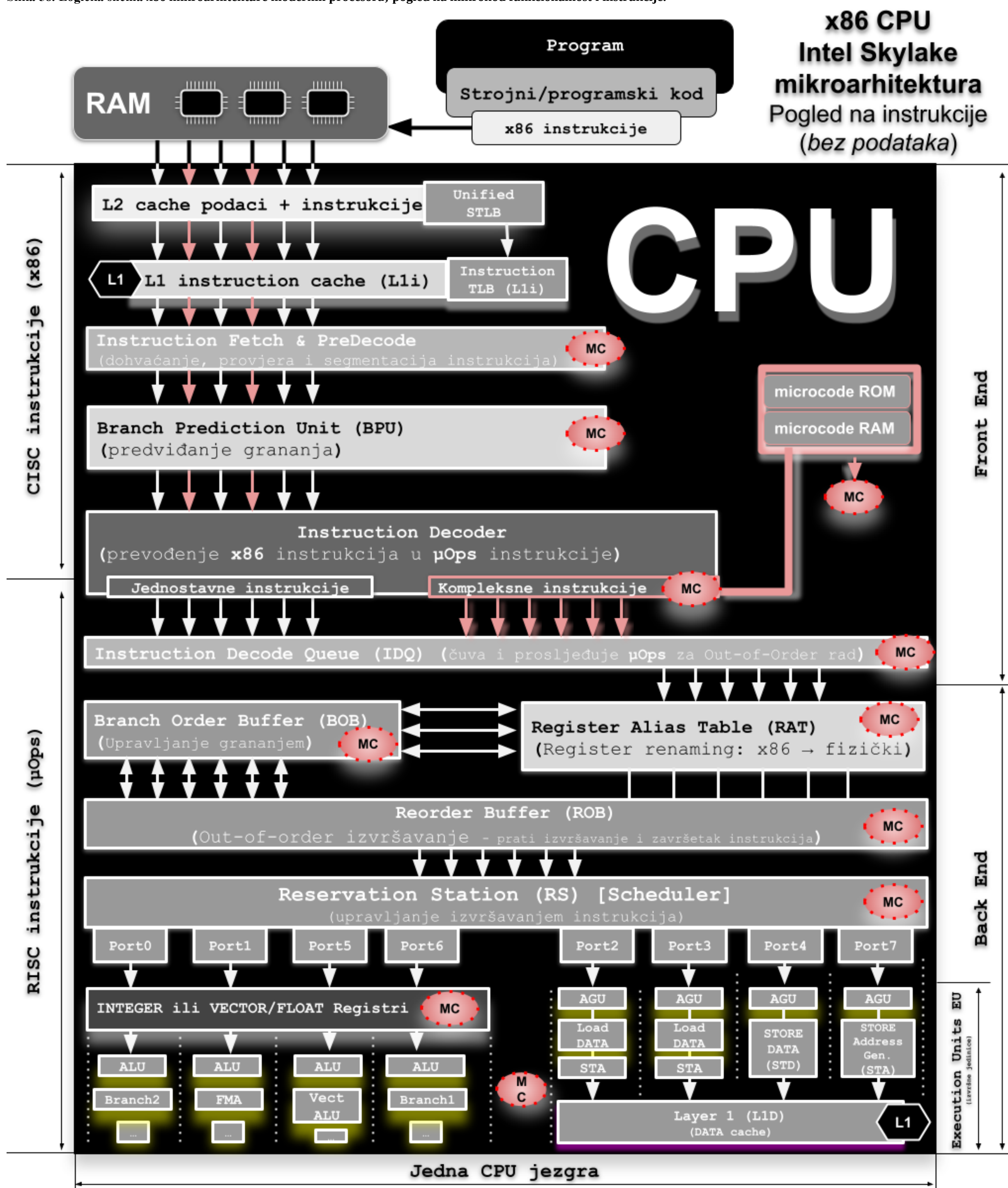
```
/usr/lib64/ld-linux-x86-64.so.2 --help
. . .
Subdirectories of glibc-hwcaps directories, in priority order:
  x86-64-v4
  x86-64-v3
  x86-64-v2 (supported, searched)
```

U našem slučaju (ispisu) naš procesor podržava samo `x86-64-v2` *mikroarhitekturu*.
To znači da na njega možemo instalirati **Red Hat 9x** i kompatibilne distribucije poput: **Fedora 32+**, **Rocky 9+**, **Oracle Linux 9+** te sve druge distribucije Linuxa koje za rad zahtijevaju ovu *mikroarhitekturu* procesora.
Dok najnoviji **Red Hat v.10.** (izlazi tijekom 2025.g.) zahtjeva minimalno `x86-64-v3` *mikroarhitekturu* procesora.

**Pohrana podataka u RAM memoriju (*little-endian* i *big-endian*)**
Na najnižoj hardverskoj razini podaci se u RAM memoriju pohranjuju ovisno o arhitekturi procesora. Primjerice kod **x86, ARM i RISC-V** arhitektura procesora se koristi takozvani ***little-endian*** način pohrane. To znači da kada se niz podataka (bajtova) pohranjuje u memoriju, bajt najmanje važnosti dolazi prvi. Što znači da je redoslijed bajtova podataka efektivno obrnut kada se pohranjuje u memoriji. Ako se primjerice treba pohraniti heksadecimalni niz: `0x AB CD EF 11`, on će tada u memoriji biti pohranjen kao: `0x 11 EF CD AB`. Dok se kod ***big-endian*** načina pohrane podaci pohranjuju bez obrtanja.

**Instrukcije**
Vezano za instrukcije, postoji i još niža razina. Ako se spustimo na tu nižu razinu, tada shema cijelog procesa obrade našeg programa, sa svim instrukcijama, logički izgleda kao na slici 58.
Slika predstavlja pojednostavljenu logičku shemu x86 ***mikroarhitekture Intel Skylake*** procesora iz 2015. godine.

**Slika 58. Logička shema x86 mikroarhitekture modernih procesora; pogled na mikrokôd funkcionalnost i instrukcije.**




U prvom koraku procesor preuzima strojni programski kôd iz RAM memorije, odnosno naš program, sa svim pripadajućim **x86** instrukcijama od kojih se naš program sastoji (ovdje nećemo spominjati podatke koji se pri tome dohvaćaju ili obrađuju).
To radi tako da se instrukcije prvo dohvaćaju u vrlo brzu internu RAM memoriju unutar procesora koja se naziva ***Layer 2 cache*** međumemorija koja se koristi i za podatke i instrukcije, a primjerice u ***Intel Skylake mikroarhitekturi*** procesora ona je veličine 256kB po jezgri procesora.

Ako pretpostavimo da se pola ove memorije koristi za instrukcije (što je vrlo dinamički i varira), te da je prosječna veličina **x86-64** instrukcije veličine 4 bajta to nas dovodi do 32.768 instrukcija koje se mogu ovdje privremeno pohraniti.
Međutim, ***Layer 2*** međumemorija je ipak sporija od ostatka procesora pa se stoga iz nje instrukcije dohvaćaju u ekstremno brzu ***Layer 1*** međumemoriju koja u dio namijenjen samo za instrukcije (***L1i cache***), koji je u navedenoj mikroarhitekturi procesora iznosi 32kB po jezgri procesora, pohranjuje manji dio instrukcija. Po prijašnjoj računici do maksimalno 8.192 instrukcije.

*U navedenim okvirnim računicama moramo biti svjesni toga da **x86-64** instrukcije nisu fiksne veličine već variraju od 1 do 15 bajta. Stoga je jasno da je gornji izračun varijabilan.*

Tek sada se instrukcije koje se nalaze u ***Layer 1 L1i*** *cache* međumemoriji mogu dohvatiti (*cache hit*) odnosno, ako su potrebne, a ne nalaze se u toj međumemoriji, dolazi do promašaja (*cache miss*) pa se moraju dohvatiti iz ***Layer 2*** međumemorije (što je sporije) ili se u najgorem slučaju moraju dohvatiti iz RAM memorije (što je još sporije). U svakom slučaju ovaj mehanizam dohvaćanja instrukcija se uvelike oslanja na *Branch Prediction Unit* (**BPU**) koji pokušava predvidjeti smjer grananja programskog kôda smanjujući čekanje i ubrzavajući obradu predviđanjem toka instrukcija u kôdu.

## CISC i RISC arhitektura i instrukcije

U **x86** mikroarhitekturi procesora operativni sustav i aplikacije koriste **x86** instrukcije koje se općenito nazivaju *complex instruction set computer* (**CISC**) instrukcije. Međutim s **x86** (*CISC*) instrukcijama procesor barata samo unutar vršnog dijela procesora koji se naziva ***Frontend***. Zatim se **x86** (*CISC*) instrukcije u donjem dijelu procesora pretvaraju u *reduced instruction set computer* (**RISC**) instrukcije koje interno **x86** procesor i izvršava. Donji dio procesora koji radi s *RISC* instrukcijama naziva se ***Backend***. Ovakva arhitektura (CISC izvana + RISC iznutra) je specifična za **x86** arhitekturu procesora.

Zbog čega **CISC** arhitektura procesora na ***x86*** procesorima prema operativnom sustavu i aplikacijama (*Frontend*):

- **Kompatibilnost i naslijeđe** – x86 arhitektura procesora je razvijena 1978. godine i koristi CISC filozofiju kako bi omogućila izvršavanje složenih zadataka s manje redaka programskog kôda.
- **Smanjenje veličine kôda** – CISC omogućuje kraći programski kôd jer složene operacije poput množenja matrica ili upravljanja memorijom mogu biti obavljene u jednoj (x86) instrukciji.
- **Jednostavno programiranje** – složene instrukcije omogućuju programerima jednostavniji razvoj programa.

Moderni x86 procesori mogu imati čak do **nekoliko tisuća x86** instrukcija (~3.684) (**izvori: 1456, 1457**).

Unutar x86 procesora (*Backend*) imamo i *reduced instruction set computer* (**RISC)** arhitekturu procesora čije značajke su:

- **Povećanje performansi** – interno x86 procesori pretvaraju **CISC** instrukcije u skup jednostavnijih **RISC** mikroinstrukcija (tzv. ***µOps***). Ove *mikroinstrukcije* omogućuju brže izvođenje na modernim *superskalarnim* arhitekturama procesora s više cjevovoda.
- **Bolja optimizacija** – jednostavne instrukcije lakše se paraleliziraju i izvršavaju na više procesorskih jezgri. Dodatno to omogućuje modernim procesorima korištenje dinamičko izvršavanje instrukcija izvan slijeda kojim su došli (tzv. *Out-of-Order Execution*), u RISC-stilu kako bi se smanjila latencija.
- **Energetska učinkovitost** – jednostavne RISC instrukcije troše manje energije i omogućuju brže prebacivanje između stanja mirovanja i rada.

Broj korištenih **RISC** instrukcija (*µOps*) je višestruko manji od broja **x86** instrukcija.

Pogledajmo tablicu s arhitekturama nekih od procesora:

| Arhitektura | Vrsta arhitekture | Opis |
|---|---|---|
| **x86** | **CISC + RISC** | Koristi CISC instrukcije; dok interno rade u RISC stilu (***µOps***). |
| **x86-64** | **CISC + RISC** | Produžetak x86 sa 64-bitnom podrškom; koristi CISC, ali interno radi s RISC instrukcijama (Produžetak x86 sa 64-bitnom podrškom; koristi CISC, ali interno ima RISC optimizacije.). |
| **ARM** | **RISC** | Klasična RISC arhitektura, jednostavne instrukcije, popularna u mobilnim uređajima. |
| **MIPS** | **RISC** | Čista RISC arhitektura, koristi se u ugrađenim sustavima i akademskim projektima. |
| **SPARC** | **RISC** | Dizajnirana za poslužitelje i radne stanice, fokus na jednostavne instrukcije i paralelizam. |
| **RISC-V** | **RISC** | Moderna otvorena arhitektura s jednostavnim i modularnim instrukcijama. |

**Vratimo se na x86 instrukcije**

**x86** instrukcije koje su dohvaćene iz ***Layer 1*** (***L1i***) međumemorije sada dolaze do dijela zaduženog za njihovo dekoriranje.
To je dio procesora koji se naziva ***Instruction decoder*** i on je zadnja cjelina unutar *Frontend* dijela procesora. On pretvara složene *x86* instrukcije u jednostavnije *mikro-operacije* (µOps) odnosno u specifične **RISC instrukcije**. Pri tome se jednostavne instrukcije dekodiraju direktno, dok se složene šalju u tzv. ***Microcode Sequencer***. *Microcode Sequencer* dekodira složene instrukcije u niz µOps instrukcija koristeći ***mikrokôd***.

Naime, definicija i popis ovih *mikro* instrukcija nalazi se u posebnom dijelu **microcode ROM** memorije ugrađene u CPU, koja se kontaktira u fazi dekodiranja instrukcija. Dodatno, CPU ima i **microcode RAM** memoriju, u koju se mogu spremati i nove inačice *CPU mikrokôda*. Naime često se u nekim od mikro instrukcija otkrivaju greške ili se rade određene poboljšanja, pa ih je moguće nadograditi, nadogradnjom *mikrokoda* CPU-a. Ako gledamo strojne instrukcije koje kao programeri na najnižoj programerskoj razini, primjerice programirajući u programskom jeziku *assembler*, prosljeđujemo procesoru (***x86***), one su više manje standardne, osim onih s kojima baratamo specijaliziranim registrima ili dijelovima procesora. Međutim kada se te strojne ***x86*** instrukcije prevedu u *mikro instrukcije* (*RISC instrukcije*) unutar procesora, tu dolazi do značajnih promjena.

Tako pojedina strojna *x86* instrukcija, prevedena u *mikro instrukcije*, može biti prevedena u više *mikro instrukcija*, jer ne postoji jedna od njih koja može riješiti pojedinu višu instrukciju, već se ta radnja mora odraditi s više *mikro instrukcija* i potencijalno u više taktova procesora. S druge strane moguće je i da se nîz od nekoliko strojnih *x86* instrukcija prevede i obradi kao sâmo jedna *mikro instrukcija*.
Mjera s kojom se označava efikasnost procesora odnosno broj *mikro instrukcija* po ciklusu/taktu procesora je takozvani ***IPC*** ***(engl.*** *Instructions per cycle)* pri čemu je viša vrijednost bolja. Proizvođači procesora se trude u svakoj generaciji procesora povećati ***IPC*** pa je tako recimo **AMD** prelaskom sa *ZEN2* na ZEN3 arhitekturu u ***Ryzen*** procesorima, povećao ***IPC*** za 19%.
Nemojte pobrkati ovaj ***IPC*** s mehanizmom vezanim za komunikaciju između procesa (*inter-process communication*).
Možemo se spustiti na još nižu razinu, jer ako gledamo registre poput (***eax***, ***ebx***, ***ecx***, ***edx***, …), oni uopće nisu fiksirani odnosno ugrađeni u procesor direktno kao određeni tranzistori ili logički sklopovi koji se koriste za točno određenu funkcionalnost registra. Naime gotovo svaki moderni *CPU* ima interno, znatno veći broj registara nego što je stvarno vidljivo.
Pri tome se koristi tehnika znana kao ***register renaming*** za translaciju logičkih registara koje mi koristimo, u one stvarne koje ima procesor i koji su u konačnici zaduženi za primanje, obradu i prosljeđivanje podataka (o tome malo kasnije u tekstu).

***Mikrokod*** se ne koristi samo za dekodiranje složenih instrukcija. On može mijenjati i druge komponente unutar procesora poput:

- **Cache međumemorije** – može optimizirati ponašanje međumemorija, poput L1, L2 i L3 razine, uključujući upravljanje podacima i mehanizmima za otkrivanje podataka u međumemoriji kao i utjecati na spekulativno dohvaćanje podataka u *cache* međumemorije.
- **Dekoderi instrukcija** – može ažurirati način na koji se složenije CISC instrukcije (u x86 arhitekturi) dešifriraju u ***μOps*** (*mikrooperacije/instrukcije*).
- **Instruction Decode Queue (IDQ)** – može promijeniti logiku punjenja i raspodjele instrukcija iz **IDQ** prema ostalim dijelovima procesora.
- **Branch predictor** (*predviđanje grananja*) – može ažurirati algoritam predviđanja grananja kako bi se smanjila stopa pogrešnih predviđanja.
- **Register Alias Table (RAT)** – može ažurirati logiku mapiranja registara arhitekture procesora (x86) na fizičke registre.
- **Reorder Buffer (ROB) –** može utjecati na pravila završavanja instrukcija i vraćanja rezultata u ispravnom redoslijedu ili utjecati na ispravke grešaka povezanih s upravljanjem podacima ili redoslijedom izvršavanja.
- **Reservation Station (RS) –** može poboljšati logiku rezervacije resursa za izvršavanje instrukcija u izvršnim jedinicama (ALU, FPU, …) te riješiti probleme povezane s ovisnostima između instrukcija.
- **Izvršne jedinice (ALU, FPU, …) –** može popraviti greške ili raditi optimizacije specifičnih aritmetičkih operacija (npr. podaci o IEEE 754 standardu za FPU kao i ažurirati instrukcije specifičnih za funkcijske jedinice (npr. AVX-512).
- **Upravljanje sigurnosnim značajkama -** može ispraviti neke od sigurnosnih ranjivosti (pr. *Spectre*, *Meltdown* i sl.).


Za više detalja o ovoj najnižoj razini *mikro instrukcija* procesora, možete pogledati upute tvrtke *Intel*, za njihove procesore: https://software.intel.com/en-us/articles/intel-sdm

Sada dekodirane ***μOps*** (*mikrooperacije/instrukcije*) ulaze u *Instruction Queue* (**IDQ**) koji privremeno pohranjuje dekodirane ***μOps*** prije obrade u *Backend* dijelu procesora. On djeluje kao međuspremnik između *Frontend* i *Backend* dijela procesora.

**Backend dio procesora**
U *Backend* dijelu procesora koji radi samo i isključivo s ***μOps*** (*RISC instrukcijama)* u *Register Alias Table (***RAT***)* se događa preslikavanja virtualnih registara koje koriste instrukcije, na fizičke (stvarne) registre procesora. Naime kada ***μOps*** stigne u *Backend* dio, ***RAT*** provjerava koji logički registri se koriste kao ulazi (operandi) i izlazi (rezultati). Svakom izlaznom logičkom registru (npr. **EAX**) dodjeljuje se novi fizički registar. Ulazni operandi se preslikavaju na odgovarajuće fizičke registre na temelju trenutnog stanja u RAT-u. *μOps* se šalju dalje u *Reservation Station* (**RS**) s fizičkim registrima kao operativnim operandima. *RAT* surađuje s *ROB*-om kako bi se osiguralo pravilno vraćanje preslikavanja registara u slučaju *rollback*-a ili krivog predviđanja grananja (*branch misprediction*). Ukratko, ***RAT*** je tablica koja preslikava logičke registre na fizičke, čime omogućuje procesoru da rješava ovisnosti podataka te povećava brzinu i paralelizaciju obrade instrukcija.
*Branch Order Buffer* (**BOB**) prati redoslijed instrukcija povezanih s grananjem (npr. skokovi, uvjetne naredbe).
Naime ako je predikcija grananja bila točna, instrukcije ostaju u tijeku izvršavanja. A ako nije, **BOB** poništava sve spekulativno izvršene *μOps* (instrukcije) nakon pogrešnog predviđanja te vraća procesor na početak spekulativnog izvršavanja.
Dakle on osigurava izvršavanje samo valjanih instrukcija.
*Reorder Buffer* (**ROB**) pohranjuje *μOps* i njihove privremene rezultate dok se ne potvrdi njihovo izvršavanje. On osigurava da se rezultati izvršavanja zapisuju pravilnim redoslijedom (redoslijed rada programa). Osim toga on provjerava da su svi operandi i uvjeti spremni. Ako *μOps* (instrukcija) nije spremna za izvršavanje, čeka dok ne bude (sinkronizacija). U biti on sinkronizira redoslijed rezultata izvršavanja i čuva privremene rezultate

***Reservation Station*** (**RS**) pohranjuje μOps dok čekaju potrebne operande ili resurse za izvršavanje. A tek kada su svi operandi dostupni, *μOps* se šalju odgovarajućim jedinicama za izvršavanje, preko takozvanih portova (**Port 0 - 7**). Dakle **RS** čeka na valjanost svih uvjeta te na kraju prosljeđuje instrukcije na odgovarajuću izvršnu jedinicu (**EU**) preko specijaliziranih portova.
Pri tome su neki portovi rezervirani za instrukcije (0,1,5 i 6) to jest jedinice za njihovo izvršavanje poput aritmetičkih, logičkih i drugih računskih jedinica poput: ***ALU***, ***FPU***, ***AES***, ***DIV*** ,..
Dok su drugi portovi (2,3,4 i 7) specijalizirani za memorijske operacije poput učitavanja ili spremanja podataka u memoriju (***STA***, ***STD***, …). Odvajanjem memorijskih portova od portova za instrukcije omogućuje se procesoru da istovremeno obrađuje i memorijske operacije i aritmetičko/logičke operacije. Ovakva podjela smanjuje zagušenja na pojedinim portovima i omogućuje da procesor u svakom trenutku koristi maksimalan broj svojih izvršnih jedinica (EU).
← *Ovdje govorimo o detaljima **Intel Skylake** mikroarhitekture procesora!*

**I konačno sve dolazi na paralelnu obradu**
Broj i vrsta *Execution Unit-a* (**EU**) ovisi o dizajnu mikroarhitekture i specifičnim funkcijama procesora. Primjerice navedeni *Intel Skylake* procesori sadrže različite vrste izvršnih jedinica, a njihova paralelizacija i performanse ovise o broju raspoloživih resursa, vrsti instrukcija i upravljanju njihovim međuovisnostima.

U navedenoj *Skylake* mikroarhitekturi procesora po pitanju izvršnih jedinica (***EU-a)*** tako imamo:

- **ALU** (*Arithmetic Logic Units*) - obavljaju osnovne aritmetičke i logičke operacije (zbrajanje, oduzimanje, AND, OR, itd.). *Skylake* procesor obično ima 4 ALU-a po jezgri. To znači da se 4 aritmetičke ili logičke operacije mogu paralelno izvršavati unutar jedne jezgre procesora.
- **AGU** (*Address Generation Unit*) – obavlja izračune za adresiranje memorije (npr. za LOAD/STORE operacije). *Skylake* procesor obično ima 4 AGU-a.
- **FPU** (*Floating Point Unit*) – obavlja operacije s pomičnim zarezom (množenje, dijeljenje, itd.). *Skylake* procesor obično ima 2 FPU-a po jezgri. To znači da svaka jezgra obično ima 2 FPU-a za izvođenje operacija s pomičnim zarezom, poput množenja i dijeljenja.
- **MUL/DIV** Units – obavljaju množenje i dijeljenje. *Skylake* procesor ima 1 ili 2 MUL/DIV jedinice po jezgri, ovisno o konkretnoj implementaciji.
- **SIMD** Units (*Single Instruction, Multiple Data*) – omogućuju istovremeno izvršavanje iste instrukcije na više podataka (npr. uvećavanje registara s vektorima ili matrix operacije). *Skylake* procesor podržava AVX2 i AVX-512, koji omogućuju veći broj paralelnih operacija unutar vektora.
- **Store Data Unit** (**STD**) – odgovorna za obavljanje operacija zapisa u memoriju. *Skylake* procesor ima 1 STD jedinicu po jezgri.

Važno za paralelizam sve ove izvršne jedinice unutar svake jezgre procesora omogućuju paralelno izvršavanje različitih vrsta instrukcija, što doprinosi većoj učinkovitosti. Pri tome je ukupna brojka za **ALU**, **AGU**, **FPU** i druge jedinice u procesoru broj izvršnih jedinica po jezgrama pomnožen s brojem CPU jezgri. Na primjer, ako *Skylake* procesor ima 4 jezgre, svaku s 4 ALU-a, procesor bi imao ukupno 16 ALU-a (4 ALU-a po jezgrama × 4 jezgre).
Svaka jezgra procesora posjeduje svoj vlastiti skup izvršnih jedinica, pa se broj ALU, AGU, FPU, i drugih jedinica odnosi na pojedinu jezgru procesora. To omogućuje svakoj jezgri procesora paralelno izvršavanje različitih operacija, povećavajući ukupnu procesorsku snagu.
Svaka fizička CPU jezgra *Skylake* arhitekture ima ukupno 8 portova, od kojih su 4 porta za instrukcije i 4 porta za memorijske operacije. Ti portovi su ključni za izvršavanje operacija unutar procesora i igraju važnu ulogu u kontekstu ***Hyper-Threadinga*** i povezanosti s logičkim CPU jezgrama. Naime *Hyper-Threading* (**HT**) omogućuje da svaka fizička jezgra procesora simulira dvije logičke jezgre. To znači da procesor može paralelno izvršavati dvije programske niti na svakoj logičkoj jezgri. To također znači da svaka logička jezgra dijeli iste fizičke resurse procesora, uključujući portove, rezervacijske stanice (**RS**), **ROB**, i jedinice za izvršavanje (**EU**). To se događa dinamički: ako jedna logička jezgra ne koristi određeni port, druga ga može iskoristiti u potpunosti. A kada obje logičke jezgre trebaju iste portove, operacije se redaju prema prioritetima i dostupnosti.
Važno je znati da ***Intel Skylake*** mikroarhitektura procesora, kao i većina modernih procesora (od 2015.g.) koristi niz sofisticiranih tehnologija za omogućavanje paralelnog izvršavanja instrukcija poput:

- ***Hyper-Threading*** (***Simultaneous multithreading***) – omogućuje izvršavanje više programskih niti na jednoj fizičkoj jezgri procesora. To znači da svaka fizička jezgra procesora može istovremeno obrađivati dvije nîti koje se izvršavaju unutar logičkih jezgri procesora.
- ***Memory-Level Parallelism (MLP)*** – omogućuje istovremeni pristup različitim razinama memorijske hijerarhije. Konkretno *Address Generation Units* (**AGU**) omogućuje paralelno dohvaćanje podataka iz L1D, L2 međumemorija ili čak iz RAM-a. To znači da se višestruki pristupi memoriji obrađuju se istovremeno.
- ***Multi-Ported Execution Units –*** omogućava razdvajanje aritmetičko/logičkih i memorijskih jedinica. Ovakvim razdvajanjem omogućuje se istovremeno izvršavanje aritmetičko/logičkih (računskih) operacija i operacija čitanja i zapisivanja u memoriju.
- ***Single Instruction, Multiple Data*** (***SIMD***) proširenja – omogućuje paralelno izvođenje iste operacije na više skupova podataka, korištenjem dodatnih skupova instrukcija poput: ***SSE***, ***AVX***, ***AVX2***, i ***AVX-512***.
- **Speculative Execution** - omogućuje paralelno izvršavanje instrukcija iz određenih grana programa prije nego što se utvrdi ispravnost predviđanja (***Branch Prediction***).
- ***Branch Prediction*** - omogućuje proaktivno izvršavanje instrukcija prije nego što se utvrdi tijek programa. On radi tako da predviđa ishod grananja programskog kôda (npr. **if-else**) koristeći *Branch Target Buffer* (**BTB**) i složene prediktivne algoritme.
- ***Micro-Op Fusion –*** omogućuje spajanje dvije ili više uobičajenih x86 instrukcija u jedan **µOp** (RISC instrukciju). To radi tako da se primjerice instrukcije **CMP** (usporedba) i **JMP** (skok) mogu se spojiti u jedan ***µOp*** što smanjuje broj *µOps* koje treba izvršiti, omogućujući veću paralelnost u radu.



Pogledajte i poglavlja:
**10.7.1. CPU registri.**
**10.7.2.1. SMP.**

**Izvori informacija: (140),(141),(142),(143),(144),(145),(150),(151),(152),(1453),(1454),(1455),(1456),(1457),(1459),(1537),(1538),(1776),(1777),(1778),(1779),(1780),(1781),(1782),(1783),(K-9).**

## 10.7.1.2. CPU Mikrokod

Kako smo već spomenuli, s vremena na vrijeme otkrivaju se greške, ali se rade i poboljšanja *mikro instrukcija*, te je zbog toga potrebna nadogradnja *mikrokôd* dijela procesora. Dodatno, moguće je kreirati i nove mikro instrukcije, unutar *mikrokôd* nadogradnje. Ipak, veće greške unutar dizajna procesora, ne mogu se riješiti nadogradnjom mikro kôda procesora.
Nadogradnja mikro kôda radi se obično tako, da se ona nadograđuje unutar *Firmwarea BIOSa* matične ploče. Ova nadogradnja se odrađuje na način, da se nakon nadogradnje *BIOSa* matične ploče, tijekom svakog novog pokretanja ili restarta računala, mikro kôd za CPU učitava u *microcode RAM* memoriju unutar procesora. Mana ove metode jest u tome, što je vrlo često proizvođačima matičnih ploča potrebno dosta vremena kako bi novi *mikrokôd* ubacili u novi *BIOS* za konkretnu matičnu ploču. Naročito se kod starijih matičnih ploča događa, da proizvođač više ne objavljuje novije inačice *BIOSa*, pa samim time više nema niti nadogradnji *mikrokôda*.
U slučaju sigurnosnih nadogradnji *mikrokôda*, ovakav pristup njegove nadogradnje uopće nije dobar, jer obično proizvođači procesora prvi reagiraju objavljivanjem novog *CPU mikrokôda*, koji bi tada trebalo nadograditi što prije, ne čekajući i proizvođače matičnih ploča, jer to može potrajati. Srećom, postoji i druga metoda nadogradnje *CPU mikrokôda*, a to je nadogradnja iz Linuxa. Naime Linux u vrlo ranoj fazi inicijalizacije kernela, a prije podizanja ostatka operativnog sustava, može napraviti „*microcode update*“ proceduru. Slična procedura nadogradnje *mikrokôda* može se odrađivati i za drugi hardver, poput RAID ili disk kontrolera, mrežnih kartica i drugog hardvera, koji podržava *microcode* funkcionalnost.
Ovakve nadogradnje se obično nazivaju *Firmware update*.



Za *Firmware update* odnosno nadogradnju, pogledajte poglavlje:
**11.1.2.1.2 Nadogradnja firmwarea uređaja.**

Vratimo se na mikrokôd te provjerimo inačicu mikrokôda, našeg *CPUa*, pogledom u posebnu datoteku: `/proc/cpuinfo`:

```
grep microcode /proc/cpuinfo

microcode       : 0x5
microcode       : 0x5
```

U ovom slučaju imamo CPU sa mikrokôdom inačice `0x5`. Trenutna inačica *CPU* mikrokôda se nalazi zapisana (za svaku CPU jezgru), u posebnoj datoteci: `/sys/devices/system/cpu/cpuXX/microcode/version` što bi za prvu CPU jezgru bila datoteka: `/sys/devices/system/cpu/cpu0/microcode/version`

Pogledajmo i što se događalo u trenutku inicijalizacije (pokretanja) računala, vezano za mikrokôd, pomoću naredbe `dmesg`:

```
dmesg | grep microcode

microcode: CPU0 sig=0xf43, pf=0x10, revision=0x5
platform microcode: firmware: agent loaded intel-ucode/0f-04-03 into memory
microcode: CPU1 sig=0xf43, pf=0x10, revision=0x5
platform microcode: firmware: agent loaded intel-ucode/0f-04-03 into memory
microcode: Microcode Update Driver: v2.00 <tigran@aivazian.fsnet.co.uk>, Peter Oruba
```

Vidljivo je kako je mikrokôd koji je ugrađen u CPU, inačice `0x5` te kako je i mikrokôd koji imamo, očito iste inačice, jer nije došlo do nadogradnje.
Naime da smo imali noviji mikrokôd, dobili bi i druge poruke poput:

```
microcode: CPU0 sig=0xf43, pf=0x10, revision=0x5
microcode: CPU0 updated to revision 0x7, date = 2010-09-28
microcode: CPU1 sig=0xf43, pf=0x10, revision=0x5
microcode: CPU1 updated to revision 0x7, date = 2010-09-28
```

Dakle ako je došlo do nadogradnje, morale bi se vidjeti poruke: `CPU updated to revision XY`.
Ako nemate instaliran, potrebno je instalirati paket: `microcode_ctl`, uz koji dolazi i mikrokôd datoteka od *Intel-a* i *AMD-a*.

```
yum install microcode_ctl
```

Međutim u novijim inačicama *Red Hat/CentOS/Fedora/Rocky* Linuxa, ovaj paket više ne sadrži izvršni program: `microcode_ctl` već sâmo program `intel-microcode2ucode`, koji mijenja dio njegovih funkcionalnosti.



Čisto za informaciju, mikrokôdovi za ***Intel***, ***AMD***, ***VIA*** i ***Freescale*** procesore za Linux su dostupni na lokaciji:
https://github.com/platomav/CPUMicrocodes odnosno samo za *Intel*: https://github.com/intel/Intel-Linux-Processor-Microcode-Data-Files

Standardna lokacija i ime *CPU mikrokôd* datoteke za **Intel** CPU je: `/lib/firmware/microcode.dat`.
Ova datoteka sadrži *mikrokôd* za sve *Intel* procesore (*CPUove*). Ako je naš aktivni kernel kompiliran sa opcijom `CONFIG_MICROCODE_OLD_INTERFACE` tada je moguća nadogradnja *mikrokôda* pomoću ove datoteke (stara metoda).

Provjerimo je li to uopće moguće na našem kernelu, sa sljedećim nîzom naredbi:

```
grep „CONFIG_MICROCODE_OLD_INTERFACE" /boot/config-`uname -r`

CONFIG_MICROCODE_OLD_INTERFACE=y
```

Na novijim generacijama *RedHat/CentOS* (7.x+), koristi se novija metoda nadogradnje CPU mikrokôda. U svakom slučaju, bilo da se radi o staroj ili novoj metodi, njih odrađuje poseban *kernel modul* imena `microcode` koji može biti ugrađen u sâm kernel ili dostupan kao kernel modul datoteka. Ako je ovaj kernel modul aktivan i ispravan, kreirana je i posebna datoteka koja je zadužena za nadogradnju mikrokôda, koja se zove `/dev/cpu/microcode`. Dodatan kernel modul, koji je također važan je i `cpuid`. Oba kernel modula možete privremeno učitati upotrebom naredbe `modprobe`, na sljedeći način:

```
modprobe microcode
modprobe cpuid
```

Kod nove metode za *Intelove procesore*, se datoteka `/lib/firmware/microcode.dat`, koja sadrži *mikrokôdove* za sve modele procesora, konvertira u poseban *ucode* format, pomoću programa: `intel-microcode2ucode`. Potom se kreiraju zasebne datoteke za svaki model procesora koje se potom snimaju unutar direktorija: `/lib/firmware/intel-ucode/`.

Svaka datoteka ima ime koje u nazivu sadrži oznaku (kôd) odnosno identifikator (*signature*) procesora na koji se odnosi.
Najnoviji mikrokôd od strane oba proizvođača, sada dolazi u oba formata. To znači da, ako kopirate *mikrokôd* unutar sâme arhive, za primjerice *Intel* procesore, nalazit će se i datoteka `/lib/firmware/microcode.dat` i sve datoteke u direktoriju `/lib/firmware/intel-ucode/`. Navedeni direktorij sadrži mikrokôd datoteke za svaku generaciju i model procesora, prema nazivu: ***FAMILY-MODEL-STEPPING***.

Provjeriti identifikator odnosno *signaturu* procesora, možemo napraviti sa sljedećim nîzom naredbi:

```
cat /proc/cpuinfo | sort | uniq | egrep „family|model|stepping" | grep -v name
```

S tim da ovdje nećemo vidjeti ***Type*** polje, koje je obično vrijednosti `0`
Stoga za potpunu informaciju, možemo pokrenuti ovu naredbu:

```
dmidecode -t 4 | grep -i Signature
```

Ako želimo dobiti dekadsku oznaku *CPUa*, to možemo dobiti sa sljedećom naredbom:

```
dmidecode -t 4 | grep -i Signature | uniq | awk '{print $5, $7, $9}'
```

Dobit ćemo nešto poput:
`6, 63, 2`
Što, ako ručno pretvorimo u heksadecimalno (svaki broj zasebno), dobivamo:
`06 3F 02`

Stoga će naša CPU datoteka imati ime: `06-3f-02` (koriste se mala slova). Odnosno, datoteka će biti imena: `/lib/firmware/intel-ucode/06-3f-02`.

Za *AMD* procesore se već standardno kreiraju zasebne datoteke unutar direktorija: `/lib/firmware/amd-ucode` koje su već u *ucode* formatu. Nazivi i sadržaj ovih datoteka se odnosi na cijelu jednu generaciju AMD procesora, što znači kako jedna datoteka sadrži *mikrokôd* za cijelu generaciju odnosno nîz procesora. Nakon što su kreirane ili kopirane sve *ucode* datoteke, za *Intel* ili *AMD* procesore, kod prvog sljedećeg restarta operacijskog sustava, gore navedeni kernel modul će provjeriti inačicu trenutnog mikrokôda procesora, te ako je potrebno, napravit će nadogradnju na osnovi navedenih *mikrokôd* datoteka.

Sada provjerimo je li naš trenutni kernel *kompiliran* s podrškom za nadogradnju *mikrokôda*, jer bez ove podrške nadogradnja *mikrokôda* neće biti moguća. To ćemo učiniti pomoću sljedećeg nîza naredbi:

```
grep MICROCODE /boot/config-`uname -r`

CONFIG_MICROCODE=y
CONFIG_MICROCODE_INTEL=y
CONFIG_MICROCODE_AMD=y
CONFIG_MICROCODE_OLD_INTERFACE=y
```

Pošto i za **INTEL** i **AMD** (procesore) imamo `y`, to znači kako je sve u redu te kako će tijekom sljedećeg restarta računala biti uredno učitan *CPU mikrokôd*, ako je to potrebno odnosno, ako postoji novija inačica od one koja se nalazi u *mikrokôd* ROM inačici procesora. Ako želimo ručno napraviti nadogradnju *mikrokôda CPU-a*, to možemo napraviti s naredbom:

```
echo 1 > /sys/devices/system/cpu/microcode/reload
```

Naime ova naredbe će inicirati nadogradnju *mikrokôda CPU-a*, koja će, ako imamo noviju *mikrokôd* datoteku u direktoriju `/lib/fimware/intel-ucode/` (za *Intel* CPU), napraviti nadogradnju.



Kako bi ova metoda radila, kernel također mora biti kompiliran s opcijom: `CONFIG_MICROCODE`.

Ako je sve u redu, vidjet ćemo poruke, poput ovih na kraju ispisa, a ispis smo znatno skratili:

```
dmesg

microcode: CPU0 sig=0xf43, pf=0x10, revision=0x5
microcode: CPU0 updated to revision 0x7, date = 2010-09-28
```

Iste poruke možemo vidjeti i s naredbom: `journalctl` (za ***RedHat/CentOS*** 7.x+ odnosno Linuxe koji koriste *systemd*):

```
journalctl -b | grep -n micro
```

Postoji i još jedna starija, direktna metoda trenutne nadogradnje *CPU mikrokôda*, upotrebom posebne datoteke: `/dev/cpu/microcode`, a za nju nam je potrebna datoteka: `microcode.dat`. Dakle ovo nije novija metoda koja koristi *ucode* format, već stari format *mikrokôda*. Kod ove metode nadogradnje *mikrokôda* napravit ćemo sljedeće:

```
dd if=microcode.dat of=/dev/cpu/microcode bs=1M
```

Za ovu potrebu koristimo poznatu Linux naredbu `dd` (*disk dump*), kojom kopiramo *mikrokôd* datoteku direktno na upravljački program procesora (`/dev/cpu/microcode` koji je zadužen za sâmu nadogradnju *mikrokôda*.

Dodatno, dostupna je još jedna metoda, a to je vrlo rana metoda inicijalizacije *CPU* mikrokôda u kojoj se *CPU mikrokôd* datoteka sprema u `initrd` odnosno u `initramfs` datoteku (ovisno koja se koristi), ali u posebnom formatu.



O ovoj metodi nećemo detaljnije govoriti, a opisana je u dokumentu: (**154**).

**Međutim, ako vam ne radi nadogradnja mikrokôda, ostaje vam još i ova metoda nadogradnje**
Najbrži način izrade *`initramfs`* (za ***RedHat/CentOS/Rocky*** 6.x i 7.x) datoteke, bila bi njegova automatska izrada, pomoću programa `dracut`, koji će u ovu sliku datotečnog sustava (engl. *Init RamFS Image)* uključiti i novi *mikrokod*. Dakle pokrenimo:

```
dracut -f
```

Nakon što se proces izrade *initramfs slike* završi, napravite restart računala. Naime program `dracut` će izraditi novi *initramfs* koji će u inicijalnom dijelu sadržavati i mikrokôd za naš procesor. Možete i provjeriti koja mikrokôd datoteka je dodana u prvi dio *initramfs*-a. Pogledajmo sadržaj *initramfs-a*, pomoću naredbe `lsinitrd` (← *uz naše komentare sa strane*):

```
lsinitrd /boot/initramfs-`uname -r`.img | less
```

```
drwxr-xr-x   3 root  root       0 Jan 11 11:22 .
-rw-r—r--    1 root  root       2 Jan 11 11:22 early_cpio
drwxr-xr-x   3 root  root       0 Jan 11 11:22 kernel       ← ovdje se nalaze kernel moduli (driveri)
drwxr-xr-x   3 root  root       0 Jan 11 11:22 kernel/x86
drwxr-xr-x   2 root  root       0 Jan 11 11:22 kernel/x86/microcode  ←ovdje su mikrokod datoteke
-rw-r—r--    1 root  root   12288 Jan 11 11:22 kernel/x86/microcode/GenuineIntel.bin
```

Ispis smo skratili samo na početni dio u kojemu se nalazi mikrokôd odnosno datoteka: `kernel/x86/microcode/GenuineIntel.bin`.



Za ***Intelove*** procesore ova datoteka je imena: **`kernel/x86/microcode/GenuineIntel.bin`**.
Dok je za ***AMD*** procesore ona imena: **`kernel/x86/microcode/AuthenticAMD.bin`**.

**Izvori informacija: (154),(155),(156), (K-9), `dman dmesg`,`man dmidecode`,`man lsinitrd`,`man dracut`.**

### 10.7.1.2.1. initrd i/ili initramfs i inicijalizacija sustava

Vezano za inicijalni RAM disk odnosno ***initrd*** ili ***initramf****;* on osim već navedenog, sadrži i privremeni *root filesystem* odnosno datotečni sustav s vršnim korijenskim stablom direktorija i određenih sistemskih datoteka. Naime u ranom trenutku pokretanja Linux sustava, nakon pokretanja kernela te *initrd-a* inicijalizira se dio sustava koji primjerice učitava CPU *mikrokôd* i upravljačke programe, ali ima i podršku za razne datotečne sustave i mrežne protokole. Zatim se u dijelu koji se zove *rootfs* kreira korijen stabla vršnih direktorija (vidljiv na ispisu dolje), a tek potom se učitava ostatak sustava.

```
/bin
/dev
/etc
/lib
/proc
/sys
```

Osim navedenih direktorija kreiraju se i ostali direktoriji, ali i važne konfiguracijske datoteke (pr. unutar `/etc` direktorija) kao i izvršne datoteke u direktorijima poput: `/bin`, `/sbin`, `/usr/sbin` i drugih, koje su nam potrebne dok se još nije montirala niti jedna particija diska odnosno datotečni sustav. Ovdje se obično nalaze sve važne konfiguracije, izvršne datoteke i biblioteke, potrebni kernel moduli (upravljački programi) kao i programi i alati koji nam mogu pomoći da u slučaju kada je došlo do nekih grešaka na sistemskom disku, sustav vratimo u upotrebljivo stanje.
Naime i u slučajevima kada sustav prebacujemo u takozvani jednokorisnički način rada (engl. *Single user mode*) za održavanje, obično se uz kernel učitava samo *root filesystem* (***rootfs***) sa svim direktorijima (mapama) i pripadajućim datotekama (izvršne i njihove osnovne biblioteke).

Kod specijaliziranih upotreba Linuxa (tzv. *Embedded* sustavi) poput usmjerivača, preklopnika, sustava automatizacije (automobili, razni strojevi, roboti i industrijska postrojenja, vatrozida i drugih) ovaj *root filesystem* (***rootfs***) može sadržavati cijeli ili gotovo cijeli Linux sa svim direktorijima i potrebnim datotekama na koji se kasnije sâmo prema potrebi mogu montirati drugi diskovi odnosno particije. One mogu sadržavati neke druge sadržaje: podatke, navigacijske karte, multimediju i slično. Važno je razumjeti da se ovaj *root filesystem* (***rootfs***) učitava u RAM memoriju računala, a kasnije se povezuje s montiranim diskovima odnosno particijama na njima, na kojima se nalazi ostatak datotečnog sustava odnosno *embeded* Linuxa.

Međutim, u normalnom radu na standardnim računalima i poslužiteljima (ne *embedded*) nakon što se *rootfs* inicijalizirao i odradio što je potrebno, događa se nešto malo drugačije nego na *embedded* sustavima. Naime namjena *rootfs* sustava je ovdje u svrhu učitavanja dodatnih kernel modula (upravljačkih programa) koji se ne nalaze unutar sâmog kernela s kojima se sustav počeo inicijalizirati, a eventualno su potrebni za kasniji pristup sistemskom disku (*SATA/SAS/SCSI/...*) ili nekom od RAID polja (softverskom ili hardverskom), mrežnoj kartici, mrežnom disku, mrežnim protokolima (*TFTP/NFS/HTTP/...*) i slično.

U sljedećem koraku, se cijeli *rootfs* datotečni sustav, sa svojom strukturom direktorija (mapa) i datoteka:

- Prazni, ako koristimo ***initramfs***, te popunjava s novom strukturom direktorija i datoteka koja se nalazi na sistemskom Linux disku (počevši od vrha / korijenskog direktorija) i nastavlja se s daljim pokretanjem sustava.
- Dok se kod metode kada se koristi ***initrd***, sustav s Linuxovog sistemskog diska montira kao privremeni pa se zamijeni (zarotira) s postojećim *rootfs* sustavom odnosno strukturom direktorija i datoteka te se nastavlja s daljim pokretanjem sustava. ***Initrd*** se koristi i kod većine ***PXE*** instalacija sustava preko mreže, nakon inicijalizacije kernela.

Bez obzira na primjenu (*embedded* ili ne) povezivanje točaka montiranja kao i sve radnje vezane za pristup i rad s datotekama i direktorijima općenito, odrađuje podkomponenta diskovnog sustava zvana ***VFS*** (engl. *Virtual File System*).



Vezano za ***VFS***, pogledajte poglavlje: **14.1. Diskovni ulazno/izlazni sustav (*I/O*).**
Za ručno kreirani ***RAM** disk* pogledajte poglavlje: **12.6.2. Upotreba RAM diska.**

**Slijedi više detalja (napredna cjelina) !**

## Initramfs i kernel

Podsjetimo se procesa pokretanja računala (malo detaljnijeg):

***BIOS → GRUB2 stage 1 →GRUB2 stage 1.5 →GRUB2 stage 2 → Kernel → [initramfs ⇒ init skripta (init ili systemd)] ⇒ → Montiranje datotečnog sustava → učitavanje servisa i programa***

Nakon što BIOS prepozna hardver te pronađe i odabere disk s kojeg će pokrenuti sustav, s prvog sektora diska odnosno *master boot recorda* (**MBR**) učitava se prva faza *GRUB2 boot loadera*, Ona potom najčešće pokazuje na lokaciju *GRUB2 boot loadera faze 1.5* (ponekad se odmah koristi *GRUB2 faza 2*) koji se nalazi na sektorima diska između **MBR** i **VBR** zapisa.
*GRUB2 boot loader faze 1.5* tada pomoću svojih mehanizama s aktivne (formatirane) particije, pomoću zapisa koji se zove *volume boot record* (**VBR**) pronalazi drugu fazu *GRUB 2 boot loadera* (*stage 2*) iz direktorija `/boot/`.
Odnosno učitava sve *GRUB2* potrebne datoteke iz direktorija: `/boot/grub2/`.
Ovaj završni dio *GRUB2 (faza 2)* s datotečnog sustava na označenoj aktivnoj (*boot*) particiji učitava kernel.



Vezano za detalje oko *boot* procesa, pogledajte poglavlja:
**12.1. Što se događa kod pokretanja računala (*boot process*).**
**12.1.2. Boot proces detaljnije.**

Kernel potom uobičajeno učitava ili ***initrd*** (stariji sustavi) ili ***initramfs*** (noviji sustavi [***Red Hat*** *7+*]).
***Initramfs*** pruža minimalno Linux okruženje koje omogućuje izvođenje programa prije nego što se montira stvarni korijenski (/) datotečni sustav i pokrene ostatak operativnog sustava.

*Initramfs* sadrži svoj inicijalni korijenski datotečni sustav u posebnoj datoteci koja je zapravo diskovna slika (*disk image*).
Važno je razumjeti da je **Initramfs** nasljednik starijeg **Initrd** formata.

**Pokretanja računala od kernela na dalje – manje uobičajena metoda**
Tijekom pokretanja računala u trenutku kada se učita kernel, on inicijalizira hardver pomoću ugrađenih upravljačkih programa te montira navedenu korijensku particiju. *Init* sustav odnosno servis (`init` ili `systemd`) tada pokreće ostatak sustava, i prema potrebi se učitavaju dodatni kernel moduli (upravljački programi) i pokreću preostali servisi te u konačnici korisnički programi. Ovakvo ponašanje odnosno način rada bi bio dovoljan za određene primjene, ali obično se koristi nešto drugačiji način rada.

**Pokretanja računala od kernela na dalje - uobičajena metoda**
Na većini distribucija Linuxa, nakon učitavanja kernela, prvo se učitava ***initramfs*** diskovna slika. U većini slučajeva prije nego što se korijenska (*root*) particija montira i nastavi s pokretanjem ostatka sustava, potrebno je obaviti određene stvari.
Stoga se učitava *initramfs* diskovna slika i pokreću određene komponente unutar nje.

Neke od stvari koje omogućuje *initramfs* su:

- Učitavanje CPU *mikrokoda* i dodatnih kernel modula (upravljačkih programa) koji nisu integrirani u postojeći kernel:
  - Primjerice za disk kontrolere (i pristup njihovim diskovima) potrebne za nastavak procesa pokretanja sustava.
  - Za mrežne kartice i bilo koji drugi hardver.
- Pružanje minimalističkog naredbenog retka za spašavanje sustava (ako nešto pođe po zlu). Dodatno omogućuje i dodavanje raznih programa i biblioteka. Primjerice dodavanje uslužnih programa koji su potrebni ili dobrodošli prilikom spašavanja/popravljanja ili eventualnog testiranja sustava.
- Omogućuje montiranje particija koje su kriptirane.
- Omogućuje razne optimizacije i prilagodbe procesa pokretanja računala.
- I na kraju omogućuje sve druge radnje koje sâm kernel nije u mogućnosti odraditi

## Malo više detalja o initramfs-u

U našem primjeru (za kernel 6.7.6.x) datoteka odnosno diskovna slika *initramfs*-a je sljedeća datoteka: `initramfs-6.7.6-1.el9.elrepo.x86_64.img`.
Ova datoteka je arhiva diskovne slika napravljena s *CPIO* te komprimirana s *gzip* algoritmom.
Pogledajmo detalje o ovoj arhivi (datoteci):

```
file /boot/initramfs-6.7.6-1.el9.elrepo.x86_64.img
initramfs-6.7.6-1.el9.elrepo.x86_64.img: gzip compressed data, max compression.
```

← vidimo da se radi o *GZIP* arhivi, iako je ponekad lučaj da se ona komprimira i sa *xz/LZMA* algoritmom.

Kopirat ćemo ovu *initramfs* datoteku u naš testni direktorij:

```
mkdir /root/initramfs
cp /boot/initramfs-6.7.6-1.el9.elrepo.x86_64.img /root/initramfs
cd /root/initramfs
```

I sada ju možemo dekomprimirati i provući kroz `cpio` program (jer je s njim i kreirana):

```
zcat initramfs-6.7.6-1.el9.elrepo.x86_64.img | cpio -idmv
```



**Cpio** je kratica za "*copy in, copy out*".
Koristi se za obradu arhivskih datoteka poput `*.cpio` ili `*.tar`.
Ova naredba može kopirati datoteke u i iz arhive.

S prijašnjom naredbom smo dobili novu strukturu direktorija i datoteka iz *initramfs* arhive odnosno diskovne slike:

```
lrwxrwxrwx  1 root root   7 Oct 28 06:57 bin -> usr/bin
drwxr-xr-x  2 root root  74 Mar 13 10:08 dev
drwxr-xr-x 13 root root 4096 Mar 13 10:08 etc
lrwxrwxrwx  1 root root  23 Oct 28 06:57 init -> usr/lib/systemd/systemd
lrwxrwxrwx  1 root root   7 Oct 28 06:57 lib -> usr/lib
lrwxrwxrwx  1 root root   9 Oct 28 06:57 lib64 -> usr/lib64
drwxr-xr-x  2 root root   6 Oct 28 06:57 proc
drwxr-xr-x  2 root root   6 Oct 28 06:57 root
drwxr-xr-x  2 root root   6 Oct 28 06:57 run
lrwxrwxrwx  1 root root   8 Oct 28 06:57 sbin -> usr/sbin
-rwxr-xr-x  1 root root 4631 Jun 20  2022 shutdown
drwxr-xr-x  2 root root   6 Oct 28 06:57 sys
drwxr-xr-x  2 root root   6 Oct 28 06:57 sysroot
drwxr-xr-x  2 root root   6 Oct 28 06:57 tmp
drwxr-xr-x  8 root root  81 Mar 13 10:08 usr
drwxr-xr-x  3 root root  40 Mar 13 10:08 var
```

← Ovdje vidimo cijelo stablo direktorija i datoteka.

- Primjerice u direktoriju `/bin` se nalaze binarne izvršne datoteke (pr. `bash`, `chmod`, `chown`, `grep`, `gzip`, `kmod`, `mount`, `udevadm`, … ).
- Dok su u direktoriju `/lib` (pokazuje na `/usr/lib`) razne programske biblioteke, ali i kernel moduli i sistemske binarne izvršne datoteke.
- U direktoriju `/usr` se nalaze i razni programi tako dalje.

U ovom stablu direktorija i datoteka uočite datoteku imena `init`.

Osim što ona (datoteka imena `init`) mora ovdje postojati, moguće su i dvije opcije:

- Ona može biti simbolički link kao u ovom slučaju u kojem je ona simbolički link na `systemd` servis (od **Red Hat 7+**).
- Ona može biti i jednostavna skripta koja uobičajeno nastavlja s inicijalizacijom sustava

## O init-u unutar initramfs-a

Nakon učitavanja kernela, kernel provjerava sadrži li *initramfs* datoteku imena `init`, i ako postoji, izvršava ju.
Dakle *Init* servis (ili skripta) unutar *initramfs-a* je prva komponenta koja se pokreće prilikom pokretanja operativnog sustava.
Njena osnovna funkcija je pripremiti okruženje za pokretanje pravog operativnog sustava i montiranje korijenskog datotečnog sustava sa svim programima, servisima i svim ostalim datotekama.
Pogledajmo nekoliko osnovnih zadataka koje obavlja *init* komponenta unutar *initramfs*-a:

- Inicijalno montiranje `/proc` i `/sys` (*sysfs*), `/dev` (*devtmpfs*) i `/run` (*tmpfs*) pseudo datotečnih sustava.
- Učitavanje potrebnih kernel modula - kako bi se osigurala podrška za različite hardverske uređaje i značajke potrebne za daljnji rad sustava. Naime prije nego što se korijenski datotečni sustav uopće može montirati i pokrenuti operativni sustav, kernel treba odgovarajuće upravljačke programe za pristup (diskovnom) uređaju na kojem se nalazi korijenski datotečni sustav. Ovi upravljački programi mogu uključivati posebne upravljačke programe za određene vrste tvrdih diskova ili čak mrežne upravljačke programe za pristup mrežnim datotečnim sustavima; pooput primjerice: *NFS, CEPH, iSCSI* i sličnih.
- Montiranje datotečnih sustava - inicijalizacija datotečnih sustava, uključujući montiranje korijenskog (/) datotečnog sustava.
- Konfiguracija mreže i drugih sustavnih postavki - može konfigurirati mrežu, postaviti lokalne varijable okoline, pokrenuti servise i postaviti druge sustavne konfiguracijske opcije koje su potrebne za daljnji rad sustava.
- Pokretanje ostatka sustava - nakon što su sve pripreme gotove, pokrenuti ***init*** ili ***systemd*** (od Red Hat 7+), koji je roditeljski proces za sve ostale procese u sustavu, potom pokreće sve preostale servise i programe.
- Prikazivanje poruka o pokretanju - može prikazati korisne poruke o pokretanju, poput ispisivanja logova na zaslon ili u log datoteke.

## Initramfs + Squashfs

Neke izvedbe *initramfs-a* unutar diskovne slike imaju i dodatnu **squashfs** diskovnu sliku unutar sebe.
Primjerice na sustavu uobičajeno postoji i *initramfs* diskovna slika (datoteka) s **KDUMP** mehanizmom, ako smo omogućili *KDUMP* značajku. U našem slučaju, za našu inačicu kernela, naziv ove datoteke je sljedeći:
`initramfs-6.7.6-1.el9.elrepo.x86_64kdump.img`.
Ova *initramfs* diskovna slika se koristi prilikom *KDUMP* metode rada (analize sustava), ako smo tako izričito odabrali.

U tom slučaju mehanizam pokretanja računala je sljedeći:

*BIOS → GRUB2 stage 1 → GRUB2 stage 1.5. → GRUB2 stage 2 → Kernel → **initramfs.img** ⇒ init shell skripta→ montiranje **squash-root.img** ⇒ init servis (**systemd**) → montiranje datotečnog sustava (/) → bazični servisi i programi*

Vezano za *squashfs*, potrebni programi se nalaze u programskom paketu `squashfs-tools`, koji možemo instalirati:

```
yum -y install squashfs-tools
```

Pogledajmo kako dekomprimirati *squashfs* datoteku odnosno *squashfs* diskovnu sliku, ako to baš želimo:

```
unsquashfs -d /odredišni-direktorij datoteka.squashfs
```



Prilikom dodavanja novog hardvera u računalo ili nakon promjene važnijih sistemskih konfiguracijskih datoteka koje se koriste u početnoj fazi procesa pokretanja računala, kao i kada se mijenjaju opcije vezane za kernel module (upravljačke programe), možda će biti potrebno ponovno izgraditi *initramfs* (ili *initrd*), kako bi uključio navedene promjene.

## Kreiranje initramfs-a

Na **Red Hat** kompatibilnim distibucijama *initramfs* se izgrađuje i kreira pomoću `dracut` komponente.
Naime `dracut` (naredba) se koristi za stvaranje *initramfs* slike kopiranjem programa i raznih datoteka s aktivnog (instaliranog) sustava pomoću *dracut* definicija, koje se obično nalaze u `/usr/lib/dracut/` direktoriju.
Ostale datoteke dracut sustava su:

- `/usr/lib/dracut/` - ovdje se nalaze postavke i finkcije dracut sustava.
- `/usr/lib/dracut/dracut.conf.d/` - ovdje se nalaze specifični dijelovi konfiguracije (pr. specifične za distribuciju Linuxa, CPU mikrokod i sl.).
- `/usr/lib/dracut/modules.d/` - ovdje se nalaze moduli s postavkama *dracut* sustava (pr. `systemd-initrd`, `systemd-jurnald`, `nss`, `watchdog`, …) ..

Za najjednostavnije ponovno rekreiranje *initramfs*-a pogledajte prethodno poglavlje ili pokrenite naredbu:

```
dracut -d
```

Odnosno za ispis sadržaja trenutne inačice *initramfs*-a, možemo pokrenuti naredbu:

```
lsinitrd /boot/initramfs-`uname -r`.img | less
```

**Izvori informacija: (789),(790),(791),(792),(1609),(1610),(1611),(1612),(1613),(1614),(1621),** **`man dracut`**, **`man dracut.conf`**,**`man zcat`**,**`man cpio`**,**`man unsquashfs`**.

#### 10.7.1.2.2. Malo više detalja o mikrokod datotekama

Dalje u tekstu ćemo govoriti o *Intelovim* procesorima, te ćemo koristiti program: `iucode_tool`.
Kako smo spomenuli, u paketu `microcode_ctl` (***RedHat/CentOS*** 7.x), nalazi se jedino izvršni program `intel-microcode2ucode` (uz *Intelove* mikrokôd datoteke).

S ovim programom možemo primjerice provjeriti koje su sve inačice i revizije *mikrokôd* datoteka unutar direktorija: `/lib/modules/intel-ucode`. Stoga provjerimo koje je trenutno stanje:

```
intel-microcode2ucode

intel-ucode/06-7a-01
signature: 0x706a1
flags:     0x01
revision:  0x22
date:      2017-12-26
size:      73728
```

Ispis je skraćen jer se za svaku mikrokôd datoteku dobiva statistika. U novije vrijeme, za *Intel CPU-ove*, sve više se koristi alat: `iucode-tool` koji nudi i dodatne funkcionalnosti.
Ovaj alat dolazi standardno sa **RedHat/CentOS** distribucijama inačice 7.x.+.
Ako ovaj softverski paket nije instaliran, možete ga instalirati pomoću sljedeće naredbe:

```
yum install iucode-tool
```



Više informacija o `iucode-tool` programu, možete vidjeti na poveznici:
https://gitlab.com/iucode-tool/iucode-tool/blob/master/README.

Mi ćemo stoga skinuti zadnju inačicu *mikrokôd* datoteka sa službenih stranica proizvođača (*Intel*).
Kako uopće vidjeti koju inačicu *mikrokôda* procesora imamo, vidjet ćemo pomoću naredbe `iucode_tool`:

```
iucode_tool -l -S

iucode_tool: system has processor(s) with signature 0x00000f43
```

Dakle inačica našeg *mikrokôda* procesora je: *0x00000f43*

Sada možemo iz datoteke `microcode.dat` pronaći koja je zadnja inačica *mikrokôda*, koju smo kopirali sa stranica proizvođača procesora (*Intel* konkretno):

```
iucode_tool -l -t d /lib/firmware/microcode.dat | grep 0x00000f43

094: sig 0x00000f43, pf mask 0x9d, 2005-04-21, rev 0x0005, size 2048
```

Dobili smo odgovor kako je zadnji *mikrokôd* (koji smo skinuli), revizije: `0x0005`.

Ako pak želimo izlistanje svih *mikrokodôva* koji se nalazi u datoteci, sâmo pokrenimo:

```
iucode_tool -l -t d /lib/firmware/microcode.dat
```

Odnosno, ako želimo da nam ovaj alat prođe sve *ucode* datoteke i ispiše inačicu svake:

```
iucode_tool -l -t a /lib/firmware/intel-ucode/
```

Kako provjeriti postoji li noviji *mikrokôd* (primjer je za noviji *Intel CPU*):

```
iucode_tool -tb -lS /lib/firmware/intel-ucode/
iucode_tool: system has processor(s) with signature 0x000306f2
selected microcodes:
  076/001: sig 0x000306f2, pf_mask 0x6f, 2016-03-28, rev 0x0038, size 32768
  015/001: sig 0x000306f4, pf_mask 0x80, 2016-06-07, rev 0x000d, size 15360
```

Na kraju ovog izlistanja je vidljivo:

- Da imamo trenutni *mikrokôd* inačice: `0x0038` (predzadnji red) te da je datum njegovog izdavanja 2016-03-28.
- Te da postoji nadogradnja *mikrokôda*, inačice: `0x000d`, a datum njegovog izdavanja je 2016-06-07.

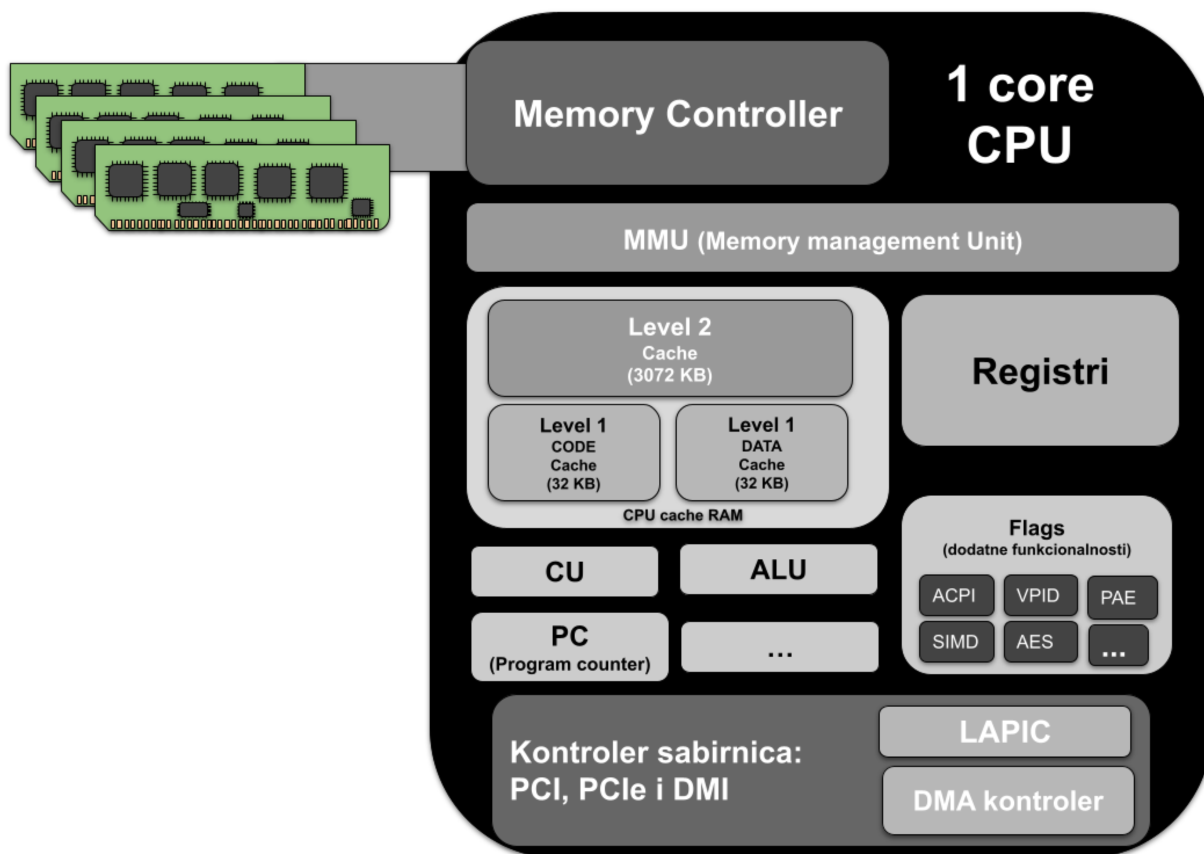
**Izvori informacija: (145),(146),(153),(154),(155),(156),** `man intel-microcode2ucode`,`man iucode_tool`.

### 10.7.1.3. CPU registri još detaljnije

Cijeloj priči oko CPU registara moramo dodati još jednu komponentu, a to je priručna vrlo brza RAM memorija unutar procesora, koja se nalazi između registara i (vanjske) RAM memorije, te ostatka *CPU*a.

Pogledajmo pojednostavljeni model procesora s jednom jezgrom (slika 59.) jer procesori s više jezgri imaju još kompleksniji model s kojim bi sada zakomplicirali priču.

**Slika 59. Detaljan pogled na dizajn mikroprocesora i njegove logičke cjeline**




Na putu između **RAM** memorije (tzv. SRAM) i ostatka procesora, nalaze se vrlo brze priručne memorije u nekoliko razina. Dakle ove memorije su ugrađene u sâm procesor, a ovisno o njegovom dizajnu, postoji ih nekoliko razina:

- Najbrža i najmanja je takozvana **Layer 1** (**L1**) memorija, podijeljene u dva dijela, jedan dio je za instrukcija (zvan ***ITLB1***), a drugi dio je za podatke (zvan i ***DTLB1***).
- Malo sporija je takozvana **Layer 2** (**L2**), ali i znatno veća memorija, koja se dijeli između više **Layer 1** memorija.
- Kod višejezgrenih procesora tu je i takozvana **Layer 3** (**L3**) memorija koja se dijeli između svih jezgri, dok su **Layer 1** i **Layer 2** memorije pozicionirane unutar svake zasebne fizičke jezgre procesora (i svih njenih logičkih jezgri).



Za detalje vezane uz višejezgrene procesore i njihove međumemorije (**L1**, **L2** i **L3**) pogledajte poglavlje**:** **10.7.2.1. SMP.**

Dizajn vašeg procesora i njegovih priručnih memorija možete vidjeti (u *ljusci*) s naredbom `lstopo` na sljedeći način:

```
lstopo --whole-system --no-io --of txt
```

Isto možemo vidjeti i s naredbom `lstopo-no-graphics` na sljedeći način:

```
lstopo-no-graphics --whole-system --no-io --of txt
```

Osim toga obje naredbe mogu rezultat snimiti kao ***PNG*** ili ***PDF***.
Ako nemate ovu/ove naredbe, možete ih instalirati sa:

```
yum -y install hwloc hwloc-gui
```

Sva ova priručna RAM memorija služi kao međuspremnik između **RAM** memorije (koja je za sâm CPU prilično spora) i ostalih komponenti *CPU*a. Posebni mehanizmi su zaduženi za spremanje podataka u ove međuspremnike (**Layer 1** i **Layer 2**) jer je potrebno znati što čuvati u ovoj memoriji, a što ne. Svaki put kada traženi podatak (ili instrukcija) nije sačuvana u ovim priručnim memorijama (prvo u **L1**, pa u **L2** te potom **L3**), potrebno ga je dohvatiti i učitati iz **RAM** memorije, što naravno troši određeni dio resursa. Pošto su ove priručne memorije unutar procesora drastično brže nego **RAM** memorija, njihova je upotreba itekako očita i opravdana.

## Što se događa kod dohvaćanja sadržaja iz (RAM) memorije?

Prvo se podaci pokušavaju dohvatiti iz **Layer 1** memorije unutar procesora, a ako dođe do promašaja (ako podataka ovdje nema), pokušavaju se dohvatiti iz **Layer 2** memorije i učitati u **Layer 1** memoriju procesora. Ako traženi podatak nije niti u **Layer 2** memoriji procesora, tada se pokušava dohvatiti iz **Layer 3** memorije (kod višejezgrenih procesora). I na kraju, ako podatak nije niti ovdje, podatak se tek tada dohvaća iz **RAM** memorije.
Ovisno a arhitekturi procesora **Layer 3** memorija se koristi i/ili kao **TLB** memorija ili se **TLB** memorija nalazi unutar **MMU (**engl. *Memory Management Unit***)** komponente koja je dio memorijskog kontrolera procesora.



Za više detalja o ***TLB*** memoriji, pogledajte poglavlje:
**12.3.4 Zbog čega toliko priče o *Page Table-u*.**

O kojim se brzinama i kašnjenjima u dohvaćanju podataka radi u prosjeku, za x86 arhitekturu na *Intel Core i7 Xeon 5500 procesoru*, pogledajmo okvirne pokazatelje u tablici:

| **Lokalno (unutar CPUa) ili udaljeno (NUMA)** | **Memorija kojoj se pristupa** | **Koliko je CPU ciklusa potrebno** | **Koliko ns je potrebno (najlošije - najbolje)** |
|---|---|---|---|
| Lokalno (unutar CPUa) | L1 | 4 ciklusa | 2.1 - 1.2 ns |
| Lokalno (unutar CPUa) | L2 | 10 ciklusa | 5.3 - 3.0 ns |
| Lokalno (unutar CPUa) | L3 - direktno dohvaćeno | 40 ciklusa | 21.4 - 12.0 ns |
| Lokalno (unutar CPUa) | L3 - dijeljeno s drugom jezgrom | 65 ciklusa | 34.8 - 19.5 ns |
| Lokalno (unutar CPUa) | L3 - modificirano na drugoj jezgri | 75 ciklusa | 40.2 - 22.5 ns |
| **Udaljeno (NUMA)** | L3 - NUMA - udaljeni NUMA CPU | 100-300 ciklusa | 160.7 - 30.0 ns |
| Lokalno (unutar CPUa) | DRAM | | 60 ns |
| **Udaljeno (NUMA)** | DRAM - NUMA - udaljeni NUMA CPU | | 100 ns |

**Izvori informacija za ovu cjelinu: (304),(305),(306),(307), `man lstopo`.**

Pogledajmo i objašnjenje iz prethodne tablice: CPU ciklusi označavaju radni takt procesora.
Pogledajmo i što znače oznake takta procesora:

| **Takt procesora (MHz/GHz)** | **Opis** |
|---|---|
| 1 MHz | 1 000 000 Hz (ciklusa u sekundi) |
| 10 MHz | 10 000 000 Hz (ciklusa u sekundi) |
| 100 MHz | 100 000 000 Hz (ciklusa u sekundi) |
| 1000 MHz (1 GHz) | 1 000 000 000 Hz (ciklusa u sekundi) |
| 2000 MHz (2 GHz) | 2 000 000 000 Hz (ciklusa u sekundi) |



Za malo više detalja vezanih uz memoriju, pogledajte i poglavlje**:**
**12. Sustav virtualne memorije i memorijski menadžment.**

Kako bi vam bili jasniji odnosi brzina pristupa odnosno vremena potrebnih za razne operacije, pogledajmo i tablicu koja prikazuje vrijeme potrebno za pristup ili dohvaćanje, odnosno obradu podataka:

```
           0.5 ns - CPU - referenciranje L1 dCACHE memorije
           1   ns - pri brzini svjetlosti: foton prolazi udaljenost od 30.5cm
           5   ns - CPU L1 iCACHE Branch mispredict
           7   ns - CPU - referenciranje L2  CACHE memorije
          30   ns - sistemski poziv na obradu podataka
                      (ovo vrijeme varira od 30ns do 90ns ovisno o CPU)
         100   ns - MUTEX lock/unlock (novije implementacije CPUa - 17 ns)
         100   ns - Referenciranje DDR memorije (RAM)
      10,000   ns - Komprimiranje 1KB korištenjem ZIP algoritma
      20,000   ns - Slanje 2KB preko 1 Gbps (1000 Mbps) mreže
     250,000   ns - Sekvencijalno čitanje 1 MB podataka iz RAM memorije
     500,000   ns - Odziv mreže (kašnjenje) unutar podatkovnog centra
  10,000,000   ns - Pristup tvrdom disku (DISK seek)
  10,000,000   ns - Sekvencijalno čitanje 1 MB s mreže
  30,000,000   ns - Sekvencijalno čitanje 1 MB s tvrdog diska
 150,000,000   ns - Slanje mrežnih paketa iz SAD-a u Europu
|   |   |   |
|   |   | ns|
|   | µs|
| ms|
```

Objašnjenje nekih od navedenih pojmova slijedi:

- `sistemski pozivi` – operativni sustav mora nekako reagirati kod primanja svakog zahtjeva za obradu podataka. Ovdje se radi o vremenu potrebnom kako bi se uopće odradio određeni sistemski poziv i pokrenula cijela procedura obrade. Nerijetko je ovo vrijeme u granicama između 40ns i 90ns.
- `MUTEX lock/unlock` se odnosi na proces zaključavanja odnosno otključavanja radne nîti (*thread*a) unutar programa, zbog dalje obrade od strane *CPU*a.



Za više detalja, pogledajte poglavlje:
**9.3.1 Context switching**

- `branch predictor` je dio *CPU*a odnosno logika unutar njega, koja pokušava predvidjeti grananje programa, odnosno smjerove grananja ovisne o programiranim uvjetima unutar sâme logike programa. Dakle pojednostavljeno uvjeti poput: **`if – then - else`** i slično. Naime moderni procesori tijekom izvršavanja programa, a s obzirom na to kako imaju nekoliko *cjevovoda* za izvršavanje (engl. *Pipelines*), pokušavaju unaprijed predvidjeti grananje programa, ne bi li se ono moglo što brže odraditi. Dakle `CPU L1 iCACHE Branch mispredict` se odnosi na promašaj u predviđanju grananja, te ponovno (novo) dohvaćanje odnosno vrijeme potrebno kako bi se to odradilo.

---

## Vrijeme obrade mrežnih paketa

**Pogledajmo detalje vezane uz CPU, memoriju i brzinu obrade podataka, kroz primjer obrade mrežnih paketa.**
Sada uzmimo primjer u kojem procesor mora obraditi mrežne pakete, kako ih mrežna kartica zaprima. Ako govorimo o brzinama od 1Gbps, tada za pakete veličine 64 bajta (najmanji mrežni paketi) imamo 1.488.095 paketa u sekundi, koje treba obraditi.
To znači da dobivamo vremenski okvir od 672ns unutar kojeg se svaki pojedini mrežni paket mora obraditi.

Pogledajte odakle nam ova brojka:

$$\frac{1}{1\ 488\ 095\ x\ 1\ 000\ 000\ 000} = 672 \text{ ns}$$

Ako s druge strane govorimo o većim brzinama poput primjerice 10Gbps tada imamo sljedeće stanje:

$$\frac{1}{14\ 880\ 952\ x\ 1\ 000\ 000\ 000} = 67{,}2 \text{ ns}$$

To znači, da procesoru koji radi na taktu od 2GHz, jedan takt (engl. *clock*) traje **0,5ns**.

Prema tome ovisno o brzini mreže imamo sljedeće stanje, vidljivo u tablici:

| Brzina mreže | Maksimalno vrijeme obrade | Broj taktova/ciklusa CPU (CPU radnog takta 2GHz) unutar kojih se mora odraditi svaki paket |
|---|---|---|
| 1 Gbps | 672 ns | 1.340 ciklusa |
| 10 Gbps | 67,2 ns | 134 ciklusa |
| 40 Gbps | 16,8 ns | 33,5 ciklusa |
| 100 Gbps | 6,7 ns | 13,4 ciklusa |
| 200 Gbps | 3,35 ns | 6,7 ciklusa |
| 400 Gbps | 1,68 ns | 3,35 ciklusa |

Prema navedenoj tablici, za brzinu (zapravo propusnost) mreže od 1 Gbps, procesor ima vremena 672ns da bi obradio svaki pojedini mrežni paket. Povećanjem brzine mreže na 10Gbps to se vrijeme smanjuje na samo 67,2ns unutar kojih se svaki mrežni paket mora stići obraditi.

Dakle kod mreža koje rade na 10Gbps za procesor koji radi na 2GHz, svaki mrežni paket mora se obraditi unutar 134 takta procesora, što je vremenski okvir od 67,2ns. Za još veće brzine (40 Gbps i 100 Gbps) problemi postaju sve veći i veći.



Cijelo vrijeme govorimo o najmanjim mogućim mrežnim paketima od **64** bajta jer je rad s njima najzahtjevniji.

Zbog razumijevanja ove problematike, moramo znati kako se unutar vremenskih okvira u kojima se svaki paket mora obraditi događaju i druge stvari:

- Prvo sâm operativni sustav mora nekako reagirati tijekom zaprimanja svakog mrežnog paketa. Dakle na sâmom početku dolazi do pozivanja određenog sistemskog poziva na osnovu kojega dolazi do bilo koje akcije operativnog sustava. Ovdje se radi o vremenu potrebnom kako bi se uopće odradio određeni sistemski poziv i pokrenula cijela procedura. Nerijetko je ovo vrijeme u granicama između 40ns i 90ns.
- Pokušavaju se dohvatiti podaci priručne (*cache*) memorije procesora:
    - *L1 cache*: oko 5ns.
    - *L2 cache*: oko 5-7ns.
    - *L3 cache*: oko 7-9ns.
- U slučaju kada podaci nisu u priručnoj memoriji procesora (*CPU*), dolazi do promašaja (*cache miss*), te se moraju ponovno dohvatiti: ova vremena mogu biti između 10ns i 40ns (i više). Primjerice za *Intel Xeon E5-2650*, ovo vrijeme je 32ns.
- Dodatno za prebacivanje na određeni proces, događa se ***lock*** odnosno ***mutex*** operacija kojoj treba od 8ns do 17ns (trebamo dvije ovakve operacije za svaki paket). Recimo kako je to ovdje ukupno 17ns.
- I sada na kraju konačno kreće obrada jednog paketa te potom iskorištavanje onog drugog ***mutexa.***
- … dolazi nam drugi mrežni paket → sve kreće iz početka.

Ovdje smo načeli i problematiku obrade mrežnih paketa na brzinama od 10Gbps. Zbrojite sve i provjerite može li se sve odraditi unutar 67,2ns za svaki mrežni paket.



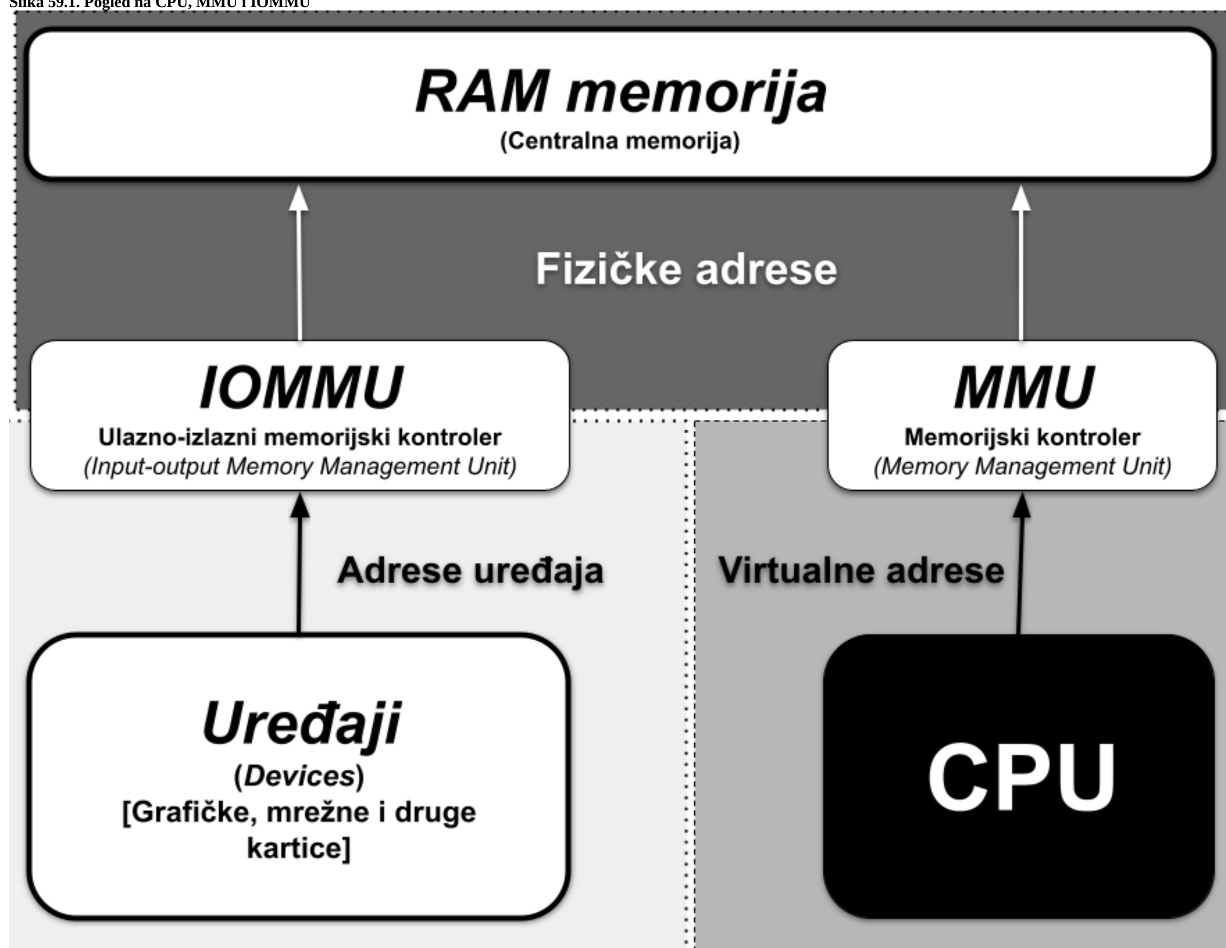
Za detalje obrade mrežnih paketa, pogledajte i poglavlja:
**20.4. *Gbps* i *Mpps* – u čemu je veza**.
**25.5.6. Optimizacije mrežnih sučelja propusnosti veće od 10Gbps.**

**Izvori informacija: (304),(305),(306),(307),(K-9)**

### 10.7.1.4. Ulazno-izlazni memorijski kontroler (IOMMU)

Ulazno-izlazni memorijski kontroler odnosno jedinica za upravljanje memorijom znana je kao **IOMMU (***engl. input–output memory management unit*) i ona je komponenta koja povezuje ulazno/izlaznu (I/O) sabirnicu s mogućnošću izravnog pristupa memoriji; dakle sposobnu za **DMA** prijenos podataka s glavnom (RAM) memorijom. Poput tradicionalnog MMU-a, koji pretvara virtualne adrese vidljive CPU-u u fizičke adrese, *IOMMU* preslikava virtualne adrese vidljive uređaju (koje se u ovom kontekstu nazivaju i adrese uređaja ili I/O adrese) u fizičke adrese RAM memorija. Neke *IOMMU* jedinice također pružaju zaštitu memorije od neispravnih ili zlonamjernih uređaja. Jedan od primjera upotrebe ovakvog rada je upotreba tablica translacije grafičkih adresa (**GART**) koju koriste ***AGP*** i ***PCI Express*** grafičke kartice na *Intel* i *AMD* arhitekturi računala (**x86**). Kod starijih x86 arhitektura procesora (CPUa), prije podjele funkcionalnosti sjevernog i južnog mosta između CPU-a i *Platform Controller Hub-a* (***PCH***), ovu I/O virtualizaciju nije izvodio CPU, već čipset.
Zbog lakšeg razumijevanja, oslanjamo se na ovakav dizajn, kako je vidljivo na slici **59.1.**

**Slika 59.1. Pogled na CPU, MMU i IOMMU**




Upotrebom ***IOMMU*** u odnosu na klasični ***DMA*** pristup, dobivaju se mnoge koristi, od kojih ćemo nabrojati samo neke:

- Moguće je dodjeljivati velike regije memorije, bez potrebe da budu uzastopne (u nizu), u fizičkoj memoriji.
- Uređaji koji ne podržavaju dovoljno velike memorijske adrese za adresiranje cijele fizičke memorije još uvijek mogu adresirati cijelu memoriju putem IOMMU-a. Primjerice uređaji na 32-bitnoj PCI sabirnici koji ne mogu adresirati više o 4GB RAMa.
- Memorija je zaštićena od zlonamjernih uređaja koji pokušavaju izvesti **DMA** napade i neispravnih uređaja koji namjerno pokušavaju izvesti pogrešne prijenose podataka memorije jer uređaj ne može čitati ili pisati u memoriju koja mu nije eksplicitno dodijeljena (mapirana). Zaštita memorije temelji se na činjenici da operativni sustav koji se izvršava na CPU-u isključivo kontrolira i ***MMU*** i ***IOMMU***. Stoga periferni (drugi) uređaji fizički ne mogu zaobići ili prepraviti tablice upravljanja memorijom.
- U ***virtualizaciji***, operativni sustavi unutar virtualnog računala mogu koristiti hardver koji nije posebno napravljen za virtualizaciju. Međutim hardver koji zahtjeva više performanse kao što su grafičke i mrežne kartice, koriste **DMA** za izravan pristup memoriji. Pošto u virtualnom okruženju sve memorijske adrese ponovno preslikava sustav unutar virtualnog računala, to uzrokuje probleme u **DMA** načinu rada. ***IOMMU*** upravlja ovim ponovnim preslikavanjem (translacijom/mapiranjem adresa) memorije, dopuštajući da se izvorni upravljački programi uređaja koriste unutar operativnog sustava virtualnog računala, što može drastično ubrzati njihov rad.
- U nekim arhitekturama ***IOMMU*** također izvodi ponovno preslikavanje hardverskih signala prekida (*IRQ*), na način sličan standardnom ponovnom preslikavanju memorijskih adresa. …

## Primjena u virtualizaciji
Kada operativni sustav radi unutar virtualnog računala, uključujući sustave koji mogu koristiti paravirtualizaciju, kao što su ***Xen*** i **Linux KVM**, on obično ne poznaje fizičke adrese RAM memorije računala (*hipervizora*) kojoj pristupa. To otežava pružanje izravnog pristupa hardveru računala, jer ako bi gostujući operativni sustav (VM) pokušao dati instrukcije hardveru da izvrši izravan pristup memoriji (pomoću ***DMA***) koristeći fizičke adrese (RAM memorije) korištene unutar virtualnog računala, vjerojatno bi oštetio podatke u memoriji, jer emulirani hardver unutar virtualnog računala nije svjestan preslikavanja između fizičke adrese gosta (VM) i fizičke adrese domaćina (*hipervizor*) za dano virtualno računalo. Oštećenje se može izbjeći, ako *hipervizor* ili njegov operativni sustav interveniraju u I/O operacije za primjenu prevođenja memorijskih adresa. Međutim, ovaj (standardni) pristup uvodi kašnjenje u I/O operacijama. ***IOMMU*** rješava ovaj problem ponovnom translacijom adresa kojima pristupa hardver prema istoj (ili kompatibilnoj) tablici prevođenja koja se koristi za translaciju (prevođenje) fizičke adrese unutar virtualnog računala u fizičke adrese fizičkog računala (*hipervizora*). Podršku za ***IOMMU*** odnosno *IOMMU* ugrađen u hardver (obično u procesor), imaju svi proizvođači procesora i sve arhitekture procesora. Pogledajmo neke od njih:

- Tvrtka ***Intel*** ju naziva **VT-d** (*Intel Virtualization Technology for Directed I/O*), a podrška postoji od **Core 2** procesora, od 2006.g.
- Tvrtka ***AMD*** ju naziva **AMD-Vi** (*I/O Virtualization Technology*).
- **ARM** arhitektura procesora ju naziva **SMMU** (*System Memory Management Unit*).

Međutim osim podrške CPU-a, i čipset matične ploče i ***BIOS*** ili ***UEFI*** moraju u potpunosti podržavati IOMMU I/O virtualizacijsku funkcionalnost, da bi ona bila upotrebljiva. Samo PCI ili PCI Express uređaji koji podržavaju takozvani ***FLR*** (*function level reset*) mogu se virtualizirati na ovaj način, jer je to potrebno za ponovno dodjeljivanje različitih funkcija uređaja između virtualnih računala.
Nadalje, ako uređaj koji se koristi ne podržava **MSI** signalizaciju (*Message Signaled Interrupts*) (***MSI***), on ne smije dijeliti signale prekida (***IRQ***) s drugim uređajima kako bi dodjela uopće bila moguća. Nadalje, svi konvencionalni PCI uređaji spojeni na pojedini PCI/PCI-X-to-PCI Express most (engl. *bridge*) mogu se dodijeliti virtualnom računalu samo i isključivo svi odjednom. S druge strane ***PCI Express*** uređaji nemaju takvo ograničenje pa se mogu dodavati jedan po jedan. I na kraju i kernel Linuxa mora imati ovu podršku, kao i *hipervizor* (KVM/QEMU), ali i virtualno računalo.

## Upotreba u Linuxu
Mi ćemo se u primjerima koji slijede fokusirati na Intel ***VT-d*** tehnologiju.
Prvi korak je uključivanje *VT-d* tehnologije u *BIOSu* ili *UEFIju*. Tek nakon restarta računala, ako je VT-d omogućen, Linux će konfigurirati DMA translacije tijekom pokretanja sustava. Međutim to ovisi o postavkama kernela (i uključenim opcijama).
Najlakši način da to provjerite je potražiti takozvane ***DMAR*** unose u `dmesg` porukama.
Ako ne vidite pogreške, *VT-d* je omogućen. Pogledajmo kako to provjeriti:

```
dmesg | grep -e DMAR -e IOMMU
ACPI: DMAR 0000000079a20300 000C4 (v01 SUPERM SMCI--MB 00000001 INTL 20091013)
DMAR: dmar0: reg_base_addr fbffc000 ver 1:0 cap 8d2078c106f0466 ecap f020de
DMAR-IR: IOAPIC id 8 under DRHD base  0xfbffc000 IOMMU 0
DMAR-IR: HPET id 0 under DRHD base 0xfbffc000
DMAR-IR: x2apic is disabled because BIOS sets x2apic opt out bit.
DMAR-IR: Use 'intremap=no_x2apic_optout' to override the BIOS setting.
DMAR-IR: Enabled IRQ remapping in xapic mode
```

Na kraju ispisa vidimo važnu poruku „`Enabled IRQ remapping in` **`XY`** `mode`“ koja govori da se radi i translacija signala prekida (***IRQ***). U slučaju kada ne vidimo ***DMAR*** (*IOMMU*), očito je da imamo kernel koji ga standardno ne aktivira, pa je u *GRUB2* potrebno ručno dodati opcije, koje ćemo potom razjasniti.
*GRUB2* nećemo ručno mijenjati već ćemo koristiti program `grubby`, jer nam je to jednostavnije.

```
grubby --update-kernel=ALL --args=intel_iommu=on
```

Opcija „**`intel_iommu=on`**“ naznačuje kernelu da se prilikom inicijalizacije koristi *IOMMU* (*Intel VT-d*).
Druga opcija koju ćemo spomenuti (**`iommu=pt`**) je također korisna, ako koristimo ***SR-IOV*** (za posebne mrežne kartice koje to podržavaju), jer omogućuje upotrebu u tzv. propuštajućem načinu rada u kojem mrežna kartica ne treba koristiti ***DMA***, što u ovakvoj primjeni poboljšava performanse u virtualizaciji.

```
grubby --update-kernel=ALL --args=iommu=pt
```

Nakon što smo dodali ove opcije u kernel, potreban je restart računala.
Nakon restarta, provjerite je li sve u redu (s novim opcijama kernela), pokretanjem sljedeće dvije naredbe:

```
cat /proc/cmdline | grep -i iommu
dmesg | grep -i iommu
```

Važno je da u ispisu nemamo nikakve poruke o greškama (pr. „*Unknown kernel command line parameters*“ ili slično).

## Hipervizor i virtualno računalo
Kao što smo već naveli i *hipervizor* mora imati podršku za IOMMU, što je u slučaju QEMU/KVM podržano već neko vrijeme, a naziva se ***guest vIOMMU***. Važno je razumjeti da QEMU mora virtualnom računalu proslijediti podršku za IOMMU, što se odrađuje aktivacijom posebnih QEMU uređaja, samo ako se koristi Q35 platforma (`-machine q35`). Zatim je potrebna definicija tzv. `kernel-irqchip=split` opcije za emuliranje signala prekida, dostupne od kernela 4.4+.
Ostatak konfiguracije je: `-device intel-iommu,intremap=on`, s time da je prva opcija koja dodaje poseban *vIOMMU* uređaj, a druga uključuje preslikavanje hardverskih signala prekida (*IRQ*) [*interrupt remapping*]. Ako se koriste *VirtIO paravirtualizirani* uređaji, dodaje se i opcija: `device-iotlb=on`. Dok je unutar virtualnog računala, potrebna podrška odnosno učitan kernel modul imena: `virtio-iommu`, koji je podržan od *VirtIO* standarda v.1.1. Sve točne opcije i parametri uvijek ovise o inačicama komponenti sustava.

***Virtio-iommu*** uređaj upravlja izravnim pristupom memoriji (***DMA*** pristup) s jedne ili više krajnjih točaka. Može djelovati i kao posrednik (proxy) za fizički IOMMU koji upravljaju uređajima dodijeljenim gostu (virtualnom računalu) i kao virtualni IOMMU koji upravlja emuliranim i *paravirtualiziranim* uređajima (*VirtIO* uređajima: pr. *VirtIO* mrežna kartica i slično).



Za više detalja o programu **grubby** i dodatnim postavkama za virtualizaciju, pogledajte poglavlja:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**
**26.8.5. Open vSwitch servis.**
**27.1.2. Rad s virtualizacijom (KVM+QEMU).**

Kako smo već naučili mnogi sustavi omogućuju **DMA** pristup te mogućnost ponovnog mapiranja signala prekida kako bi se osiguralo da se I/O uređaji ponašaju unutar granica koje su im dodijeljene. To između ostalih uključuje navedeni ***x86*** hardver s podrškom za ***AMD-Vi*** i Intel ***VT-d***. Sada dolazimo i do sljedeće važne komponente, a to je **VFIO** upravljački program. On je *IOMMU*/uređaj koji sustavu daje agnostički okvir za izlaganje izravnog pristupa uređaju unutar korisničkog prostora sustava, u sigurnom, *IOMMU* zaštićenom okruženju. Neke aplikacije, posebno u području računalstva visokih performansi, također imaju koristi od niskog opterećenja sustava, te izravnog pristupa uređajima iz korisničkog prostora (*user-space*). Primjeri uključuju posebne mrežne kartice i računalne akceleratore.
Prije VFIO-a, ovi upravljački programi morali su ili proći kroz puni razvojni ciklus da bi postali ispravan upravljački program, ote se održavati izvan stabla kernela ili koristiti ***UIO*** programski okvir koji nažalost nije svjestan *IOMMU* mehanizama te ima ograničenu podršku za signale prekida (IRQ), a dodatno zahtijeva administratorske (root) privilegije za pristup stvarima kao što je konfiguracijski prostor PCI uređaja.
Ideja razvoja ***VFIO*** upravljačkog programa je objediniti sve navedeno, zamjenjujući KVM PCI specifičan programski kôd za dodjelu uređaja kao i pružanje sigurnijeg i razvojno bržeg okruženja upravljačkog programa korisničkog prostora od ***UIO***.

## Zbog čega je onda uopće razvijen UIO?

Za mnoge vrste uređaja izrada upravljačkog programa za Linux kernel je prekomplicirana. Sve što je stvarno potrebno je neki način da se riješi baratanje sa signalima prekida (IRQ) te omogući pristup memorijskom prostoru uređaja. Logika upravljanja uređajem ne mora nužno biti unutar kernela, budući da uređaj ne mora iskoristiti niti jedan drugi resurs koji pruža kernel.
Kako bi se riješila ova situacija, dizajniran je I/O sustav korisničkog prostora, koji se naziva ***UIO***. Za tipičnu (jednostavniju) periferiju odnosno *kartice*, potreban je samo vrlo mali kernel modul. Glavni dio upravljačkog programa izvodit će se u korisničkom prostoru. To pojednostavljuje razvoj i smanjuje rizik od ozbiljnih grešaka unutar kernel modula. Osim toga za bilo koju promjenu nije potrebno ponovno kompilirati cijeli kernel. Mane **UIO** smo naveli gore, dakle svode se uglavnom na performanse i brzinu razvoja (programiranja) kernel modula (upravljačkog programa).

***Vratimo se na VFIO***
***VFIO*** možemo privremeno (do restarta računala) učitati ovako:

```
modprobe vfio-pci
```

Ipak, ponekad (za određene komponente sustava) potreban je i **UIO**, koji možemo učitati ovako:

```
modprobe uio_pci_generic
```

Potom je potrebno promijeniti ovlasti:

```
chmod a+x /dev/vfio
chmod 0666 /dev/vfio/*
```

I nakon toga slijedi dio konfiguracije u kojem je potrebno povezati mrežnu karticu (NIC) s *VFIO*. Ovaj korak ovisi o mrežnoj kartici, pa ga nećemo opisivati. Ova procedura povezivanja mrežne kartice i VFIO zove se „*binding NIC to VFIO*“ **(1325)**.



Za trajno učitavanje kernel modula, pogledajte poglavlje:
**11.1.1.3. Parametri koje možemo poslati kernel modulima te automatsko učitavanje modula.**



Za više detalja i dodatnim postavkama za virtualizaciju, pogledajte poglavlja:
**26.8.5. Open vSwitch servis.**
**27.1.2. Rad s virtualizacijom (KVM+QEMU).**

**Izvori informacija: (1319), (1320),(1321),(1322),(1323),(1324),(1325), (1335),(1336),(1337),(1338),(1339),**
`man grubby`,`man dmesg`,`man modprobe`,`man chmod`,`man procfs`.

## 10.7.2. SMP i NUMA

### 10.7.2.1. SMP

***Simultaneous multithreading*** (**SMT**) je tehnika za poboljšanje ukupne učinkovitosti *superskalarnih* procesora s hardverskom implementacijom višenitnog rada. To znači da *SMT* dopušta izvršavanje više nezavisnih nîti programa kako bi se bolje iskoristili resursi koje nam pružaju moderne arhitekture procesora i u konačnici cijelog računala. Pojam ***multithreading*** zmači da se više programskih nîti može izvršavati istovremeno na jednoj jezgri procesora, ali i da se više programa koji koriste različite segmente RAM memorije može izvršiti (pokrenuti) čak i u različitim zaštitnim prstenovima izvršavanja, s različitim ulazno/izlaznim dozvolama i tako dalje. Ovi programi iako se izvode (rade) na istoj jezgri procesora, potpuno su odvojeni jedan od drugog. *Multithreading* je po konceptu sličan *preventivnom multitaskingu* (engl. *preemptive multitasking*), ali je implementiran na razini nîti izvršavanja u modernim ***superskalarnim*** procesorima odnosno unutar takozvanih logičkih jezgri procesora.

***Superskalarni*** procesor je procesor koji implementira oblik paralelizma koji se naziva paralelizam na razini instrukcija unutar procesora. Za razliku od *skalarnog* procesora, koji može izvršiti najviše jednu instrukciju po ciklusu takta, *superskalarni* procesor može izvršiti više od jedne instrukcije tijekom ciklusa takta simultanim slanjem više instrukcija različitim izvršnim jedinicama na procesoru (tzv. *processing units*: **PU**). Dakle pojam *superskalarni* podrazumijeva mogućnost izvršavanja više od jedne procesorske instrukcije po taktu procesora. Različiti proizvođači drugačije nazivaju *SMT* tehnologiju, pa ju tvrtka *Intel* naziva *Hyper-threading*, dok ju tvrtka *AMD* naziva jednostavno *SMT*.

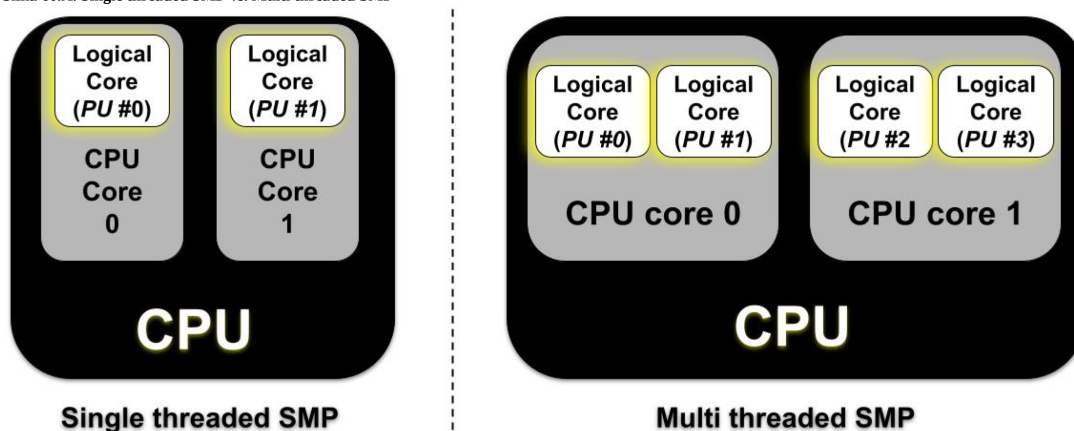
**Instrukcijski cjevovod (*instruction pipeline*)**
Upotreba instrukcijskog cjevovoda je tehnika za implementaciju paralelizma procesorskih instrukcija unutar jednog procesora. Ova tehnika pokušava zadržati svaki dio procesora zauzet procesorskim instrukcijama dijeljenjem dolaznih instrukcije u nîz odnosno u cjevovod iz kojeg ih potom mogu izvršiti različite procesorske jedinice (***PU***) odnosno jezgre procesora paralelno.
U arhitekturama procesora koje koriste cjevovode, instrukcije teku kroz središnju procesorsku jedinicu (CPU) u fazama.
U pravilu, moguće je imati jednu fazu za svaki korak tzv. *von Neumannovog* ciklusa: dohvaćanje instrukcija, dohvaćanje operanda, izvršavanje instrukcije, ispis rezultata. U ovakvoj arhitekturi procesora obično postoje "*registri cjevovoda*" nakon svake od ovih faza. Oni pohranjuju informacije iz instrukcija i izračuna (rezultata) tako da elementi sljedeće faze mogu učiniti sljedeći korak. Takav način rada omogućuje procesoru da izvršava instrukcije u svakom radnom taktu.

To se radi bez velikih promjena u osnovnoj arhitekturi procesora: glavni potrebni dodaci su mogućnost dohvaćanja naredbi (procesorskih instrukcija) iz više nîti u ciklusu obrade, za što su potrebni veći registri za čuvanje podataka iz više nîti.
Broj istodobnih nîti ovisi o dizajnu procesora. Dvije istodobne nîti po fizičkoj jezgri procesora su uobičajene, međutim neki procesori podržavaju čak do osam istodobnih (paralelnih) niti [tzv. *logičkih jezgri*] po svakoj fizičkoj procesorskoj jezgri.

Međutim važno je razumjeti da je *simultaneous multithreading* (**SMT**) jedna je od dvije glavne implementacije *multithreadinga*, dok se drugi oblik odnosno implementacija naziva temporalni ili vremenski multithreading, također poznat kao super-threading. U vremenskom *multithreadingu*, samo jedna nît programa može se izvršiti u bilo kojoj fazi cjevovoda procesora odjednom. Dok se u ***SMT*** implementaciji, instrukcije iz više nîti mogu izvršavati u bilo kojoj fazi cjevovoda odjednom.

Na slici vidimo usporedbu ***single threaded SMP*** procesora u odnosu na ***multi threaded SMP*** procesor, a logičke jezgre koje smo označili kao **PU** odnosno tzv. *processing Unit* možemo promatrati kao komponente procesora zadužene za procesiranje (obradu) instrukcija. Prema logici rada one predstavljaju logičke jezgre procesora, kako je vidljivo na slici. **60.A.**

**Slika 60.A. Single threaded SMP vs. Multi threaded SMP**




Dok *superskalarni* procesor tipično koristi instrukcijske cjevovode, *superskalarno* i cjevovodno izvođenje smatraju se različitim tehnikama poboljšanja performansi. Prvi izvršava više instrukcija paralelno koristeći više izvršnih jedinica (***PU***), dok drugi izvršava više instrukcija u istoj izvršnoj jedinici paralelno dijeleći izvršnu jedinicu u različitim fazama rada.
*Superskalarna* tehnologija se tradicionalno povezuje s nekoliko identifikacijskih karakteristika (unutar danog procesora):

- Instrukcije se dohvaćaju iz sekvencijalnog tôka instrukcija.
- Procesor dinamički provjerava ovisnosti podataka između instrukcija tijekom samog izvođenja, za razliku od provjere u softveru tijekom kompilacije programa.
- Procesor može izvršiti više instrukcija po taktu.

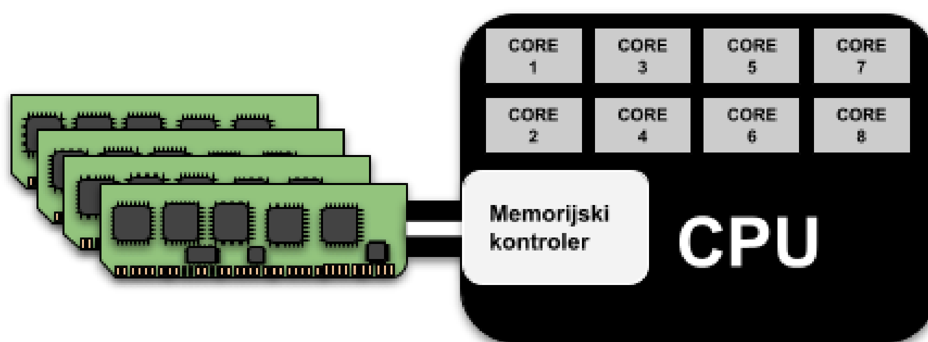
U konačnici, *simultaneous multithreading* (***SMT***) je tehnika za poboljšanje ukupne učinkovitosti superskalarnih procesora.



Za više detalja, pogledajte i poglavlje:
**10.7.1.1. Malo detaljnije o registrima, instrukcijama i arhitekturi procesora.**

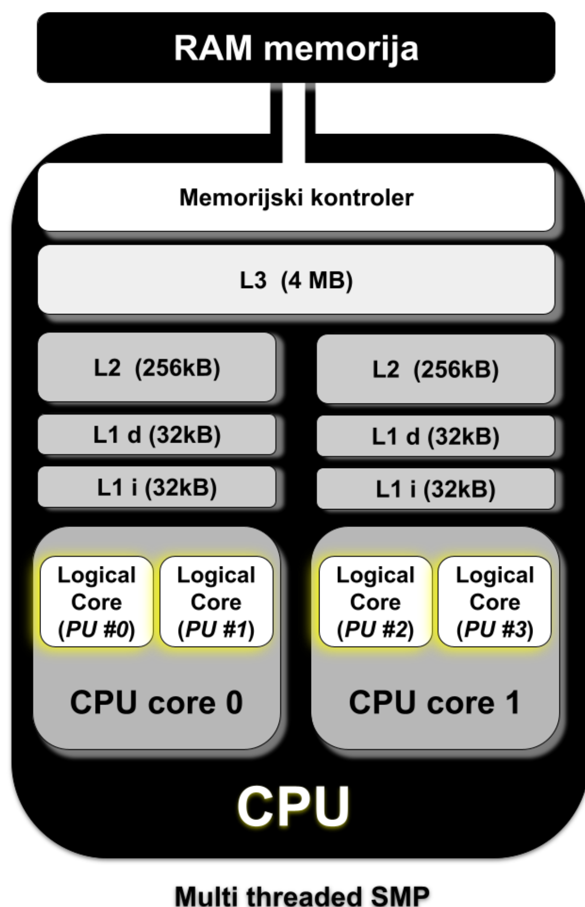
Sumirajmo što je **SMP (engl. Symmetric Multi-Processor)** odnosno simetrični višeprocesorski sustav.
Na današnjim stolnim i prijenosnim računalima, može se primijetiti kako njihov procesor (CPU) ima nekoliko procesorskih jezgri. Dakle jedan fizički procesor sadržava više takozvanih *logičkih* jezgri. Tako danas u standardnim inačicama modernih procesora imamo osam ili više jezgri unutar jednog fizičkog procesora. Ovakva arhitektura fizičkog procesora koji sadržava više procesorskih jezgri se naziva **SMP** (engl. *Symmetric Multi-Processor*) arhitektura.
Sve jezgre unutar jednog fizičkog procesora u ovakvim *SMP* sustavima preko svog memorijskog kontrolera pristupaju svôj raspoloživoj RAM memoriji, korištenjem memorijske sabirnice, bez nekih dodatnih ograničenja. Ovakav dizajn jedino ima svoja ograničenja po pitanju proširivosti odnosno skaliranja. Problem je u tome što je jedino moguće proširenje po pitanju procesorskih jezgri, ugradnja novog procesora (CPU) s više jezgri i to sâmo, ako to podržava matična ploča računala. Međutim ovo najčešće znači kupovinu i nove matične ploče i novog procesora. Sve ovo nije problem ukoliko vaše potrebe ne prelaze ograničenja jednog fizičkog procesora i svih njegovih (logičkih) procesorskih jezgri, kao i pripadajuće RAM memorije. Pozicija fizičkog procesora i njegovih jezgri (na slici **60**. označenih kao: ***CORE 1***, ***CORE 2***, …) te memorijskog kontrolera, kao i pripadajuće RAM memorije za *SMP* sustave, vidljiva je na slici **60.B.**

**Slika 60.B. Pogled na SMP mikroprocesor (CPU) i njen memorijski kontroler.**




Sada detaljnije promotrimo arhitekturu jednog višenitnog (*multi threaded*) **SMP** procesora, kao na slici **60.C**.

**Slika 60.C. Pogled na multihreaded SMP mikroprocesor (CPU) i njegove priručne memorije (L1, L2 i L3)**




Kako je vidljivo na slici, da bi procesor ili njegove logičke jezgre ([*PU#*]) dohvatile ili zapisale podatke u *RAM* memoriju ili dohvatile instrukcije za izvršavanje, koriste se posebne predmemorije koje su hardverski implementirane unutar samog procesora. Dakle hardverske predmemorije koriste procesorske jedinice odnosno logičke ili fizičke jezgre računala za smanjenje potrošnje vremena za pristup podacima ili instrukcijama iz glavne (RAM) memorije. To je tako jer su predmemorije manje i drastično brže vrste memorija u odnosu na RAM memoriju te su smještene bliže jezgrama procesora. One pohranjuju kopije podataka ili instrukcija s često korištenih lokacija RAM memorije.
Većina današnjih procesora ima hijerarhiju višestrukih razina predmemorija (L1, L2, često i L3, a ponekad čak i L4), s različitim predmemorijama specifičnim za instrukcije i podatke na razini 1 i zajedničkoj većoj memoriji na razinama 2 i 3.

Drugi problem je temeljni kompromis između latencije (kašnjenja) predmemorije i stope pogodaka odnosno dohvaćanja podataka iz nje. Naime, veće predmemorije imaju bolje stope pogodaka, ali veću latenciju. Kako bi riješili ovaj kompromis, mnoge arhitekture procesora koriste više razina predmemorija, s malim brzim predmemorijama (**L1**) koje mogu koristiti veće, sporije predmemorije (pr. L2 i L3). Upotreba predmemorije s više razina općenito radi tako da se prvo provjerava najbrža predmemorija razine 1 (**L1**); ako je podatak (ili instrukcija) pronađena (pogodak [*cache hit*]), procesor ih dohvaća velikom brzinom. Ako ta manja predmemorija nema traženi podatak ili instrukciju, provjerava se sljedeća najbrža predmemorija, razine 2 (**L2**), i tako dalje, prije končanog pristupanja vanjskoj najsporijoj (RAM) memoriji.

Kod *multi threaded* **SMP** procesora, dijeljenje **L1** predmemorije (često i L2) između fizičkih jezgri je nepoželjno jer bi rezultirajuće povećanje latencije uzrokovalo znatno sporiji rad od jednojezgrenog procesora.
Međutim, dijeljena predmemorija najviše razine (obično **L3**), posljednja koja se koristi prije pristupanja RAM memoriji, poželjna je iz nekoliko razloga: dopuštanje bilo kojoj pojedinoj jezgri da koristi cijelu predmemoriju, smanjenje redundantnosti podataka omogućavanjem različitih procesa ili nîti za dijeljenje predmemoriranih podataka i smanjenje složenosti korištenih protokola koherentnosti predmemorije.

Na primjeru sa slike, četvero jezgreni procesor s tri razine međumemorije uključuje **L1** predmemoriju (zasebno za podatke i zasebno za instrukcije) za svaki par logičkih jezgri unutar fizičke jezgre, kao i jednu međumemoriju **L2** za svaki par jezgri te jednu **L3** predmemoriju koju dijele sve fizičke i logičke jezgre unutar procesora. Pri tome se zajednička predmemorija najviše razine (**L3**), koja se dohvaća prije pristupanja RAM memoriji, obično se naziva predmemorija posljednje razine „*Last level cache*" (**LLC**). U njenom radu se koriste dodatne tehnike za povećanje razine paralelizma kada se LLC dijeli između više jezgri, uključujući njeno dijeljenje na više dijelova koji se odnose na određene raspone memorijskih adresa, kojima se može pristupiti neovisno.



Za više detalja o specifičnostima u radu s **L1**, **L2** i **L3** memorijama, pogledajte poglavlja:
**10.7.1.3. CPU registri još detaljnije.**
**12.3.2.3. Proces translacije adresa.**

Međutim da bi *SMP* arhitektura uopće radila, osim podrške u hardveru i BIOS-u računala, potrebna je i podrška u kernelu Linuxa, odnosno kernel mora bit kompiliran s podrškom za nju. Podršku u kernelu za SMP možemo provjeriti sa:

```
grep CONFIG_SMP= /boot/config-`uname -r`
CONFIG_SMP=y
```

← pošto ovdje imamo =y to znači da ju naš kernel podržava (što je zadana postavka).

Kada je SMT odnosno SMP operativan, operativni sustav vidi da procesor ima dvostruke jezgre. Operativni sustavi ovu akumulaciju jezgri i nîti vide kao logičke procesorske jezgre, koje su potom dostupne svim sistemski servisima i svim aplikacijama.

`/proc/schedstat` datoteka sadrži statistike vezane za raspodjelu procesorskog vremena i procesa na različite procesorske jezgre. Ove statistike omogućuju praćenje performansi raspodjele procesa u višejezgrenim sustavima. U toj statistici svaki redak odnosi se na pojedinu procesorsku jezgru. U konačnici to nam omogućuje praćenje koliko se često procesi (programi) prebacuju (tzv. *context switch*) između jezgri, a može biti korisno i za dijagnostiku i analizu performansi u višejezgrenim okruženjima.
Provjeru je li SMT uključen možemo napraviti ovako:

```
cat /sys/devices/system/cpu/smt/active
1
```

← ako imamo vrijednost 1, to znači da je *SMT* aktivan.

**SMT** je moguće isključiti na tri načina:

- U ***BIOSu*** računala.
- U radu računala: ova metoda može zahtijevati noviji kernel, ali ne zahtijeva ponovno pokretanje sustava te nije postojano (na restart računala). Noviji kerneli uključuju sljedeće sučelje za kontrolu SMT-a tijekom izvođenja, pri čemu je na sustavu dostupna posebna datoteka: `/sys/devices/system/cpu/smt/control`. Tada imamo nekoliko stanja:
    - `on` – ovo je zadano stanje koja naznačava da je SMT uključen.
    - `off` – ovo je stanje s kojim se SMT isključuje.
    - `forceoff` – ovo je stanje u kojem je SMT (već) isključen, a na ovo stanje ne možemo utjecati.
    - `notsupported` – ovo je stanje koje naznačava da naš procesor ne podržava SMT.

Privremeno (do restarta računala) isključivanje SMT-a u radu moguće je pokretanjem sljedeće naredbe:

```
echo off > /sys/devices/system/cpu/smt/control
```

- Dodavanjem posebne kernel opcije odnosno tzv. ***GRUB boot*** parametra `nosmt`. Nakon čega slijedi restart računala. Ako je opcija "`nosmp`" aktivirana u *Grub* konfiguraciji, ona će onemogućiti podršku za *Simultaneous MultiProcessing* (**SMP**) i koristit će se samo fizičke jezgre vašeg procesora, bez aktivnih *Hyper-Threading* nîti odnosno dodatnih jezgri koje podržava tehnologija poput *Hyper-Threadinga* kod *Intel* procesora. Ovo je korisno samo u nekim specifičnim situacijama, ali ograničava iskorištavanje svih raspoloživih resursa. Ako želite iskoristiti sve jezgre i nîti (logičke jezgre) na vašem CPU-u, trebali biste onemogućiti "`nosmp`" opciju. To možete učiniti tako da uklonite tu opciju iz konfiguracije ***Grub***-a i ponovno pokrenete sustav. Tada će se aktivirati sve fizičke jezgre i pripadajuće *Hyper-Threading* nîti, što će omogućiti bolje iskorištavanje raspoloživih resursa i poboljšati performanse

To je za aktivni kernel moguće napraviti ovako (**nemojte to raditi, osim ako stvarno imate dokazani razlog za to**):

```
grubby --args=nosmt --update-kernel=DEFAULT
```

**Izvor informacija: (141),(978),(1537),(1538),(1539),(1540),(1541),(1542),(1543),(K-9).**

#### 10.7.2.1.1. CPU afinitet (CPU Affinity)

U normalnom radu sâm se operativni sustav brine o tome na kojoj jezgri procesora će se izvršavati koji program odnosno proces. Pojam CPU afiniteta odnosno ***CPU affinity*** je vezan uz *SMP* sustave kada izričito želimo narediti operativnom sustavu da se određeni (željeni) program (proces) izvršava samo i isključivo na određenoj jezgri (ili jezgrama) procesora.
To se odrađuje tako da linux ***task scheduler*** odnosno ***proces scheduler*** veže označeni proces na procesorske jezgre koje smo mu zadali. Naime linux *task/proces scheduler* inače svaki proces dodjeljuje određenoj jezgri procesora prema svom internom algoritmu i prioritetima, što u posebnim slučajevima, poglavito kada se radi o zahtjevnim aplikacijama, nije najbolje rješenje.
Kao što smo rekli, CPU afinitet je povezan s ***task scheduler****om* i njegovim mehanizmima, koji se zapravo brinu o:

- Pokretanju procesa na određenoj jezgri procesora i potencijalnom prebacivanju procesa na druge jezgre procesora.
- Prioritetima obrade podataka (engl. *Task scheduler priority & policy*) na razini ispod svakog procesa odnosno aplikacije, prema CPUu. Dakle ovaj zadatak obavlja kernel.



Za više detalja o Linux *task/proces scheduleru* pogledajte poglavlje:
**9.3 Task/process scheduler.**
Pogledajte i kako na razini cijelog sustava izolirati određene jezgre CPUa:
**10.7.2.4. Izolacija jezgri procesora.**

**Za potrebe postavljanja CPU *afiniteta* koristit ćemo naredbu** `taskset`
Ova naredba odnosno program se može koristiti:

- Za postavljanje "***CPU afiniteta***" već pokrenutog procesa.
- Za pokretanje novog programa (procesa) s definiranim CPU afinitetom.
- Za provjeru CPU afiniteta već pokrenutih procesa (programa).

Naredba `taskset` je potpuno djelotvorna na *SMP* sustavima. Na *NUMA* sustavima s ovom naredbom ne možemo biti sigurni čiju RAM memoriju (lokalnog CPU-a ili udaljenog CPU-a) će proces koristiti. Za *NUMA* arhitekturu pogledajte sljedeće poglavlje: **NUMA**. Za CPU afinitet za *NUMA* arhitekturu se koristi naredba `numactl`. CPU afinitet je na niskoj razini sustava prezentiran takozvanom *bit maskom*. Najniži bit predstavlja prvi logički CPU (jezgru), a najviši bit predstavlja zadnji logički CPU (jezgru). Osim navedene bit maske moguće je postavljati CPU afinitet s brojčanim oznakama CPU jezgri što ćemo i mi koristiti.

**Pogledajmo primjere**
**1.** Provjerimo CPU afinitet pokrenutog procesa s naredbom `taskset`. Radi se o procesu čiji ***PID*** (*Process ID*) smo saznali s naredbom `pidof`. Pogledajte primjer iz poglavlja ***Task scheduler*** (poglavlje: **9.3.**). Dakle saznali smo kako se radi o ***PID*** broju: `410561`. Upotrebom naredbe `taskset` koristiti ćemo njene sljedeće prekidače:

- `-p` označava da dajemo ***PID*** broj od pokrenutog procesa.
- `-c` označava da želimo numeričku listu CPU jezgri (umjesto bit maske koja nam je malo teže razumljiva).

```
taskset -p -c 410561

pid 410561's current affinity list: 0-7
```

Kao rezultat smo dobili odgovor da naš proces ***PID*** `410561` (zapravo se radilo o *SSH* servisu): ima CPU afinitet: 0-7, što znači kako smije koristiti CPU jezgre: 0,1,2,3,4,5,6 i 7. U našem slučaju se radilo o procesoru s osam (8) jezgri.
Dakle naš proces (program) ih može koristiti sve, odnosno može koristiti bilo koju od procesorskih jezgri.

**2.** Postavimo CPU afinitet pokrenutom procesu (programu), tako da se smije pokretati samo na CPU jezgri broj jedan (**1**):

```
taskset -p -c 1 410561

pid 410561's current affinity list: 0-7
pid 410561's new affinity list: 1
```

Rezultat ispisa je potvrda da je sada CPU afinitet promijenjen i to samo na CPU jezgru broj jedan (**1**).

**2.1.** Moguće je postaviti CPU afinitet za nîz CPU jezgri, a ne samo za jednu.
Postavimo ipak za naš proces ***PID*** broja `410561` CPU afinitet na CPU jezgre **0** i **1**, na sljedeći način:

```
taskset -p -c 0,1 410561

pid 410561's current affinity list: 1
pid 410561's new affinity list: 0,1
```

**3.** Postavimo CPU afinitet za proces koji tek pokrećemo (zove se `/bin/moja-skripta-ili-program`) na CPU jezgre **2** i **3**. To ćemo napraviti ovako:

```
taskset -c 2,3 /bin/moja-skripta-ili-program
```

**4.** Možemo kombinirati i CPU afinitet i prioritet *process/task schedulera* (engl. *Priority*) i algoritma (engl. *Scheduler policy*) za naš program iz primjera prije:

```
taskset -c 2,3 chrt -f 78 /bin/moja-skripta-ili-program
```

Što smo ovdje napravili?

Postavili smo CPU afinitet samo na jezgre `2` i `3` dakle samo na njima će se naš program moći izvršavati.
Nadalje, postavili smo parametre rada proces/task schedulera za naš program koji pokrećemo, na:

- `-f` – odabrali smo "*SCHED_FIFO*" algoritam.
- `78` – odabrali smo prioritet unutar gore navedenog algoritma, na: 78 (za taj algoritam: najniži prioritet=**1**, najviši=**99**).

**Izvor informacija: (K-4), `man taskset`.**

### 10.7.2.1.2. Process/Thread Affinity i CPU Affinity

Objasnili smo što se događa s pojedinim procesima (programima), ali što je s programskim nîtima (engl. *Thread*) unutar svakog pojedinog procesa odnosno programa (aplikacije). Naime u razvoju programa bilo bi dobro paziti da je program pisan tako da je moguće izvršavati više logičkih cjelina u paraleli odnosno razvijati aplikacije koje mogu koristiti više programskih nîti istovremeno. Sve imalo naprednije aplikacije, a pogotovo one za poslužiteljske namjene, su upravo tako i pisane odnosno razvijane. Ovdje zapravo dolazimo do malih komplikacija. L*inux task/proces scheduler,* što sâm, a što s našim podešavanjem CPU afiniteta može vrlo dobro rasporediti pojedine procese odnosno programe/aplikacije na pojedine CPU jezgre.
Međutim ostaje pitanje kako će rasporediti programske nîti unutar svakog od tih procesa. Sada zapravo ponovno dolazimo do područja koje više pripada *linux task/proces scheduleru*, ali ćemo to ipak objasniti ovdje. *Linux task/proces scheduler* ne pravi razliku radi li se o procesu ili njegovoj programskoj nîti, te primjenjuje iste mehanizme za njih. Dakle ako je pokrenutom procesu dozvoljeno da koristi sve dostupne CPU jezgre sustava, on će ih i koristiti tako što će i programske nîti unutar tog procesa koristiti sve dostupne jezgre procesora. Dakle optimizacijom svakog pojedinog procesa/programa/aplikacije, prema *linux task/proces scheduleru* radimo optimizacije za sve njegove programske nîti. Tako primjerice, ako imamo pokrenut program koji može koristiti četiri jezgre procesora, a pisan je tako da ima samo dvije programske nîti, te nîti će se zapravo pokretati na sâmo dvije jezgre procesora.

**Kako provjeriti koje programske nîti našeg programa se pokreću na kojoj jezgri procesora?**
Prvo moramo naći ***PID*** našeg pokrenutog procesa/programa. Recimo da smo pronašli da je njegov ***PID*** broj **4071**.
U našem primjeru radi se o programu imena `kvm` koji pokreće jedno virtualno računalo na našem poslužitelju, ali se može raditi o bilo kojem programu za koji znamo da koristi više nîti u radu. Provjerimo koji ***PID*** broj ima program imena `kvm`:

```
pidof kvm

4071
```

Za ovu potrebu, koristit ćemo naredbu `ps` s dodatnim parametrima, s kojima ćemo moći vidjeti sve što nas zanima o našem pokrenutom procesu ***PID*** broja **4071**. Stoga pokrenimo naredbu `ps` na sljedeći način:

```
ps -mo pid,tid,fname,user,psr -p 4071

  PID    TID COMMAND  USER      PSR
 4071      - kvm      root        -
    -   4071 -        root        5
    -   4096 -        root        2
    -   4097 -        root        3
```

Vidimo kako naš proces (***PID*** `4071`) ima više programskih nîti; pogledajte `Thread ID` (`TID`): `4071`,`4096` i `4097` koje su trenutno aktivne (možda ih ima i više, ali trenutno nisu aktivne) te kako se neke od njih pokreću na CPU jezgri `5`, a neke na CPU jezgri `3`. Svako pokretanje gore navedene naredbe očitava trenutno stanje i to u djeliću sekunde kada smo pokrenuli naredbu.
Pogledajmo opis prekidača naredbe `ps` koje smo koristili:

- `-mo` - ispiši programske nîti (*Thread*) poslije procesa (`m`), a nakon toga slijede dodatne opcije (`o`) opisane dolje:
  - `pid` - ispiši ***PID*** (*Process* ID) trenutnog (našeg) procesa.
  - `tid` - ispiši ***TID*** (*Thread* ID) (ID od svake nîti) trenutnog (našeg) procesa.
  - `fname` - ispiši ime pokrenutog procesa/programa (vidljivo u stupcu ispod "`COMMAND`").
  - `user` - ispiši ime korisnika koji je pokrenuo ovaj (naš) proces/program.
  - `psr` - ispiši procesorsku (CPU) jezgru na kojoj se trenutno izvršava programska nît (ili program).

Još više korisnih detalja, možemo dobiti sa sljedećim opcijama:

- `pcpu` – ispiši i zauzeće svake pojedine CPU jezgre
- `stat` – ispiši trenutno stanje procesa, a koje može biti:
  - `D` – *Uninterruptible sleep* - obično čeka na ulazno izlazne operacije (*I/O).*
  - `R` – *Running* - normalno se izvodi.
  - `S` – *Interruptible sleep* - čeka na neki događaj da se završi.
  - `T` – Zaustavljen je.
  - `W` – *Paging* - više se ne koristi.
  - `X` – *Dead – mrtav/zaustavljen.*
  - `Z` – *Zombie – zombi (proces koji je u neodređenom stanju/nepravilno je zaustavljen).*
- `%mem` – ispiši ukupno zauzeće RAM memorije.
- `rss` – ispiši zauzeće rezidentne memorije (*Resident Set Size*) što znači koliko je (RAM) memorije alocirao proces što uključuje i zauzeće memorije od biblioteka koje proces koristi te *stack* i *heap* memoriju. Ne uključuje dio memorije koja je *Swap*.
- `vsz` – ovo je ukupno zauzeće sustava virtualne memorije koje proces može koristiti, koje može uključiti sve od `rss` djela te dodatno i ***swap*** dio. Ovo ne znači da se ovoliko memorije i koristi, već je eventualno sâmo pripremljena.

Sve opcije koje smo naveli, zajedno bi izgledale ovako:
```
ps -mo pid,tid,fname,user,psr,pcpu,stat,%mem,vsz,rss -p 4071
```
Nakon što smo izvršili gore navedenu naredbu, dobit ćemo sljedeći ispis (ispis smo malo skratili):
```
PID    TID COMMAND     USER    PSR %CPU STAT %MEM     VSZ    RSS
4071        - kvm       root      -  130    -     49.2 9569896 8061944
    - 4071  -           root      0  0.2    Sl       -       -      -
    - 4096  -           root      6  0.0    Sl       -       -      -
    - 4098  -           root      2 14.0    Rl       -       -      -
    - 4100  -           root      4 50.5    Rl       -       -      -
    - 4101  -           root      5 10.5    Rl       -       -      -
    - 4102  -           root      1  9.9    Rl       -       -      -
```
Ako budemo pokretali gore navedenu naredbu više puta, vidjet ćemo kako se programske nîti raspoređuju po CPU jezgrama unutar granica koje su dozvoljene za trenutni proces.

Međutim, upotreba **psr** ne radi na svim inačicama programa ps i na svim sustavima, stoga možemo koristiti i prekidač **LP**:
```
ps -LP -p 4071

 PID    LWP     PSR TTY     TIME       CMD
 4071     4071    4   ?      00:01:21   kvm
 4071     4096    3   ?      00:00:00   call_rcu
 4071     4097   19   ?      00:00:01   kvm
 4071     4097   10   ?      00:00:19   CPU 0/KVM
 4071     4098   29   ?      00:00:21   CPU 1/KVM
 4071     4099    8   ?      00:00:10   CPU 2/KVM
 4071     4100    9   ?      00:00:11   CPU 3/KVM
```
← Dakle i ovdje imamo stupac PSR u kojem vidimo koja procesorska jezgra trenutno obrađuje koji proces (program).

Provjerimo programske nîti, koje se smiju koristiti za navedeni proces (što ne znači da se i koriste odnosno ne znači da smo ih stvarno i dodijelili tom procesu), kako smo naučili u prvom primjeru od gore:
```
taskset -p -c 4071

pid 4071's current affinity list: 0-7
```
Dakle granica za ovaj proces (program/aplikaciju) su CPU jezgre 0 do 7.
Upravo zbog višenîtnosti može nam izgledati kao da se naš višenîtni (engl. *Multithread*) proces/program/aplikacija prebacuje između više CPU jezgri (kada pratimo opterećenje sustava).

Isti ispis možemo dobiti i s naredbom hwloc-ps, koja dolazi u softverskom paketu **hwloc**:
```
hwloc-ps -p --pid 4071
4071  PU:0 PU:1 PU:2 PU:3 PU:4 PU:5 PU:6 PU:7     /usr/bin/kvm
```
Dakle vidimo da je procesu s **PID** brojem **4071** (/usr/bin/kvm) dozvoljena upotreba logičkih jezgri procesora 0 – 7.


Za definiranje jezgri procesora na kojim se program stvarno i može izvršavati, pogledajte poglavlja:
**10.7.2.1.2. Process/Thread Affinity i CPU Affinity.**
**10.7.2.2.2. NUMA CPU i RAM afinitet.**

***Ako imate potrebu, možete poredati pokrenute programe (procese), primjerice prema iskorištenju CPU-a, ovako:***
```
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu
```


*Postoji i **TGID** broj (engl. Thread Group ID) koji koristi **PID** broj procesa, ako se radi o njegovoj procesnoj nîti, ali ako se radi o račvanju programa (engl. Fork), tada nova instanca dobiva drugi **PID** i **TGID** broj.*


I na kraju ponovno proučite poglavlje: **9.3 Task/process scheduler** i to dio ***context switching*** te detalje direktno vezane uz *context switching*, a odnose se na RAM memoriju i proces translacije adresa u poglavlju:
**12.3 Memorijski menadžment i virtualna memorija.**


Prikazani primjeri su za **KVM/QEMU** odnosno za virtualizaciju jer je CPU afinitet tamo potencijalno koristan!

**Izvori informacija: (K-4),** `man taskset`, `man ps`, `man pidof`.

### 10.7.2.2. NUMA

Što je **NUMA (engl. *Non-uniform memory access*)** odnosno sustav s nejednolikim pristupom memoriji?.
Na poslužiteljskim generacijama procesora (CPU) koji imaju i znatno veći broj jezgri unutar fizičkog procesora: 24, 28, 32, 36 ili više, moguće je na jednu matičnu ploču instalirati dva ili više fizičkih procesora, naravno, ako to podržava sâma matična ploča, ali i sâm procesor. Iako je ova tehnologija konceptualno nastala 1960-tih, pojavom prvih superračunala, njen dizajn se mijenjao do 1990-tih zbog drastičnih promjena u odnosima između brzine procesora u odnosu na brzinu RAM memorije.

Trenutni dizajn **NUMA** arhitekture su postavili ***AMD*** 2003 (s *Opteron* procesorima) te ***Intel*** kasnije 2007 na x86 (s *Nehalem* i *Tukwila* procesorima), a potom i ***Intel*** s *Itanium* arhitekturom procesora.

S obzirom na to da svaki fizički procesor u sebi ima ugrađen i memorijski kontroler, kao i svoju memorijsku sabirnicu s pripadajućom RAM memorijom, sada dolazimo do situacije kada za sustav s dva fizička procesora na matičnoj ploči imamo:

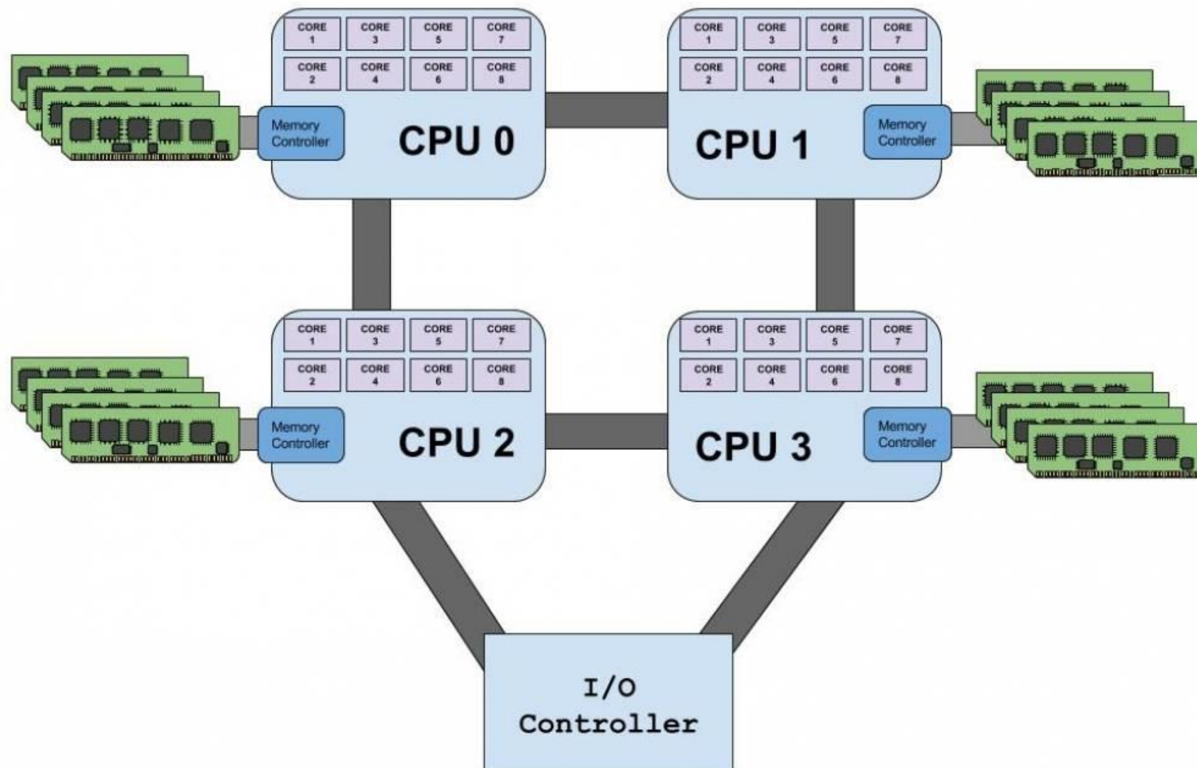
- Dva memorijska kontrolera: pri čemu svaki fizički procesor (CPU) ima svoj memorijski kontroler.
- Dvije memorijske sabirnice, a pri tome svaki fizički procesor ima svoju memorijsku sabirnicu.
- Dva kanala RAM memorija; pri tome svaki kanal RAM memorija pripada točno određenom fizičkom procesoru.
- Dodatnu sabirnicu (ili više njih) koja povezuje sve fizičke procesore, a pri tome ugrubo imamo dva rješenja: *AMD* i *Intel*, ali i nekoliko njihovih pôd varijanti:
  - ***AMD*** ovu sabirnicu naziva **HT** (engl. *Hyper Transport Interconnects*) ili noviju/bržu **IF** (engl. *Infinity Fabric)*.
  - ***Intel*** ovu sabirnicu naziva **QPI** (engl. *Quick Path Interconnect*) ili noviju/bržu **UPI** (engl. *Ultra Path Interconnect*).

Kod NUMA sustava s četiri ili osam fizičkih procesora to znači četiri ili osam memorijskih kontrolera i RAM memorija od kojih svaka pripada svom fizičkom procesoru, prema principu: jedan fizički procesor → jedan skup RAM memorija. Skup jer je moguća upotreba takozvane višekanalne RAM memorije na svakom fizičkom procesoru.

## Pogledajmo dva NUMA dizajna tvrtki *AMD* i *Intel za četvero procesorski sustav*:

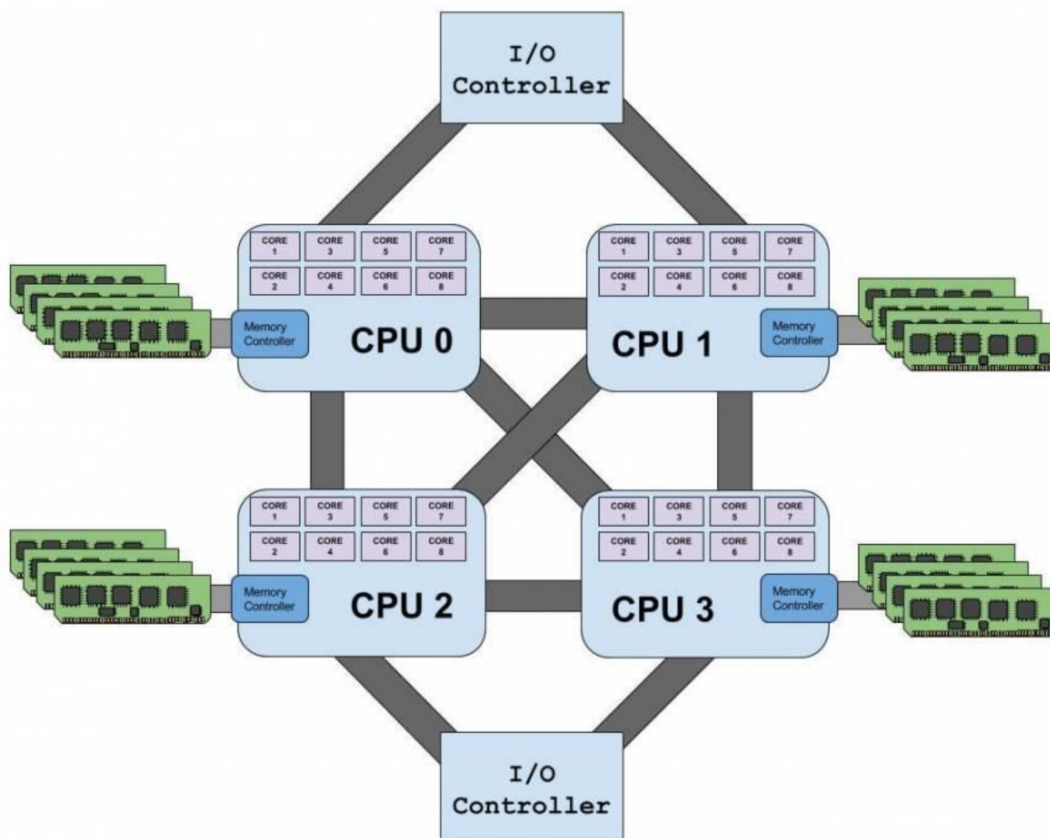
*1. Starije **AMD NUMA** (prije Quad-core - Abu Dhabi MCM (6308)) i starije **Intel NUMA** rješenje (**slika** 61)*

**Slika 61. Pogled na NUMA sustav (AMD-1).**




Kod ovakvog dizajna vidimo da je NUMA 0 (prvi fizički procesor [**CPU 0 na slici**]) spojen specijaliziranom seabirnicom s drugim NUMA fizičkim procesorom [**CPU 1** na slici] i trećim NUMA fizičkim procesorom [**CPU 2** na slici]i tako dalje. Dakle veza jest: CPU0 – CPU1 – CPU3 – CPU2 – CPU0. Dok samo NUMA 2 [CPU 2] i NUMA 3 [CPU 3] imaju vezu sa I/O kontrolerom i PCI i PCIe sabirnicama.

**2.** Novije ***AMD*** (nakon Quad-core - *Abu Dhabi MCM (6308)*) i novije ***Intel*** (od *Xeon 7000* serije (*Beckton*)) *NUMA* rješenje (***slika** 62*).
**Slika 62. Pogled na modernije NUMA sustave.**




Kod ovakvog dizajna svi fizički NUMA procesori, imaju međusobnu vezu: svaki-sa-svakim, što drastično:

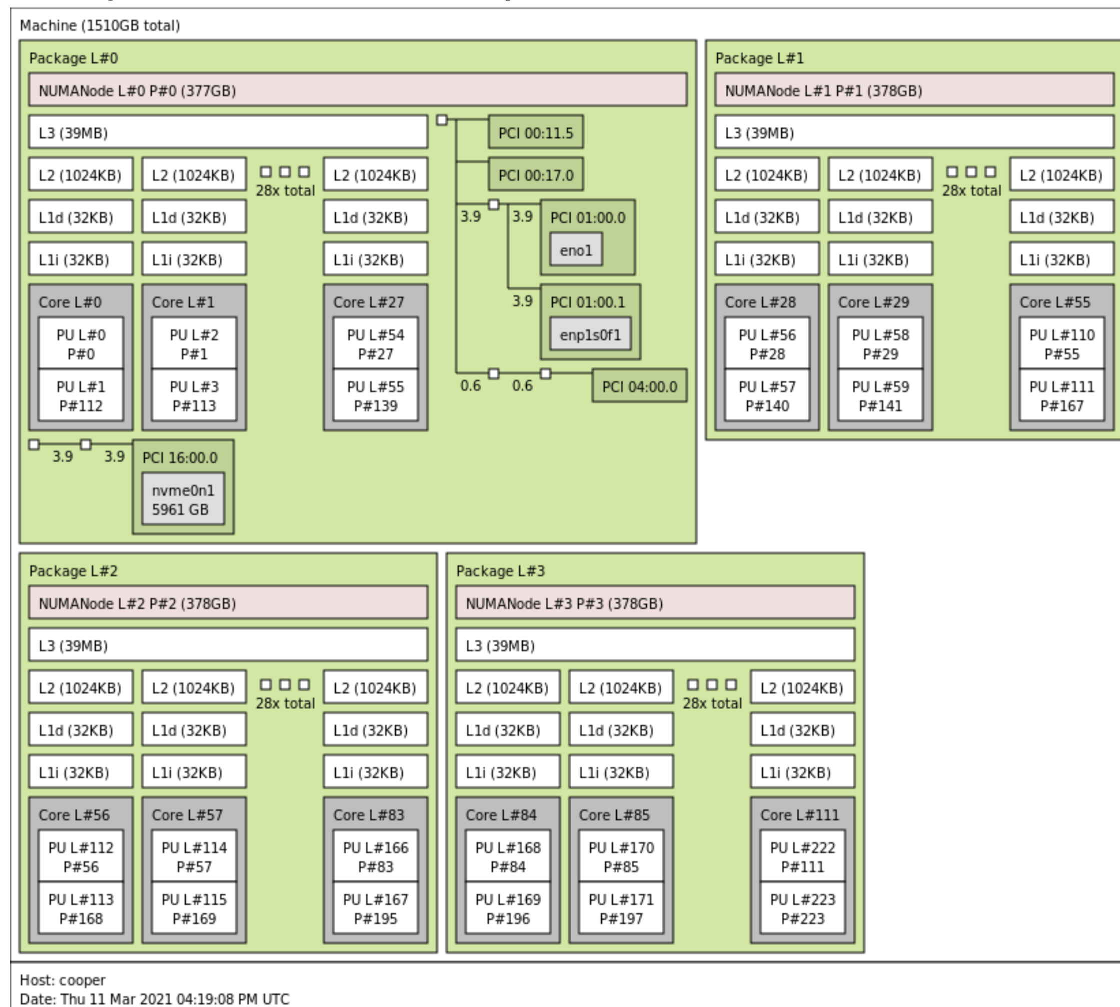
- Ubrzava njihovu međusobnu komunikaciju (bilo koja CPU jezgra s bilo kojom drugom CPU jezgrom).
    - Što direktno ubrzava i memorijsku komunikaciju: lokalna RAM memorija – udaljena RAM memorija. ← Pogledajte ostatak poglavlja, a posebno cjelinu ("**RAM memorija u NUMA arhitekturi**")
    - Ubrzava i pristup do **PCI** ili **PCIe** sabirnice i uređaja na njoj.



Vezano za NUMA I/O topologiju, pogledajte poglavlje:
**10.7.2.2.1. NUMA I/O topologija.**

Pogledajmo i logičku shemu ovakve generacije procesora (4 x *Intel Xeon Platinum 8380H*) s pogledom na logičke/fizičke jezgre i njihove **L1**, **L2** i **L3** međumemorije. Slika **62.A.** je kreirana s programom `lstopo`.

**Slika 62.A. Pogled na moderne NUMA sustave dobivene s naredbom lstopo.**




Ako pogledamo logičku shemu ovog procesaora (slika **62.A.**):

- Ovdje jasno vidimo fizičke procesore (**P#0 -P#4**) odnosno *NUMA node* 0 – 3 i njihove fizičke jezgre.
    - Primjerice na prvom fizičkom procesoru (***NUMA NODE 0*** tj. **P#0** vidimo):
        - **Core L#0 –** ovo označava jezgru s lokalnim identifikatorom (*Logical index*) **0** unutar fizičkog procesora. Lokalni identifikatori su redosljedni brojevi dodijeljeni jezgrama unutar svakog procesora. I njene logičke jezgre odnosno "Processing Unit-e" : ***PU L#0 i PU L#1***
            - **PU** (**Processing Unit**) **L#0** i **P#0**:
                - **L#0** – lokalni identifikator za procesnu jedinicu, koja može biti fizička ili logička jezgra (*hyper-thread*).
                - **P#0** - fizički identifikator za procesnu jedinicu. To je stvarni fizički broj procesne jedinice na fizičkom procesoru (NUMA NODE-u).
            - **PU (Processing Unit) L#1 i P#112:**
                - **L#1** – lokalni identifikator za drugu procesnu jedinicu unutar iste jezgre ili drugog *hyper-threada* na istoj jezgri.
                - **P#112** - fizički identifikator procesne jedinice, što znači da je ova jedinica stvarno 113 po redu procesna jedinica u ukupnom sustavu (počevši od 0).
        - **Core L#1 - . . .**

- Dakle vidimo i sve njihove logičke jezgre (u parovima) te priručne **L1**, **L2** i **L3** međumemorije.
- Osim toga vidljiva je i sabirnica te na koje fizičke procesore je spojena: konkretno na **Numa node 0 (P#0)**.

Slika odnosno shema je preuzeta sa stranice: https://www.open-mpi.org/projects/hwloc/lstopo/

**Kako to izgleda u praksi?**
Slika prikazuje noviju generaciju ***Intel Xeon*** matičnih ploča iz 2015. godine, baziranih na *Intel C602J chipsetu*, za *LGA2011 Xeon* procesore, poput: Intel Xeon E7-8870V3 sa 36 procesorskih jezgri unutar svakog fizičkog procesora.

**Slika 63. Matična ploča tvrtke Supermicro: X10QBL-4.**

Na slici 63. je vidljivo kako je na ovu matičnu ploču moguće ugraditi do četiri (1-4) fizička *Xeon* procesora *E7* generacije. Svaki od ovih procesora podržava četvero kanalni pristup RAM memoriji, preko svog internog memorijskog kontrolera i memorijske sabirnice.

Stoga svaki fizički procesor ima po dva nîza utora za RAM memoriju i to svaki od njih po četiri utora za dvokanalnu RAM memoriju gore (na slici) te po jedan nîz od četiri utora za RAM memoriju dolje (isto dvokanalni).

Dakle svaki fizički procesor (na slici) ima ukupno 8 utora za RAM memoriju, koji su raspoređeni u četiri kanala, što osigurava četiri puta veću propusnost u odnosu na jedno kanalnu memoriju.



Vezano za višekanalne memorije, pogledajte poglavlje:
**12.2.2.6. Višekanalna arhitektura memorije.**

**Na što još obratiti pažnju kod procesora - primjer za Intel (zbog lakše dostupnosti podataka od strane Intel-a)**
Svaki procesor koji podržava *NUMA* arhitekturu dizajniran je za *NUMA* sustav određene veličine.

Pogledajmo karakteristike nekoliko Intel *Xeon* “NUMA” procesora:

| **Model procesora** | **Broj QPI/UPI linkova (međuveza) koje podržava** | **Koliko se može skalirati (proširiti)** |
|---|---|---|
| Xeon E5-2620 | 2 | **2S** |
| Xeon E7-4830 v2 | 3 | **S4S** |
| Xeon E7-4830 v3 | 3 | **S4S** |
| Xeon E7-8860 | 4 | **S8S** |
| Xeon E7-8870 v3, Xeon Platinum 9282, … | 4 | **S8S** |

Podatak o skaliranju (**2S** ili **S2S**), (**S4S**) ili (**S8S**) nam govori kako su pojedini procesori dizajnirani za rad, odnosno kako se mogu ugraditi na matične ploče koje imaju:

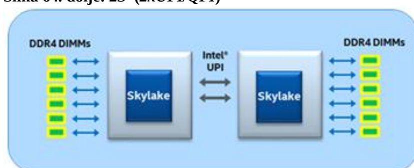
- Dva procesorska utora (**S2S** ili **2S**) – slika **64.**
- Četiri procesorska utora (**S4S**) – slika **65** ili **66.**
- Osam procesorskih utora (**S8S**) – slika **67.**

Dakle nije moguće pokrenuti sustav koji ima recimo četiri procesorska utora na matičnoj ploči, s procesorima koji ih podržavaju samo dva (pr. ***S2S*** procesori). Osim toga novije generacije procesora imaju sve veću propusnost ***QPI*** međuveza.

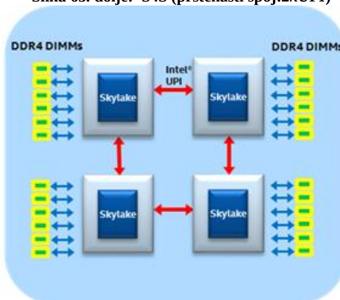
***Najnovije generacije*** *Intel* procesora uvode novu vrstu međuveza koja zamjenjuje ***QPI***, a zove se ***UPI*** (*Ultra Path Interconnect*). Ova tehnologija povećala je propusnost međuveze, ali i uvela malo drugačiji dizajn. Naime sada ovisno o namjeni procesora (**2S**, **S4S** ili **S8S**) procesori imaju 2 ili 3 ***UPI*** kanala.
Ovakvi ***Xeon*** procesori počeli su se proizvoditi negdje od 2017 godine. Jedan od predstavnika ove generacije procesora je ***Xeon Platinum 8164*** (*Skylake* arhitektura) koji podržava ***S8S*** konfiguraciju.

Pogledajmo ovaj novi dizajn, u konfiguracijama: ***2S***, ***S4S*** i ***S8S***.

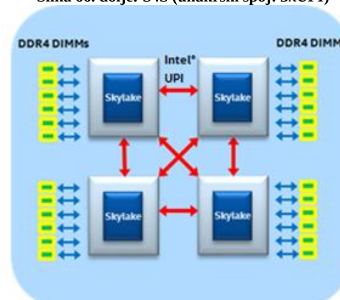
**Slika 64. dolje:** ***2S*** **(*2xUPI/QPI*)**




**Slika 65. dolje: S4S (prstenasti spoj:*2xUPI*)**



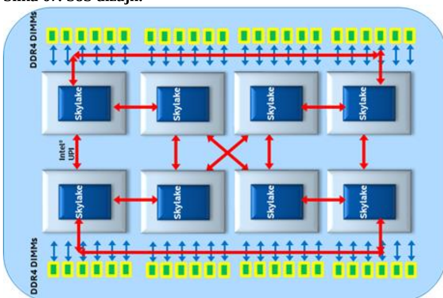

**Slika 66. dolje: S4S (unakrsni spoj: *3xUPI*)**




Izvor slika: https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview

I konačno pogledajmo **S8S** dizajn:

**Slika 67. S8S dizajn.**




Izvor slike: https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview

**U čemu je problem?**
U sâmom radu operativnog sustava, ovisno o tome koristi li se na ***SMP*** sustavu (jedan fizički procesor s više jezgri) ili ***NUMA*** sustavu (više fizičkih procesora od kojih svaki ima više procesorskih jezgri); očito postoje velike razlike.
Zbog toga što *NUMA* ima popriličan utjecaj na performanse kod pristupa RAM memoriji, potrebne su i intervencije na strani operativnog sustava, koji mora biti svjestan *NUMA* arhitekture. Osim toga potrebne su i promjene na razini softvera, vezano za višenîtni rad i pozicioniranje sâmih nîti i procesa (programa) što bliže fizičkom procesoru odnosno *NODE*-u prema NUMA terminologiji.



U *NUMA* terminologiji fizički procesor (CPU) se naziva ***NODE***.



*Podrška za NUMA tehnologiju se počela razvijati s Linux kernelom v.2.4, a buduće inačice (danas trenutne) v.3.x, 4x i 5x donijele su nam robusniju i stabilnu podršku. To znači kako već od **LTS** inačica linux kernela v.2.6.32 na dalje, Linux proces/task scheduler u kombinaciji s upravljačkim programima i modernim aplikacijama pouzdano odrađuje sve zadatke koje pred njih postavlja NUMA tehnologija odnosno arhitektura. Linux virtualizacija (**KVM**+**QEMU**) također podržava NUMA*
.

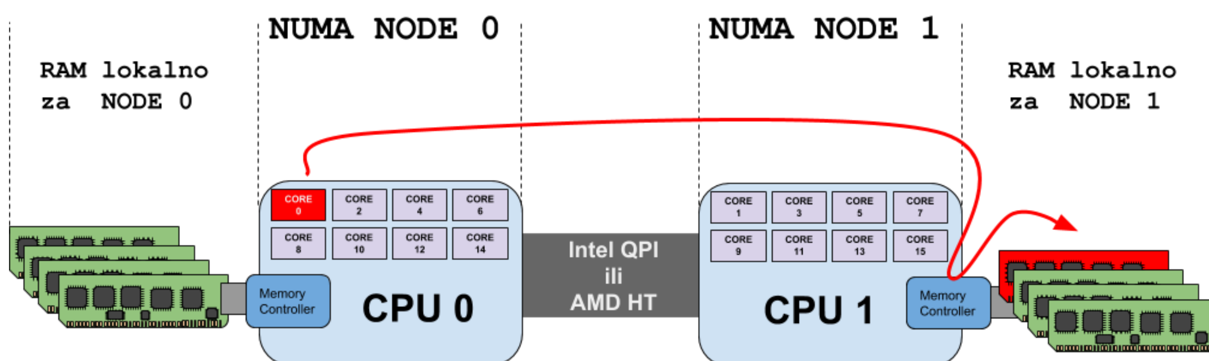


Manifestacije loše upotrebe ili konfiguracije *NUMA*-e, mogu imati za posljedicu sporiji rad ili usporavanja aplikacija ili u slučaju virtualizacije, usporavanja rada virtualnih računala.

## RAM memorija u NUMA arhitekturi

Pogledajmo jedan loš primjer upotrebe RAM memorije u *NUMA* arhitekturi (slika 68) :

**Slika 68. Pogled na NUMA arhitekturu i korištenje RAM memorije**




Na slici je vidljiv slučaj kada proces/program/aplikacija pokrenuta na prvom fizičkom procesoru (**NODE 0**) i to na njegovoj prvoj CPU jezgri (**CORE 0** - označeno crvenom bojom) u radu normalno koristi RAM memoriju.
Pošto o ovom slučaju, konkretan proces/program/aplikacija nije svjesna NUMA arhitekture, on može (i vrlo često hoće) koristiti RAM memoriju, koja nije lokalna njegovom fizičkom procesoru (**NUMA NODE 0**) i njegovom memorijskom kontroleru, već drugom fizičkom procesoru.

Zbog toga što RAM memorija koju je on počeo koristiti nije njemu lokalna, on mora preko posebne međuveze između fizičkih procesora (*Intel* **QPI** ili *AMD* **HT**) i to preko drugog fizičkog procesora (**NUMA NODE 1**) i njegovog memorijskog kontrolera dohvatiti tu RAM memoriju. Ova RAM memorija je prema tome lokalna (engl. *Local*) za **NUMA NODE 1**, ali je za **NUMA NODE 0** ona udaljena (engl. *Remote*). Dakle naša aplikacija ovdje konkretno koristi udaljenu RAM memoriju.
Jasno je kako ovo unosi kašnjenje:

- Prvo prolazom između lokalnog procesora prema drugom procesoru (**NODE 0 → NODE 1**).
- Potom korištenjem drugog memorijskog kontrolera (s drugog procesora) i njegove pripadajuće RAM memorije.
- Zatim prijenosom podataka u suprotnom smjeru. Ovaj proces se ponavlja tijekom svakog pojedinog pristupa RAM memoriji za ovakvu aplikaciju (koja koristi udaljenu RAM memoriju).

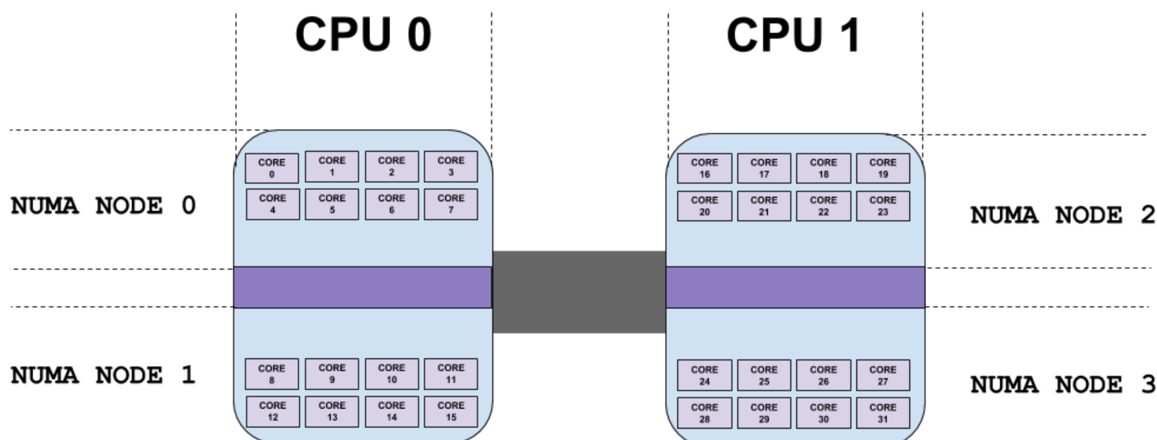
Prema nekim mjerenjima, razlike u latenciji (kašnjenju) između pristupa lokalnoj RAM memoriji i NUMA udaljene memorije dosežu i do 50%.

**Vratimo se NUMA arhitekturi**
Ovisno o dizajnu odnosno internoj arhitekturi procesora, ovisi i kako Linux numerira jezgre procesora. Naime sljedeća logika označavanja jezgri procesora ne mora biti točna za fizički procesor (CPU) s osam jezgri, na sustavu s dva fizička procesora:

- CPU 0 (***NODE 0***):
  - CPU jezgre: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**
- CPU 1 (***NODE 1***):
  - CPU jezgre: **8**, **9**, **10**, **11**, **12**, **13**, **14**, **15**

Sljedeća slika predstavlja logičku shemu AMD *NUMA* topologije koja je u upotrebi na "*Abu-Dhabi*" arhitekturi (*Opteron 6300*) procesora, gdje je numeriranje malo logičnije. Važno je uočiti kako su u ovoj arhitekturi povezana dva procesora na istoj pločici unutar fizičkog procesora označenog kao **CPU 0**, što je slučaj i kod drugog fizičkog procesora (**CPU 1**).
Iako imamo dva fizička procesora (**CPU 0** i **CPU 1**) oni su proizvedeni tako da se zapravo u svakom fizičkom pakiranju procesora nalaze povezana dva nezavisna procesora kao na slici 69.

Slika 69. AMD „Abu-Dhabi“ NUMA arhitektura.




Tvrtka **AMD** ovakvo grupiranje procesorskih jezgri i njihovih pripadajućih priručnih međumemorija (**L1**, **L2** i **L3**) u nezavisni sklop naziva **CCX** (engl. *core complex*). Dok **CCD** predstavlja fizički *čiplet* (fizički procesor) koji sadrži jedan ili više **CCX**-ova.

Slika 69.A. AMD Zen arhitektura procesora (na slici je Ryzen 9 procesor). Autor slike je tvrtka AMD.




Na slici vidimo fizički procesor koji se sastoji od dvije pločice, a unutar svake od njih su po dva **CCX** sklopa.
**CCX** sklopovi imaju po četiri fizička procesora, od kojih svaki ima pod dvije logičke procesorske jezgre.

**CCX** sklopovi unutar pločice povezani su posebnom **Infinity Fabric** sabirnicom, kao i pločice među sobom.

Međutim numeriranje jezgri u ***NUMA*** arhitekturi, na Linuxu može varirati, ovisno o modelu odnosno dizajnu vašeg procesora. Vrlo često je slučaj sljedećeg numeriranja, za fizički procesor s osam (logičkih) jezgri:

- **CPU 0** (*NODE 0*):
  - CPU jezgre: **0**, **2**, **4**, **6**, **8**, **10**, **12** i **14**
- **CPU 1** (*NODE 1*):
  - CPU jezgre: **1**, **3**, **5**, **7**, **9**, **11**, **13 i 15**

**Kakva je trenutna podrška za NUMA arhitekturu kod aplikacija i programskih jezika za Linux?**

- Linux ***KVM** hipervizor* i QEMU imaju podršku za NUMA i odličan NUMA *scheduler*. **Qemu** od v.3.x. ima riješen NUMA *scheduler* koji je u rangu *Enterprise* rješenja komercijalnih proizvođača platformi za virtualizaciju, a novije inačice imaju sve dostupne mehanizme i opcije potrebne za NUMA arhitekturu odnosno za njeno fino podešavanje.
- Programski jezik ***Java*** od inačice **7** ima podršku za "*NUMA-aware*" alokaciju RAM memorije.
- Jezici poput **C** i **C++** obično zahtijevaju upotrebu biblioteka poput **numa** ili **libnuma**.
- Jezik **Rust** podržava rad s NUMA arhitekturom upotrebom biblioteka (pr. **libnuma-sys**), …

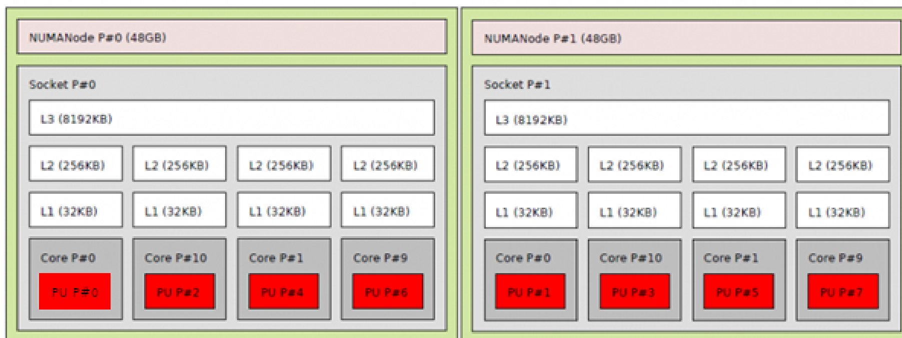
## NUMA Statistike i informacije

Za provjeru dostupnosti NUMA procesora te njihove raspored, možemo pokrenuti naredbu: `numactl --hardware`.
Ponekad je potrebno instalirati poseban softverski paket za rad s NUMA parametrima, statistikama i postavkama:

```
yum -y install numactl
```

Pogledajmo malo stariji poslužitelj s dva fizička procesora (*Intel Xeon E5606*), od kojih svaki fizički procesor ima četiri fizičke jezgre (nema logičkih jezgri). Izgled navedenog *Intel Xeon* procesora (*E5606*) je vidljiv na slici 70.

**Slika 70. Logički izgled Intel Xeon E5606 procesora**




Slika je napravljena s naredbom `lstopo`.
Jezgre procesora su označene crveno (**PU P#**BROJ JEZGRE).

Pogledajmo i kako to izgleda prema stvarnoj *NUMA* dijagnostici, upotrebom naredbe `numactl`:

```
numactl --hardware

available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6
node 0 size: 49141 MB
node 0 free: 7227 MB
node 1 cpus: 1 3 5 7
node 1 size: 49151 MB
node 1 free: 7278 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

Dakle vidimo kako imamo dva fizička procesora (`available: 2 nodes (0-1)`).
Osim toga vidljivo je i to da su na:

- Prvom fizičkom procesoru (`node 0`), dostupne jezgre, koje Linux vidi kao:
  - Jezgre: **0, 2**, **4** i **6**.
- Drugom fizičkom procesoru (`node 1`), dostupne jezgre, koje Linux vidi kao:
  - Jezgre: **1**, **3**, **5** i **7**.

Također vidimo kako:

- Prvi fizički procesor (`node0`) ima:
  - 49.141 MB dostupne lokalne RAM memorije.
  - 7.227 MB slobodne lokalne RAM memorije.
- Drugi fizički procesor (`node1`) ima:
  - 49.151 MB dostupne lokalne RAM memorije.
  - 7.228 MB slobodne lokalne RAM memorije.

## *Udaljenost između fizičkih procesora (Node distances)*

Tablica na kraju prethodnog ispisa prikazuje takozvane ***SLIT*** (engl. *System Locality Information Table*) i ***SRAT*** (engl. *Static Resource Affinity Table*) tablice, ako su podržane od strane procesora. U našem primjeru gore nije podržana *SLIT* tablica jer se radi o vrlo starom procesoru, te nemamo točna očitanja latencije (kašnjenja) između fizičkih procesora.
***SRAT*** tablica pokazuje nam kojemu fizičkom procesoru pripada koji blok RAM memorije. Ovaj pojam se zove i "*proximity domain*" prema ***ACPI***-ju. Fizički procesor se naziva i "**NODE**" prema NUMA terminologiji, kako smo već naučili.
Udaljenost odnosno vremena kašnjenja (engl. *Delay*) između fizičkih procesora (**NODES**) su upisana u **SLIT** tablicu.

Na novijim procesorima **SLIT** i **SRAT** tablicama sustav može pristupiti preko **ACPI** (engl. *Advanced Configuration and Power Interface*) poziva. **NODE** se sastoji od jednog fizičkog procesora i njegove pripadajuće RAM memorije.
Svi NODE-ovi (fizički procesori) su povezani među sobom (ovisno o implementaciji i proizvođaču kako smo već prije vidjeli):

- Serijskom vezom (direktnom vezom jednog CPUa s drugim), ako imamo dva fizička procesora.
- U topologiji prsten, ako ih imamo četiri ili više fizičkih procesora.
- U "*Mesh*" topologiji (unakrsni spoj) po principu svaki sa svakim: ako ih imamo četiri ili više fizičkih procesora.

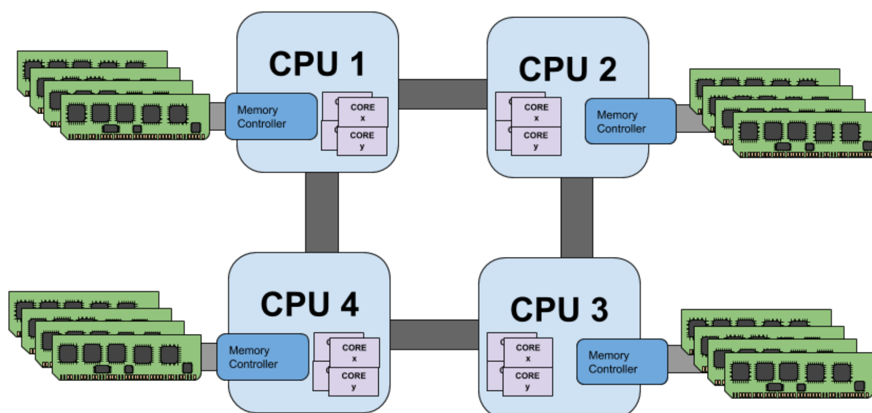
Ako se radi o više od dva procesora u prstenastom NUMA spoju to znači kako pristup RAM memoriji s NODE 0 (CPU 1) na RAM memoriju NODE 1 (CPU 2) procesora ima samo prolaz kroz NODE 1 (CPU 2) procesor.
S druge strane pristup RAM memoriji s NODE 0 (CPU 1) na RAM memoriju NODE 2 (CPU 3) zahtjeva prolaz kroz još jedan procesor odnosno kroz NODE 1 (CPU 2).

U složenijim NUMA arhitekturama, koristi se međuveza prstenastog spoja: **1→ 2 → 3 → 4 → 1**.
Slika 71. (dolje) prikazuje prstenasti spoj četiri procesora.
Dakle ovdje imamo četiri fizička procesora (CPU) te se ovdje radi o *NUMA* topologiji prsten (slika 71):

```
node distances:
node   0   1   2   3
  0:  10  12  17  12
  1:  12  10  12  17
  2:  17  12  10  12
  3:  12  17  12  10
```

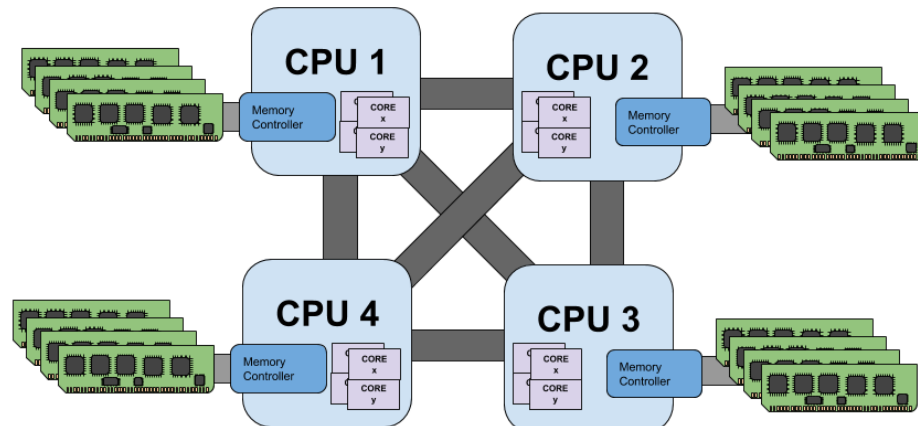
Ova tablica nam govori da, ako NUMA NODE 0 (CPU 1) pristupa svojoj lokalnoj RAM memoriji, da mu za to treba 1,0 ns (nano sekunda). Ako isti procesor želi pristupati RAM memoriji NUMA NODE 1 (CPU 2), za to mu je potrebno malo više vremena: to jest 1,2ns, a ako želi pristupati RAM memoriji od NUMA NODE 2 (CPU 3), tada mu za to treba čak 1,7 ns.

**Slika 71. Prstenasti NUMA spoj procesora.**




Slijedeća slika 72. (dolje) prikazuje ***Mesh*** odnosno isprepleteni spoj četiri procesora.
Kod ***Mesh*** NUMA topologije svaki fizički procesor ima direktnu vezu s bilo kojim drugim fizičkim procesorom pa prema tome kašnjenje može biti sâmo s jednog preko drugog procesora (nikad nema onog trećeg u nîzu), ali su za to potrebni procesori koji to podržavaju. Za procesore tvrtke *Intel* to su procesori koji maju tri **QPI** veze, primjerice: *Intel Xeon E7 v4* [pr. *E7-8894 v4*]).

**Slika 72. Ukrižani odnosno isprepleteni (*mesh*) NUMA spoj procesora.**




Na novijim procesorima koji nam daju informacije o **SLIT**, tablica nam pokazuje udaljenost odnosno kašnjenja ovisno s kojeg fizičkog procesora pristupamo kojoj RAM memoriji: lokalnoj memoriji našem CPUu ili udaljenoj memoriji koja pripada drugom CPUu.

Da bi NUMA arhitektura uopće radila, osim podrške u hardveru potrebna je i podrška u kernelu Linuxa, odnosno kernel mora bit kompiliran s podrškom za nju. Podršku u kernelu za NUMA možemo provjeriti sa:

```
grep CONFIG_NUMA= /boot/config-`uname -r`
CONFIG_NUMA=y
```

← pošto ovdje imamo =y to znači da ju naš kernel podržava (što je najčešće zadana postavka).

## Automatsko NUMA balansiranje

Automatsko NUMA balansiranje premješta izvršavanje programa bliže NUMA jezgrama i samim time memoriji kojoj pristupaju. Također se premještaju podaci programa (aplikacije) u memoriju bliže zadacima koji ih kreiraju. Sve to automatski radi kernel kada je aktivirano automatsko NUMA balansiranje. U tu svrhu postoje *sysctl* varijabla i datoteka u `/proc` pseudo datotečnom sustavu koja kontrolira automatsko uravnoteženje (balansiranje) NUMA čvorova. Pogledajmo njen status na našem poslužitelju:

```
sysctl kernel.numa_balancing
kernel.numa_balancing = 1
```

← `1` znači da je uravnoteženje omogućeno (što je očekivano i zadano na NUMA sustavima). Nula (`0`) bi značila da je NUMA uravnoteženje isključeno. Dok postoji i opcija (`2`) koja je slična opciji **1**, ali kod nje će kernel automatski omogućiti uravnoteženje NUMA čvorova samo kada je to potrebno, na primjer, kada se otkrije nejednako opterećenje ili problemi s performansama.

Datoteka u pseudo datotečnom sustavu `/proc` koja kontrolira isto automatsko NUMA balansiranje je `numa_balancing`:

```
cat /proc/sys/kernel/numa_balancing
1
```

Razlike između ručnog odabira točno određenih CPU jezgri (***CPU afinitet***) za određene programe (procese) u odnosu na automatsko NUMA balansiranje, za određene slučajeve primjene pokazuje vrlo male razlike (1-2%), dok za neke druge primjene nije toliko zanemariva: 5-15% ili čak 10-25%. **Za realnija mjerenja testiranje napravite sami prije puštanja sustava u rad!**

*Izvor mjerenja: „Automatic NUMA Balancing“, Autori: Rik van Riel, Principal Software Engineer Red Hat, Vinod Chegu, Master Technologist, HP.* **Izvor: (1553)**

## Vratimo se na naš stariji dvoprocesorski sustav

Ako želimo vidjeti listu svih dostupnih CPU jezgri i dodatne informacije, za to možemo koristiti naredbu `numactl`, ovako:

```
numactl --show

policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7
cpubind: 0 1
nodebind: 0 1
membind: 0 1
```

Vidljivo je kako imamo ukupno osam (8) CPU jezgri na sustavu (`physcpubind: 0 1 2 3 4 5 6 7`) podijeljenih u dvije grupe odnosno “*memory nodes*”, a pošto i imamo dva NODE-a tj. dva fizička CPU-a, to su onda: `0` i `1`

Za ručnu provjeru *NUMA* statistike, u slučaju kada imamo dva fizička procesora, možemo pokrenuti ovaj nîz naredbi u jednom retku, a koji će nam lijepo formatirati ispis statistika:

```
echo "Message : Node0 , Node1 "; join /sys/devices/system/node/node0/numastat
/sys/devices/system/node/node1/numastat |awk '{print $1 ":" , $2 ",", $3 }'
```

Opis: čitamo statistike koje su vidljive za svaki *NUMA node* (fizički CPU). Statistike su dostupne u datoteci:

- `/sys/devices/system/node/node0/numastat` za prvi *NUMA node* (broji se od 0).
- `/sys/devices/system/node/node1/numastat` za drugi *NUMA node.*
- Naredba `join` povezuje statistike iz obje datoteke u jednu (pošto obje datoteke imaju istu strukturu - stupaca i redova a mijenjaju se samo “brojevi”).
- Naredbom `awk` samo filtriramo ispis.



Pogledajte primjere upotrebe naredbe **`join`** u poglavlju:
**5.7.6. Naredbe *join* i *paste*.**

Rezultat će biti nešto slično ovome, ako je sve uredu s *NUMA*, odnosno ako svi programi/procesi/aplikacije uredno rade s ***NUMA*** arhitekturom:

```
Message : Node0 , Node1
numa_hit: 31413614, 16686285
numa_miss: 0, 0
numa_foreign: 0, 0
interleave_hit: 20108, 20261
local_node: 31413376, 16664958
other_node: 238, 21327
```

Ovdje nam je važno da `numa_miss` i `numa_foreign` budu nula (**0**), što znači da nije bilo prebacivanja/korištenja RAM memorije s krivog fizičkog procesora. Osim ovog ručnog rada, moguće je koristiti i naredbu `numastat`.

To onda izgleda ovako:

```
numastat

                           node0           node1
numa_hit              4144953406      3911786342
numa_miss                      0               0
numa_foreign                   0               0
interleave_hit             33306           33295
local_node            4144950529      3911749267
other_node               6648721       124510471
```

Pogledajmo i jedan loše konfiguriran **NUMA** sustav. Ovdje vidimo da su vrijednosti: `numa_miss` i `numa_foreign` velike:

```
numastat

                           node0           node1
numa_hit              4401481650      4152273756
numa_miss                7270135       143548866
numa_foreign           143548866         7270135
interleave_hit             33306           33295
local_node            4401478648      4152236667
other_node               7273137       143585955
```

U nekim, posebno rijetkim slučajevima, ako imate iznimno veliki broj aplikacija kod kojih vam nije važno odnosno kada ne želite da se sustav brine o tome da se svaka aplikacija pokreće samo na jednom ***NUMA*** procesoru s pripadajućim CPU jezgrama, možemo sustavu naložiti da isključi optimizaciju prema ***NUMA*** arhitekturi.

Ovo se podešava, mijenjanjem navedene posebne *sysctl* varijable imena: `kernel.affinity_load_balancing` odnosno postavljanjem iste u vrijednost jedan (`1`).
Naravno, ako je ta opcija u kernelu uopće dostupna, što ovisi o inačici kernela i kako je kernel kompiliran.
Pogledajmo kako uključiti ovu (uglavnom lošu) funkcionalnost, pomoću `sysctl` naredbe:

```
sysctl kernel.affinity_load_balancing=1
```

Važno je znati da je svaki kernel kompiliran da podržava određeni maksimalan broj CPU jezgri, što se može vidjeti sa:

```
grep -i CONFIG_NR_CPUS= /boot/config-`uname -r`
CONFIG_NR_CPUS=8192
```

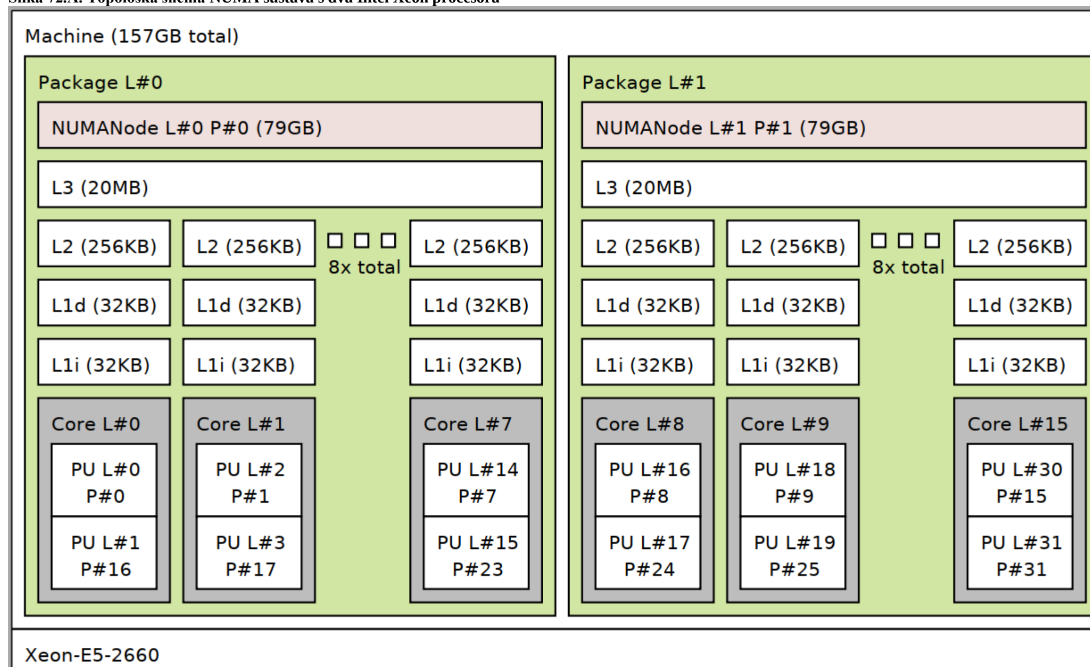
← Konkretni kernel podržava maksimalno 8192 CPU jezgre na računalu, što možemo vidjeti i ovako:

```
cat /sys/devices/system/cpu/kernel_max
8191
```

## Analiza rada programa u NUMA okruženju

Sada pogledajmo malo moderniju NUMA arhitekturu odnosno sustav s dva *Intel Xeon* procesora (***NUMA Node P#0 i P#1***).

**Slika 72.A. Topološka shema NUMA sustava s dva Intel Xeon procesora**




Dakle ovdje vidimo NUMA arhitekturu računala koje ima dva fizička procesora Intel Xeon E5- 2660: (NUMA **Node P#0** i **Node P#1**). Svaki fizički procesor ima osam fizičkih jezgri, a svaka fizička jezgra ima po dvije logičke jezgre, što ukupno daje 32 logičke jezgre (**PU**: **P#0** – **P#31**).

Pogledajmo ovaj sustav i ovako:

```
numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 80532 MB
node 0 free: 53730 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 80576 MB
node 1 free: 64402 MB
```

Vidimo dva fizička procesora od kojih prvi (**node 0**) ima logičke CPU jezgre: `0,1,2,3,4,5,6,7` i `16,17,18,19,20,21,22,23` dok drugi (**node 1**) ima logičke CPU jezgre: `8,9,10,11,12,13,14,15` i `24,25,26,27,28,29,30,31`.

Naš poslužitelj koji promatramo je poslužitelj za virtualizacijsku platformu **Proxmox VE**, koja kao *hipervizor* koristi QEMU+KVM. U ovom primjeru pokrenuli smo jedno virtualno računalo, vidljivo kao proces imena **KVM** čiji *Proxmox ID* je **106**, po kojemu ćemo i raditi pretragu.

Pronađimo ***PID*** broj tog (***kvm***) programa to jest procesa:

```
ps -ef | grep kvm | grep 106
…. 3023175 … /usr/bin/kvm …
```

Dakle njegov ***PID*** broj je `3023175`.

Sada ćemo s naredbom `numastat` vidjeti statistike koliko taj proces (program) koji je zapravo virtualno računalo, koristi, i koje NUMA node-ove odnosno njihove pripadajuće RAM memorije:

```
numastat -p 3023175

Per-node process memory usage (in MBs) for PID 3023175 (kvm)
                           Node 0          Node 1           Total
                ---------------- --------------- ---------------
Huge                       0.00            0.00             0.00
Heap                       4.59            5.03             9.62
Stack                      0.00            0.95             0.95
Private                    910.01          211.70           1121.71
----------------  --------------- --------------- ---------------
Total                      914.61          217.67           1132.28
```

← Ovdje sumirano vidimo da naš program ***KVM*** (+*QEMU*) s ***PID*** brojem `3023175` koristi oba fizička procesora (*Node 0* i *Node 1*) to jest logičke CPU jezgre i njihove pripadajuće RAM memorije na oba procesora (**914** vs. **217** MB/s).

Upotreba i neuravnoteženost nisu nužno problem, osim kod programa ili u ovom slučaju virtualnih računala, koja potencijalno zahtijevaju ekstremno brze performanse po pitanju dohvaćanja podataka iz RAM memorije, ali i upotrebe željenih CPU jezgri (zbog ***context switcha***).

Na sličan način smo mogli usporediti rad svih virtualnih računala koje pokreće proces imena `kvm`:

```
numastat -c /usr/bin/kvm

Per-node process memory usage (in MBs)
PID                    Node 0  Node 1   Total
-------------      ------ ------  -----
2200 (kvm)             1225     3087   4313
8317 (kvm)        219 2064   2283
8488 (kvm)             1119     1656   2776
8574 (kvm)            16344     44     16388
3023175 (kvm)     853 1096   1949
-------------      ------ ------  -----
Total                   19761 7947   27709
```

Postoji i naredba `numatop`, s kojom je moguće pratiti ponašanje programa i cijelog sustava u stvarnom vremenu.



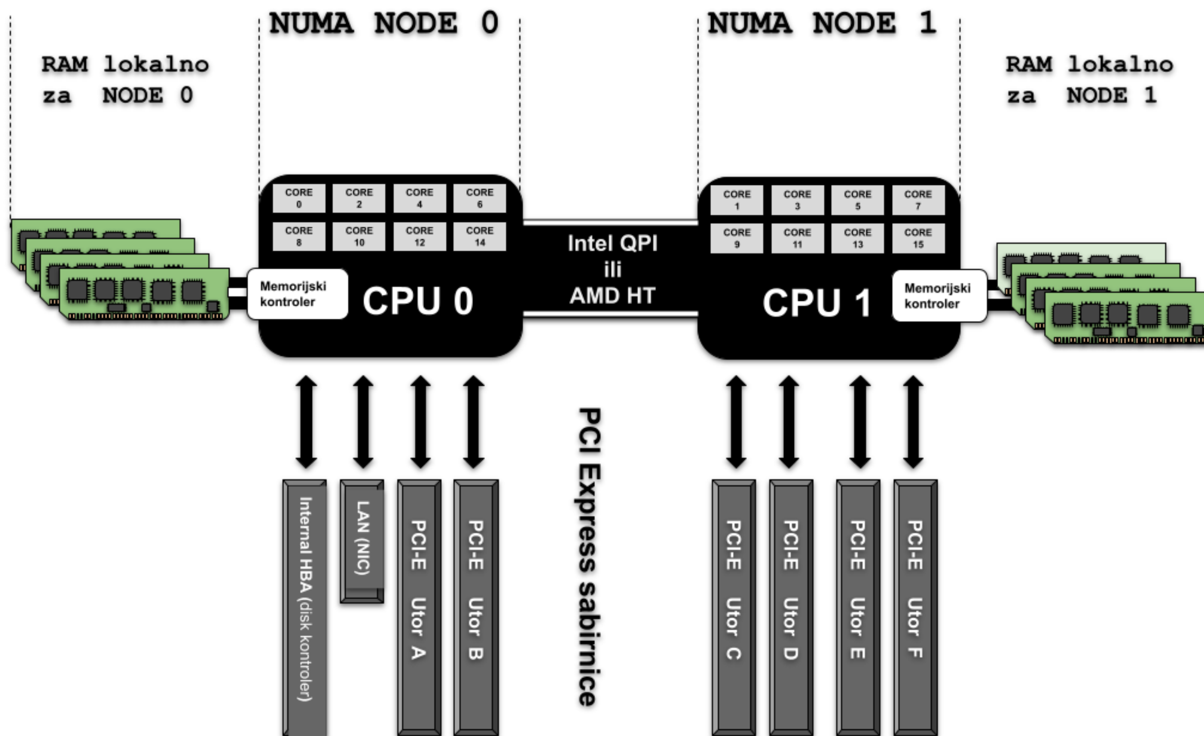
Pokretanje programa upotrebom točno definiranih CPU jezgri (***CPU afinitet***) je moguće rješenje, stoga pogledajte: **10.7.2.2.2. NUMA CPU i RAM afinitet.**

**Izvori informacija: (119),(979),(980),(1553),(K-9),** **`man numastat`**,**`man numactl`**,**`man sysctl`**,**`man 7 numa`**, **`man 5 sysfs`**,**`man lstopo`**,**`man numatop`**.

#### 10.7.2.2.1. NUMA I/O topologija

Ulazno izlazna komponenta *NUMA*-e odnosno *NUMA* I/O je vezana uz *NUMA node (fizički procesor)*, slično kao što je memorijski kontroler i njegova lokalna RAM memorija vezana uz svoj *NUMA node*. Dakle svaki fizički procesor osim što ima svoj memorijski kontroler i svoju lokalnu RAM memoriju, također ima i pristup na svoju lokalnu sabirnicu, poput primjerice ***PCI Express*** sabirnice. Stoga već u *NUMA* sustavima s dva fizička procesora (dva *NUMA node*-a) imamo i nerijetko dvije odvojene *PCI Express* sabirnice: svaka je spojena na svoj fizički procesor, kao na slici 73 (dolje).

**Slika 73. NUMA arhitektura i sabirnice.**




Slika prikazuje arhitekturu *Dell PowerEdge R720* poslužitelja s dva fizička procesora *Intel Xeon E5-2600 (**NUMA** arhitektura).*

Topološku shemu eksportiranu u PDF datoteku smo na našem poslužitelju mogli napraviti i sami s naredbom `lstopo`:

```
lstopo --whole-io posluzitelj.pdf
```

**Ponovno dolazimo do potencijalnih problema kao i s RAM memorijom u NUMA arhitekturi**!
Nakon inicijalizacije BIOS-a, BIOS operativnom sustavu (Linuxu) dalje prosljeđuje informacije o tome koji *PCI Express* (***PCIe***) uređaj pripada kojem NUMA *node*-u odnosno kojem fizičkom procesoru. Drugim riječima, koji fizički procesor kontrolira odnosno upravlja, s kojom sabirnicom. Pošto se na *PCI Express* sabirnici nalaze razni uređaji: od disk kontrolera, mrežnih, grafičkih i drugih kartica, vrlo je važno da je i operativni sustav toga svjestan, te da su u konačnici i upravljački programi koji se prema Linux terminologiji zovu "*kernel moduli*", također svjesni NUMA arhitekture.
Ovaj problem se proteže sve do procesa/programa/aplikacija koji koriste ove odnosno praktično sve uređaje.
Ako u ovu jednadžbu uvedemo i sabirnicu, sada imamo još jednu dodatnu komponentu u komunikaciji:

**Uređaj ←→ PCI Express sabirnica ←→ NUMA Node (CPU) ←→ BIOS ←→ upravljački programi (driveri) ←→ Linux kernel←→ ←→ programi/aplikacije (procesi)**

I naravno svi u nîzu moraju biti svjesni NUMA arhitekture. Na svim novijim linux kernelima (2.6.32 ili noviji) sve komponente sustava su dovoljno "svjesne" NUMA i NUMA ulazno izlaznog sustava (NUMA I/O) te su uglavnom jedine potrebne optimizacije vezane za NUMA CPU afinitet u smjeru:

- Linux program ←→ NUMA I/O ←→ [**IRQ**] - **Uređaj** (*primjerice: mrežna ili grafička kartica, disk kontroler i drugo*).

## IRQ afinitet

IRQ zahtjevi odnosno signali za prekid su zahtjevi za određenom uslugom odnosno zahtjevi poslani na hardverskoj razini u kojima neki uređaj traži pozornost sustava (obično za dohvaćanje ili prijenos podataka). Signali prekida se mogu slati ili namjenskom hardverskom linijom ili preko hardverske sabirnice kao informacijski paket (prekidi poznati kao **MSI**).

Vezano za NUMA I/O afinitet jasno je da bi najbolje bilo za su za IRQ (i uređaje na koje se odnosi) zadužene CPU jezgre najbliže samom uređaju odnosno sabirnici na koju je uređaj i spojen.

Kada se signal prekida dogodio kernel dohvaća IRQ broj i njegov pridruženi popis registriranih rutina usluga prekida (**ISR**) te poziva svaki *ISR* redom. Ovu obradu (IRQ) odrađuje određena jezgra procesora, a uobičajeno je na sustavima s više procesorskih jezgri, da različite jezgre odrađuju različite signale prekida. Ono što je nama u ovom razmatranju zanimljivo jest da i hardverski signali prekida (IRQ-ovi) imaju svojstvo afiniteta. Tako za svaki signal prekida postoji posebna datoteka imena: `smp_affinity`, koja definira CPU jezgre kojima je dopušteno izvršiti *ISR* za taj *IRQ*.

Ovo se svojstvo može koristiti za poboljšanje izvedbe aplikacije dodjeljivanjem afiniteta signalima prekida i afiniteta aplikacijske nîti na jednoj ili na više specifičnih CPU jezgri. Dakle kako bi se u NUMA sustavima poboljšala skalabilnost i smanjilo kašnjenje, preporučuje se raspodjela IRQ-ova između dostupnih CPU jezgri ili upotrebom servisa **irqbalance** (*zadano*) ili ručnom konfiguracijom, što jest čest slučaj za sustave koji ekstremno brzo moraju odrađivati signale prekida bez preraspodjele signala prekida koje radi navedeni servis **irqbalance**.



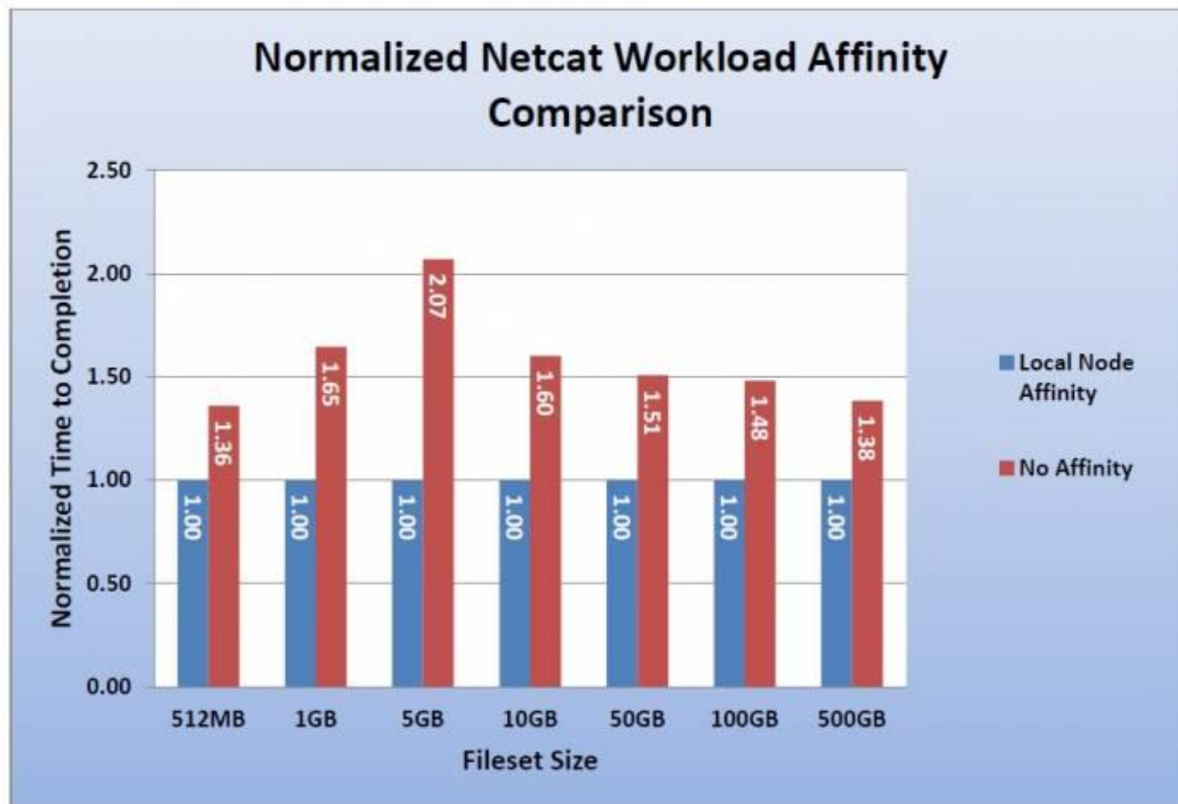
Za više detalja o hardverskim signalima prekida (IRQ) te primjerima promjene, pogledajte poglavlja:
**10.5.1.1. IRQ i Linux.**
**10.5.1.2. Servis *irqbalance*.**
**25.2.2. Multiqueue funkcionalnost.**

## CPU afinitet

I ovdje se sve svodi na pravilno pozicioniranje ***CPU afiniteta***: za program koji zahtjeva vršne performanse, treba ga pokrenuti na istom *NUMA node-u*, preko kojeg koristimo NUMA I/O operacije. To znači da, ako imamo mrežno vrlo intenzivnu aplikaciju odnosno program (pr. na brzinama 10+ Gbps), a mrežna kartica koju ona koristi se nalazi na NUMA I/0 koji je spojena na *NUMA Node 0*, tada bi se i ta aplikacija trebala nalaziti odnosno pokretati na istom NUMA *node-u*. Točnije na bilo kojoj CPU jezgri na tom *NUMA node-u*. Očekivane performanse tada mogu porasti čak do dva (2) puta. Slika 74 na sljedećoj stranici prikazuje nam rezultate mjerenja mrežnih performansi, generiranih/mjerenih pomoću naredbe `netcat` s obzirom na korištenje NUMA I/O i CPU afiniteta u svezi s njim.
Mrežne kartice korištene u mjerenjima su 40 Gbps, spojene na *PCI express* utor koji se nalazi na *NUMA Node0*, na kojem se nalazi i RAID kontroler. I mrežna kartica i RAID kontroler su natjerani da koriste signal prekida (**IRQ**) na istom *NUMA Node 0*. Razlika je sâmo u tome je li se naredba `netcat` pokretala na *NUMA Node 0* ili *NUMA Node 1* CPU jezgrama.

Na slici 74 (dolje) odnosno kao rezultat mjerenja, vidljivo je kako su ubrzanja, kod optimizacije NUMA I/O pristupa, u prosjeku 1,6 puta veća u odnosu na ona bez optimizacije. Dakle kada se naredba `netcat` pokretala na CPU jezgrama koje pripadaju istom fizičkom procesoru (*NUMA nodeu*), na koji je spojena sabirnica na koju su spojeni i mrežna kartica i disk kontroler (i naravno i sâm disk), tada su performanse rasle.

**Slika 74. Rezultati mjerenja.**




**Slika je preuzeta iz dokumenta tvrtke *Dell*:** *NUMA Best Practices for Dell PowerEdge 12th Generation Servers.* ***1**

S druge strane u slučaju kada se naredba `netcat` pokretala na CPU jezgrama koje se nalaze na drugom fizičkom procesoru i njegovim jezgrama, a koji je morao komunicirati preko među procesorske sabirnice kako bi tek preko prvog procesora mogao doći do njegove sabirnice i na kraju disk kontrolera te mrežne kartice na njemu, tada bi performanse očekivano bile manje. Očekivano manje performanse su zbog trošenja resursa u komunikaciji kroz sabirnicu između fizičkih procesora te korištenja udaljene sabirnice (od drugog fizičkog procesora).

**Kako saznati koji uređaj, na kojoj PCI sabirnici pripada kojem NUMA *node-u* odnosno fizičkom procesoru?**
Prvo pogledajmo sve PCI uređaje te ih izlistajmo, a pri tome ćemo prikazati skraćeni ispis, zbog lakšeg razumijevanja:
Za ovu potrebu ćemo koristiti naredbu `lspci` (ispis smo skratili).

```
lspci

00:00.0 Host bridge: Intel Corporation Xeon E5/Core i7 DMI2 (rev 07)
00:01.0 PCI bridge: Intel Corporation Xeon E5/Core i7 IIO PCI Express Root Port 1a (rev 07)
01:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5720 Gigabit Ethernet PCIe
00:04.0 System peripheral: Intel Corporation Xeon E5/Core i7 DMA Channel 0 (rev 07)
00:1f.2 SATA controller: Intel Corporation C600/X79 series chipset 6-Port SATA AHCI Controller
03:00.0 RAID bus controller: LSI Logic / Symbios Logic MegaRAID SAS 2208 [Thunderbolt] (rev 01)
3f:08.0 System peripheral: Intel Corporation Xeon E5/Core i7 QPI Link 0 (rev 07)
3f:09.0 System peripheral: Intel Corporation Xeon E5/Core i7 QPI Link 1 (rev 07)
40:01.0 PCI bridge: Intel Corporation Xeon E5/Core i7 IIO PCI Express Root Port 1a (rev 07)
40:04.2 System peripheral: Intel Corporation Xeon E5/Core i7 DMA Channel 2 (rev 07)
40:05.0 System peripheral: Intel Corporation Xeon E5/Core i7 Address Map, VTd_Misc, System…
7f:08.0 System peripheral: Intel Corporation Xeon E5/Core i7 QPI Link 0 (rev 07)
7f:09.0 System peripheral: Intel Corporation Xeon E5/Core i7 QPI Link 1 (rev 07)
7f:0b.0 System peripheral: Intel Corporation Xeon E5/Core i7 Interrupt Control Registers
```

U grubo je vidljivo kako su na ovom poslužitelju (***Dell PowerEdge R730***) vidljiva četiri nîza *PCI Express* sabirnica.

Prema numeričkim oznakama su to:

- **0:**yy.z → Sve ***PCI*** oznake koje počinju s **0.**
- **3f:**yy.z → Sve ***PCI*** oznake koje počinju s **3f.**
- **40:**yy.z → Sve ***PCI*** oznake koje počinju s **40.**
- **7f:**yy.z → Sve ***PCI*** oznake koje počinju sa **7f.**

Prvu oznaku nazovimo **X**.

Na svakom od tih *PCI Express* sabirnica se nalaze *PCI express* uređaji. Neki su ugrađeni na matičnu ploču (veći dio njih), a neki su spojeni na *PCI Express* utore kao zasebne kartice. Oznake koje su pod **x**, **yy** ili **z** označavaju identifikator određenog uređaja na sabirnici odnosno sabirnicama, pošto ih svaki NUMA *node* ima više.
Unutar direktorija `/sys/devices/pci0000:`**X** se nalaze informacije o tim uređajima. Pogledajmo i kako to izgleda:

```
ls -al /sys/devices/ | grep pci0000

drwxr-xr-x 31 root root 0 May 18 12:02 pci0000:00
drwxr-xr-x 46 root root 0 May 30 15:01 pci0000:3f
drwxr-xr-x 18 root root 0 May 30 15:01 pci0000:40
drwxr-xr-x 46 root root 0 May 30 15:01 pci0000:7f
```

Unutar svakog od tih direktorija se nalazi identifikator točnog uređaja.

Ako pri tome iz naredbe `lspci` izvučemo samo identifikator za prvu mrežnu karticu:

```
01:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5720 Gigabit Ethernet
PCIe
```

Dobivamo: `01:00.0` što znači kako je naš vršni direktorij `/sys/devices/pci0000:00` unutar kojeg mora postojati direktorij (mapa) za naš uređaj `01.0`.

Pazite na to kako se znak `:` u nazivu direktorija, kod izlistavanja, mora posebno označiti (*escape-ati)* sa znakom `\` i to ovako:

```
ls -al  /sys/devices/pci0000\:00/
```

I dobivamo sljedeći ispis:

```
drwxr-xr-x  3 root root    0 May 30 15:01 0000:00:00.0
drwxr-xr-x  8 root root    0 May 18 12:02 0000:00:01.0
drwxr-xr-x  8 root root    0 May 18 12:02 0000:00:01.1
drwxr-xr-x  6 root root    0 May 30 15:01 0000:00:02.0
drwxr-xr-x  7 root root    0 May 18 12:02 0000:00:02.2
drwxr-xr-x  6 root root    0 May 30 15:01 0000:00:03.0
```

U ovom skraćenom izlistanju imamo naš direktorij `0000:00:01.0`.

Uđimo u njega i provjerimo kojem *NUMA node-u* pripada:

```
cd  /sys/devices/pci0000\:00/0000\:00\:01.0/
```

I potom ispišimo sadržaj sljedeće datoteke (`numa_node`):

```
cat numa_node

0
```

U našem konkretnom slučaju vidimo kako se radi o *NUMA node-u*: `0`

Sadržaj datoteke `numa_node` nam daje informaciju na koji ***NUMA NODE*** je spojen naš *PCI (Express*) uređaj, u ovom slučaju naša mrežna kartica. Isto vrijedi i za sve druge ***PCI*** identifikatore odnosno uređaje.

**Ovo smo mogli doznati i na još jednostavniji način**
Primjerice za mrežnu karticu `eth0`, pogledajmo(***1**), odnosno uđimo u sljedeći direktorij:

```
cd /sys/class/net/eth0/device/
```

Te pogledajmo sadržaj sljedeće datoteke:

```
cat /sys/class/net/eth0/device/numa_node

0
```

Dobili smo informaciju kako je za mrežnu karticu `eth0` zadužen prvi *NUMA* fizički procesor (*NUMA 0*).
Osim ove informacije, pohranjena je i lista NUMA jezgri na navedenom *node-u*, koje ovaj uređaj smije koristiti.

Informacija o listi jezgri se nalazi u datoteci (unutar istog direktorija u koji smo ušli*1): `local_cpulist`. Pogledajmo i to:

```
cat local_cpulist

0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
```

Ovdje se nalazi definiran i signal prekida (***IRQ***) koji ovaj konkretan uređaj koristi (datoteka `irq`) u istom direktoriju od gore:

```
cat irq

95
```

Dakle ovaj uređaj koristi hardverski signal prekida (**IRQ**) broj **95**.

Napomena vezana za ove hardverske signale prekida: svi noviji uređaji koriste ***MSI*** signale prekida ("`IR-PCI-MSI-edge`" stupac u ***IRQ*** tablici) koji se povezuju na malo drugačiji način (i poput ove integrirane mrežne kartice koju smo promatrali).

Više detalja u tome koju vrstu ***IRQ*** neki uređaj koristi možemo vidjeti s naredbom `lspci -v -s` nakon koje slijedi identifikator PCI uređaja, poput ovoga koji smo gledali. Ovdje ćemo dodatno vidjeti i *NUMA* pripadnost.

Sada pogledajmo našu PCI sabirnicu odnosno ovaj uređaj (`01:00.0`) na njoj (ispis smo skratili):

```
lspci -v -s 01:00.0

01:00.0 Ethernet controller:Broadcom Corporation NetXtreme BCM5720 Gigabit Ethernet PCIe
        Subsystem: Dell Device 1f5b
        Flags: bus master, fast devsel, latency 0, IRQ 95, NUMA node 0 ←
        Expansion ROM at dc800000 [disabled] [size=256K]
        Capabilities: [58] MSI: Enable- Count=1/8 Maskable- 64bit+
        Capabilities: [a0] MSI-X: Enable+ Count=17 Masked-
        Capabilities: [ac] Express Endpoint, MSI 00
        Capabilities: [13c] Device Serial Number 00-00-e0-db-55-0f-98-70
        Capabilities: [230] Transaction Processing Hints
        Kernel driver in use: tg3
```

Ovdje u ispisu vidimo i ***NUMA*** pripadnost konkretne mrežne kartice (`NUMA node 0`).
Ako vas detaljnije zanimaju ***MSI*** signali prekida, pogledajte članak: http://lwn.net/Articles/44139/



Pogledajte i poglavlja:
**10.1.1. Matična ploča, CPU, *chipset* i sabirnica(e).** → *napredni dio ovog poglavlja.*
**10.5.1. IRQ (*Interrupt request*).**

I na kraju, optimizacija za NUMA I/O se svodi na optimizaciju nekoliko komponenti:

- Na najnižoj razini, provjeriti koji ***IRQ*** koristi uređaj čiji NUMA I/O optimiziramo. Trebao bi biti na istom *NUMA NODE-u*, odnosno CPU jezgri (ili jezgrama) unutar tog istog *NUMA NODEa*.
- Proces/program/aplikacija mora također biti na istom *NUMA NODE-u* kao i sam uređaj i njegov ***IRQ***.

**Izvori informacija: (K-9), *1, `man netcat`, `man lspci`, `man 5 sysfs`.**

#### 10.7.2.2.2. NUMA CPU i RAM afinitet

Slično kao i kod ***SMP*** sustava, na ***NUMA*** sustavima je također moguće postaviti *CPU* afinitet. Ali ovdje se radi o CPU afinitetu koji je svjestan *NUMA* arhitekture odnosno pripadnosti RAM memorije određenom memorijskom kontroleru unutar konkretnog fizičkog procesora. Sâm *NUMA* RAM afinitet se definira kroz razne parametre, a o njima u primjerima koji slijede. Kada bi za ovu namjenu koristili ***SMP*** alate, poput naredbe: `taskset`, nikada ne bi bili sigurni na kojem *NUMA node-u* (fizičkom procesoru) bi određeni proces ili njegova programska nît bila pokrenuta, a s kojeg *NUMA node-a* bi koristila dostupnu mu RAM memoriju. Stoga nam treba alat za CPU afinitet koji je svjestan NUMA arhitekture. Program koji nam daje ove mogućnosti zove se `numactl`, a koristi se slično kao i naredba `taskset`. On dolazi u softverskom (**RPM**) paketu imena: `numactl`. Slijede primjeri. Postavimo *NUMA* CPU afinitet za cijeli fizički procesor (*NUMA node*). U ovom primjeru našem procesu (programu) ćemo dozvoliti kako bi se mogao pokretati na bilo kojoj CPU jezgri, ali unutar jednog NUMA *node-a*. S ovime smo osigurali pristup bilo kojoj CPU jezgri na jednom *NUMA node-u*, pa smo sigurni kako će dohvaćanje RAM memorije biti uvijek lokalno tj. upotrebom memorijskog kontrolera ovog fizičkog procesora, bez potrebe za dohvaćanjem udaljene RAM memorije, a ovu opciju smo osigurali prekidačem: `--localalloc`. U primjeru koji slijedi, naš proces (program) ćemo vezati za *NUMA NODE 0* te će moći koristiti samo RAM memoriju od tog fizičkog procesora. Za to možemo napraviti sljedeće:

```
numactl --localalloc --cpunodebind=0 IME_PROGRAMA
```

Opis gore navedene naredbe koju smo pokrenuli:

- `--localalloc` - definira alociranje samo NUMA lokalne RAM memorije odnosno one koja pripada definiranom fizičkom procesoru (*NUMA NODEu*).
- `--cpunodebind=` - definira na kojem *NUMA node*-u se program smije izvršavati a to znači kako može koristiti sve CPU jezgre na tom fizičkom procesoru (*NUMA NODEu*).

Pogledajmo primjer kako pokrenuti aplikaciju za testiranje brzine mreže (`iperf -s`) koja će smjeti/moći koristiti samo CPU jezgre 0-7, koje se na našem poslužitelju nalaze na NUMA *node* 0.

```
numactl --physcpubind=0-7 iperf -s
```

**Što je još moguće definirati?**

- `--interleave=all` - s ovom opcijom uključujemo "*Memory policy controll*" mehanizam koji za razliku od `--localalloc` pokušava koristiti lokalnu RAM memoriju, sve dok je moguće, a tek tada počinje koristiti i udaljenu RAM memoriju, koja pripada udaljenom *NUMA node-u* i to korištenjem "*Round robin*" algoritma.
  Ovo je korisno, ako znamo da naša aplikacija treba više RAM memorije nego je dostupno pojedinom fizičkom procesoru (*NUMA NODE-u*). Osim vrijednosti `all` moguće je definirati i brojčanu oznaku *NUMA NODE-ova*.
  Drugi primjer upotrebe bi bio kod optimizacije I/O operacija na sustavima s četiri ili više *NUMA NODE-ova* (**4P** ili **8P** sustavi) kada točno znamo koji I/O uređaj: disk kontroler, mrežna kartica ili slično, fizički pripada kojem *NUMA NODE-u*. I ovdje vrijednosti mogu biti:
  - Pojedini *NUMA NODE ID*: **0** ili **1** ili **2** ili …
  - Ili nîz *NUMA NODE ID*-ova: **0**,**1**,**2**,**3** ili **0**,**2** i slično.
- `--membind=Z` - s ovom opcijom je moguće definirati s kojeg *NUMA node-a* će se koristiti RAM memorija. `Z` može biti i broj *NUMA node-a.*
- `--physcpubind=Y` - s ovom opcijom možemo raditi finiju granulaciju, na kojim CPU jezgrama unutar NUMA *NODE-a* (**X**), a koji smo definirali sa `--cpunodebind=X` se određeni program može pokretati. `Y` može biti CPU jezgra; poput: `1` ili nîz jezgri poput: `1,3,5,7` (odvojenih zarezom).



Postoje i **POSIX** varijable koje definiraju broj dostupnih CPU jezgri. Stoga pogledajte i poglavlje:
**6.2.2.1. Posebne sistemske varijable.**
Pogledajte i poglavlje vezano za usporedni test rada aplikacija koje se nalaze ili ne nalaze na istim ***NUMA NODE***-ovima
**25.2.1. Receive Side Scaling (*RSS*)** i to nekih pet stranica dalje, pod: **Test A1** i **Test B1**. Usporedite oba testa.
Pogledajte i kako na razini cijelog sustava izolirati određene jezgre CPUa:
**10.7.2.4. Izolacija jezgri procesora.**

Rezultati ovog testa su vidljivi u tablici:

| Test | Lokacija **klijenta** | Lokacija **poslužitelja** | Rezultat mjerenja brzine međusobne komunikacije: **klijent - poslužitelj** |
|---|---|---|---|
| Test **A1** | NODE1 (drugi fizički CPU) | **NODE0** (prvi fizički CPU) | **19.3 Gbps** |
| Test **B1** | **NODE0** (prvi fizički CPU) | **NODE0** (prvi fizički CPU) | **31.6 Gbps** |

Mjerenja pokazuju kako i optimirane visoko performantne aplikacije, ako se ne izvode na istim *NUMA NODE*-ovima odnosno izvode se na jezgrama procesora koje se nalaze unutar **različitih** fizičkih procesora, u međusobnoj komunikaciji, unose znatna usporavanja. Sve u odnosu na kombinaciju kada se programi (aplikacije) u međusobnoj komunikaciji nalaze (izvršavaju) na istim *NUMA NODE*-ovima odnosno jezgrama procesora koje se nalaze unutar **istih** fizičkih procesora. Dodatno, aplikacije koje znaju koristiti veći broj CPU jezgri kao i aplikacije koje opslužuju na tisuće, desetke tisuće ili više konekcija, još više su podložne ovom problemu, a naročito, ako nisu svjesne *NUMA* arhitekture, što ovisi o programskom jeziku u kojem su napisane te njegovim mogućnostima optimalnog rada na *NUMA* arhitekturi.

**Izvor informacija: (K-4), `man numactl`, `man iperf`.**

#### 10.7.2.2.3. NUMA Affinity Management Daemon (*numad*)

`Numad` je Linux servis koji automatski prati NUMA topologiju i korištenje njenih resursa. U radu `numad` može pomoći pri dinamičkoj raspodjeli resursa tako da prati iskorištavanje CPU jezgri za svaki pokrenuti proces. S obzirom na to da je on svjestan *NUMA* arhitekture on pokušava rasporediti procese prema CPU jezgrama, kako bismo uvijek bili u mogućnosti pristupiti podacima iz lokalne memorije za određeni procesor te tako smanjiti latenciju (kašnjenje).

Dakle on se brine o tome kako bi svaki pokrenuti proces bio dodijeljen *NUMA CPU* jezgri unutar jednog *NUMA node-a*.
Sve što je potrebno je pokrenuti ovaj servis. Za ***CentOS 6.x*** je potrebno napraviti sljedeće:

```
service numad start
```

A ako se nakon nekog vremena njegove upotrebe odlučimo da se on automatski pokreće s pokretanjem sustava, dovoljno je napraviti sljedeće: `chkconfig numad on`. Odnosno za ***CentOS 7.x/8.x*** koji koristi ***systemd*** je potrebno napraviti sljedeće:

```
systemctl start numad
```

A ako se nakon nekog vremena njegove upotrebe na ***CentOS 7.x/8.x*** odlučimo kako bi bilo dobra da se on automatski pokreće s pokretanjem sustava, dovoljno je napraviti sljedeće:

```
systemctl enable numad
```

**Napomena**: **Ako je *KSM* u upotrebi na NUMA sustavu, promjenite vrijednost parametra:** `/sys/kernel/mm/KSM/merge_nodes` na `0` kako bi se izbjeglo spajanje stranica (misli se na *Memory Pages*) na *NUMA NODE-ovima*.



`Naziv` **KSM** servisa na Linuxu je **`ksmtuned`**.

Inače, ***KSM*** tada može povećati kašnjenje odnosno latenciju jer će koristi pristup i na udaljenoj *NUMA* memoriji, koja se nalazi na drugom fizičkom procesoru i na lokalnoj memoriji istovremeno, tako da koristi memorijske stranice s lokalno dostupne RAM memorije i s NUMA udaljene memorije, naravno unutar adresnog prostora istog procesa.

Osim toga, statistike o korištenju memorijskih stranica (Engl *Memory pages*) koje prati kernel, mogu postati kontradiktorne, nakon velikog broja spajanja stranica u memoriji, a koje se nalaze između *NUMA NODE*-ova. Stoga kombinacija `numad` i ***KSM*** odnosno `ksmtuned` servisa zajedno, nije baš najbolje rješenje.



*U slučajevima korištenja Linux virtualizacije pomoću **KVM/QEMU**, najbolji rezultati se postižu bez `numad` jer i KVM i QEMU imaju mogućnost i opcije za prepoznavanje i rad s NUMA arhitekturom!*

Sumirajmo sve zadatke *NUMAD* servisa:

- **Optimizacija dodjeljivanja CPU-a i memorije** – osigurava da se procesi i programske nîti pridruže odgovarajućim CPU jezgrama i memorijskim stranicama unutar istog *NUMA* čvora. Time se smanjuje latencija pristupa memoriji jer procesi (programi) tada koriste RAM memoriju koja je fizički bliža procesoru na kojem se izvršava.
- **Balansiranje opterećenja između NUMA čvorova** – kada je jedan *NUMA* čvor preopterećen, servis može redistribuirati procese kako bi se ravnomjerno rasporedilo opterećenje među čvorovima, vodeći računa o minimalnoj upotrebi udaljene memorije.
- **Praćenje i upravljanje resursima** – servis prati upotrebu CPU-a, memorije i I/O operacija u realnom vremenu na svakom NUMA čvoru. On i prepoznaje procese koji narušavaju *NUMA* optimizaciju i poduzima odgovarajuće radnje, poput migracije procesa ili prilagođavanja CPU sklonosti (*afiniteta*).
- **Podrška za specifične NUMA politike** – implementira i provodi politike za određene aplikacije koje imaju specifične NUMA zahtjeve, npr. aplikacije koje preferiraju fiksnu upotrebu CPU jezgri i memorije.
- **Migracija procesa** – kada se promijeni stanje sustava (npr. povećanje opterećenja), servis može migrirati procese na različite NUMA čvorove radi boljih performansi. Migracija može uključivati premještanje i CPU niti i pripadajuće memorije.
- **Upravljanje *HugePages* resursima** – ako se koristi *HugePages*, servis pomaže u optimizaciji njegove alokacije između NUMA čvorova, osiguravajući da procesi koriste *HugePages* od odgovarajućeg *NUMA* čvora.
- **Interakcija sa upravljačkim slojem sustava** – servis može komunicirati sa sistemskim alatima poput `numactl`, `taskset` i kontrolnim grupama (`cgroups`) kako bi proveo svoje odluke i optimizacije.

**Izvori informacija: `man numad`,`man ksmtuned`,`man systemctl`.**

### 10.7.2.3. Praćenje procesa i opterećenja CPU jezgri

S obzirom na činjenicu, kako je potrebno pratiti stanje sustava s vremena na vrijeme, a ponekad i konstantno, važno je upoznati se s alatima koji su nam dostupni na Linuxu za tu namjenu. Svi alati koje ćemo spomenuti, mogu se koristiti i na *SMP* i *NUMA* sustavima. Većinu naredbi koje ćemo spomenuti, koristit ćemo na napredniji način.

---

#### 10.7.2.3.1. Naredba *ps*

Naredba `ps` se koristi za ispis podataka o pokrenutim programima odnosno takozvanim *procesima*.
Ovaj primjer smo već vidjeli za *SMP* sustav, ali ćemo ga ipak ponoviti. Prvo moramo naći ***PID*** broj našeg pokrenutog procesa/programa. U našem primjeru radi se o programu imena `kvm` koji pokreće jedno virtualno računalo na našem poslužitelju, ali se može raditi o bilo kojem programu za koji znamo da koristi više niti u radu. Pronađimo taj ***PID*** broj:

```
pidof kvm

4071
```

Pronašli smo da je ***PID*** (*Process ID*): `4071`. Za ovu potrebu, koristiti ćemo naredbu `ps`, ali sa dodatnim parametrima s kojim ćemo moći vidjeti sve što nas zanima o našem pokrenutom procesu (programu):

```
ps -mo pid,tid,tgid,fname,user,psr -p 4071

   PID     TID  TGID  COMMAND   USER      PSR
  4071       -  4071  kvm       root        -
     -    4071     -  -         root        5
     -    4096     -  -         root        3
     -    4097     -  -         root        5
     -    4098     -  -         root        3
```

Vidimo kako naš proces (***PID*** `4071`) ima više programskih nîti; pogledajte *Thread ID* (`TID`): `4071`, `4096`, `4097` i `4098` koje su trenutno aktivne; možda ih ima i više, ali trenutno nisu aktivne, te kako se neke od njih pokreću na CPU jezgri **5** a neke na CPU jezgri **3**. Svako pokretanje gore navedene naredbe očitava trenutno stanje, u djeliću sekunde kada smo naredbu pokrenuli.

Opis prekidača naredbe `ps` koje smo koristili slijedi:

- `-mo` - ispiši nîti (*Thread*) poslije procesa (`m`), a nakon toga slijede dodatne opcije (`o`)
  - `pid` - ispiši ***PID*** (*Process ID*) trenutnog (našeg) procesa.
  - `tid` - ispiši ***TID*** (*Thread ID*) (ID od svake nîti) trenutnog (našeg) procesa.
  - `tgid` - ispiše ***TGID*** (*Thread Group ID*) broj koji koristi ***PID*** broj procesa, ako se radi o njegovoj procesnoj nîti. Ako se radi o račvanju programa (engl. *Fork*), tada nova instanca dobiva drugi ***PID*** i ***TGID***.
  - `fname` - ispiši ime pokrenutog procesa/programa (vidljivo u stupcu ispod: `COMMAND`).
  - `user` - ispiši ime korisnika koji je pokrenuo ovaj (naš) proces/program.
  - `psr` - ispiši CPU jezgru na kojoj se trenutno izvršava programska nît.

  **Ovdje su dostupne i mnogi drugi korisni prekidači, poput:**
  - `cgroup` – ispisuje pripadnost ***cgroups*** mehanizmu za proces.

***Ako imate potrebu, možete poredati (sortirati) pokrenute programe (procese), primjerice prema iskorištenju CPU-a, ovako:***

```
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu
```

Za osnovni napredni prikaz odnosno izlistanje svih procesa, dovoljno je naredbu `ps` pokrenuti ovako:

```
ps -ef
```

Za ispis svih procesa koje je pokrenuo korisnik *root*, možemo koristiti:

```
ps -U root -u root u
```

Za ispis stabla pokrenutih procesa, možemo koristiti:

```
ps -e --forest
```

Za ispis procesa (primjerice po imenu procesa **httpd**) i njegovih programskih nîti, možemo koristiti:

```
ps -fL -C httpd
```

Za ispis sigurnosnih konteksta (***SELinux***) procesa možemo koristiti:

```
ps -eM
```

**Izvor informacija: (K-14),** **`man ps`**,**`man pidof`**.

---

#### 10.7.2.3.2. Naredba *top*

Naredba `top` daje nam statistike rada svih pokrenutih programa odnosno procesa na sustavu, kao i opterećenja cijelog sustava. Jedan dio opisa ove naredbe pokrili smo u poglavlju: **9. Proces menadžment.** Pokrenimo ju na sljedeći način:

```
top
```

Nakon pokretanja ove naredbe, vidjet ćemo slijedeće:

```
top - 18:55:42 up  4:55,  1 user,  load average: 0.00, 0.05, 0.07
Tasks: 122 total,   1 running,  77 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.5 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2032932 total,  1686696 free,   111512 used,   234724 buff/cache
KiB Swap:  2085884 total,  2085884 free,        0 used.  1758916 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 9243 root      20   0   42856   3684   3036 R   0.7  0.2   0:00.05 top
 1435 www-data  20   0  159908   3288   1448 S   0.3  0.2   0:00.53 nginx
    1 root      20   0  204908   7096   5156 S   0.0  0.3   0:04.52 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_par_gp
    5 root      20   0       0      0      0 I   0.0  0.0   0:03.50 kworker/0:0-eve
    6 root      20   0       0      0      0 S   0.0  0.0   0:00.11 ksoftirqd/0
    7 root      20   0       0      0      0 I   0.0  0.0   0:01.30 rcu_sched
```

Prvo ćemo objasniti gornje retke ispisa:

- `top - 18:55:42 up  4:55,  1 user` označava trenutno vrijeme (`18:55:42`) te kako je sustav u radu (`up  4:55`) četiri sata i 55 minuta. Te kako imamo prijavljenog (*logiranog*) jednog korisnika (`1 user`).
- `load average: 0.00, 0.03, 0.04` označava prosječno opterećenje cijelog sustava: CPU + I/O (diskovni podsustav) koje se izračunava za svaku minutu (prvi broj) te svakih pet minuta (drugi broj) i svakih 15 minuta (treći broj u nîzu). Konkretno prate se procesi u **R** (*running*) i **D** (*waiting for disk I/O*) stanjima. Dakle ovo nije pokazatelj sâmo postotka opterećenja CPUa. Vrijednost jedan (**1**) koja bi označavala 100% opterećen sustav s jednom CPU jezgrom: i sâm CPU i prema diskovnom podsustavu (pr. čekanje na I/O operacije prema diskovnom sustavu). Međutim maksimalna vrijednost može biti daleko viša od jedan (**1**) jer ova vrijednost označava ukupan broj procesa koji čekaju na izvršavanje. To znači da, ako imamo vrijednost dva (**2**), da dva procesa čekaju na izvršavanje, pa to znači da je opterećenje sustava 200%. Što bi moglo značiti 2 CPU jezgre na 100% ili su manje opterećene CPU jezgre, ali uz veće I/O opterećenje. Kod sustava s velikim brojem CPU jezgri: primjerice 32; vrijednost 32 praktično znači kako 32 procesa čekaju na izvršavanje, što bi značilo po jedan na svakoj CPU jezgri, što može značiti puno, ali može značiti i da je veliko čekanje na I/O sustav. Naime iz ove statistike ne znamo je li veliko opterećenje zbog CPU resursa ili čekanja na diskovni (I/O) podsustav. U svakom slučaju, manja vrijednost je dobrodošla. Napomena: sustavi s više CPU jezgri podnose veće vrijednosti (pr. vrijednost **5** za poslužitelj koji ima 32 jezgre nije velika).
  Ova statistika (*load average*) se konstantno izrađuje od strane linuxa i zapisuje u datoteku `/proc/loadavg` koja sadrži još dvije vrijednosti: četvrta po redu (pr. **1/171**) govori nam koji je trenutni broj kernel procesa najnižeg prioriteta (u primjeru je to **1**) na obradi te koliki je ukupan broj pokrenutih procesa na sustavu (**171**). Dok nam zadnji broj (peti) daje ***PID*** broj trenutno aktivnog procesa. Prve tri vrijednosti iz ove datoteke (dakle *load average*) čita i naredba `uptime` kao mnoge druge, uključujući naredbu `top`. Naredbom `vmstat` dobit ćemo detaljniji prikaz opterećenja CPUa (o tome u jednom od sljedećih poglavlja).
- `Tasks: 122 total,   1 running,  77 sleeping,   0 stopped,   0 zombie` znači kako ukupno imamo pokrenuto `122` programa (procesa), jedan je trenutno aktivan (`1 running`) u djeliću sekunde u kojem se dohvatila statistika. Potom vidimo da ih je `77` u stanju čekanja (*pauzirano*) (`77 sleeping`) te kako nema potpuno zaustavljenih procesa (`0 stopped`), a nemamo ni takozvanih „*zombi*" procesa (`0 zombie`) što je u redu.
- `%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.5 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st` označava ukupno opterećenje CPU resursa u postotku, kako slijedi:
  - %us - daje nam postotak koliko CPU vremena (resursa) koriste korisnički programi, u tzv. „*user-space*" dijelu memorije.
  - %sy – on nam daje postotak koliko CPU vremena (resursa) koriste kernel programi, u tzv. „kernel-*space*" dijelu memorije.
  - %ni – on nam prikazuje koliko CPU vremena (resursa) koriste korisnički programi kojima je mijenjan prioritet (*niceness*).
  - %wa -on nam govori koliko se CPU vremena (resursa) koristi za ulazno/izlazne (*I/O*) operacije, prema diskovnom sustavu.
  - %hi – on nam govori koliko se CPU vremena (resursa) koristi za obradu *hardverskih* signala prekida (*IRQ*).
  - %si - govori koliko se CPU vremena (resursa) koristi za obradu *softverskih* signala prekida (*soft IRQ*).
  - %st - govori koliko CPU resursa je preopterećeni hipervizor oduzeo virtualnom računalu (ako je ovo virtualno računalo).
- Potom vidimo iskorištenost memorije (`KiB Mem:`) i *swap* diska (`KiB Swap:`) te *buffer*/*cache* memorije (`buff/cache`). Vezano za *buffers*/*cache* pogledajte poglavlje: **12.3.3.1. Naredba *free***.
- Na kraju, u donjem dijelu slijede statistike za svaki pojedini program (proces) koje smo već objasnili u poglavlju: **9. Proces menadžment.**

Ispis odnosno prikaz naredbe `top` možemo mijenjati i u radu; objasnit ćemo samo par mogućnosti koje mijenjamo stiskanjem sljedećih tipki (slova).

Naime stiskanjem navedene tipke ili mijenjamo ili uključujemo/isključujemo navedenu opciju prikaza:

- Veliko slovo `E` – mijenjamo gornji dio prikaza zauzeća memorije (`Mem :`) ili *swap*-a (`Swap:`) u jedinicama: **KiB**, **MiB**, **GiB**, **TiB**, **PeB**, **EiB.**
- Malo slovo `e` – mijenjamo donji dio prikaza zauzeća memorije za pojedine procese (programe) u jedinicama: **KiB**, **MiB**, **GiB**, **TiB**, **PeB**, **EiB.**
- Veliko slovo `H` – mijenjamo gornji dio prikaza (drugi red od gore) pa možemo mijenjati prikaz da nam se prikazuje i ukupan broj programskih niti.
- Malo slovo `c` – desni dio prikaza programa/procesa prikazuje punu putanju pokrenutih procesa.
- Veliko slovo `I` – mijenjamo donji dio prikaza zauzeća CPU jezgri za pojedine procese (programe); i to u ***Solairs***/***Irix*** (zbroj svih CPU jezgri/100) ili u standardnom Linux stilu prikaza: 1 CPU jezgra=100%, dvije 200%, itd. (zbrajaju se).
- Malo slovo `m` – mijenjamo gornji dio prikaza zauzeća memorije (četvrti red od gore) cijelog sustava.
- Brojevi:
  - `1` – mijenjamo gornji dio prikaza zauzeća CPUa (treći red od gore) cijelog sustava: prikaz svih CPU jezgri (svake zasebno).
  - `2` – mijenjamo gornji dio prikaza zauzeća CPUa (treći red od gore) cijelog sustava: sumirani prikaz svih CPU jezgri po **NUMA** fizičkim procesorima (pr. *NUMA CPU0* i *NUMA CPU1* za 2 NUMA *node*-a).
- Veliko slovo `V` – mijenjamo donji dio prikaza procesa da se vidi stablo procesa (engl. *Forest*) odnosno koji proces je pokrenuo koji pod proces.
- Sortiranje po stupcima: `M` – po `%MEM`, `N` – po ***PID*** broju, `P` – po `%CPU` i `T` – po vremenu rada procesa (`TIME+`)

Naredba `top` osim standardnog rada, može nam prikazati stanje rada točno određenog procesa (prema ***PID***-u). Stoga ćemo naredbu `top` pokrenuti s prekidačem `-p` iza kojega mora slijediti ***PID*** broj procesa koji pratimo (u našem slučaju `4071`):

```
top -p 4071
```

Sada ćemo dobiti klasični ekran naredbe `top`, ali sâmo sa statistikama rada našeg traženog procesa odnosno programa:

```
top - 11:10:10 up 112 days, 12:19,  1 user,  load average: 0.08, 0.07, 0.05
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,  0.7 sy,  0.0 ni, 97.2 id,  1.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  98990200 total, 84836948 used, 14153252 free,  1342968 buffers
KiB Swap: 91488252 total,    39116 used, 91449136 free, 63898336 cached

    PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
   4071 root      20   0 8737m 7.8g 4060 S   7.7  8.3   2246:53 kvm
```

Ako želimo vidjeti detalje o ovom procesu, na kojoj CPU jezgri se trenutno izvršava, to možemo napraviti na sljedeći način:
**1.** Pritisnimo tipku (slovo) `f`
**2.** Odaberite (*kursorskim* tipkama; strelice gore/dolje): `P = Last Used Cpu (SMP)`
**3.** Stisnite razmak (*Space*) kada ste došli na polje `P = Last Used Cpu (SMP)` i pojaviti će se znak `*` ispred odabranog polja.
**4.** Stisnite tipku **`q`** (malo slovo Q).

Sada ćemo dobiti novi prikaz, poput ovoga:

```
top - 11:13:33 up 112 days, 12:22,  1 user,  load average: 0.00, 0.03, 0.04
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.2 us,  0.5 sy,  0.0 ni, 97.1 id,  1.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  98990200 total, 84837136 used, 14153064 free,  1342968 buffers
KiB Swap: 91488252 total,    39116 used, 91449136 free, 63898568 cached

    PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND   P
   4071 root      20   0 8737m 7.8g 4060 S   7.3  8.3   2247:12 kvm       5
```

Potom pogledajte donji red, zadnji stupac `P` u kojemu je vidljiv broj trenutne CPU jezgre koja obrađuje naš proces (***PID*** `4071`).

Vidimo kako se na kraju (desno) pojavio stupac `P` ispod kojega se za naš proces prikazuje CPU jezgra koja ga trenutno obrađuje (pričekajte malo i vidjet ćete kako se mijenja). Iz programa `top` se Izlazi s tipkom `q`. Dodatno, moguće je pratiti i opterećenja pojedinih CPU jezgri, u stvarnom vremenu.

Izađimo iz programa `top` i pokrenimo ga ponovno:

```
top
```

Sada pritisnite broj `1` te će se prikaz promijeniti u prikaz opterećenja CPU jezgri:

```
top - 10:25:29 up 107 days, 23:08,  1 user,  load average: 0.06, 0.07, 0.01
Tasks: 557 total,   1 running, 556 sleeping,   0 stopped,   0 zombie

%Cpu0  :  9.7 us,  1.3 sy,  0.0 ni, 89.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  2.3 us,  1.0 sy,  0.0 ni, 96.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 10.7 us,  2.7 sy,  0.0 ni, 85.3 id,  1.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  1.7 us,  1.3 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.7 us,  0.7 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  1.3 us,  0.3 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  2.8 us,  0.3 sy,  0.0 ni, 96.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

Ispis je skraćen zbog potrebe prikaza, ali u donjem dijelu slike je vidljivo opterećenje prema CPU jezgrama: `%Cpu0 - %Cpu7`. Ovdje se radi o računalu s osam (8) CPU jezgri.



Za detalje oko naredbe `top` i korištenja RAM memorije odnosno sustava virtualne memorije pogledajte poglavlja: **12.3.3.4. Naredbe ps i top za memoriju** te poglavlje: **9. Proces menadžment.**

**Izvori informacija: (570),(571),(572), `man top`.**

---

#### 10.7.2.3.3. Naredba *htop*

Ova naredba ponekad nije instalirana na sustav. Ako nije instalirana, instalirajte ju na sljedeći način:

```
yum -y install htop
```

Ova naredba dolazi u ***EPEL*** repozitoriju, kako dodati ***EPEL*** repozitorij.



Pogledajte poglavlje:
**7.2.2.1. Rad s *YUM-om***

Naredba `htop` ima i nekolicinu korisnik prekidača prilikom pokretanja, a prilikom rada možemo koristiti i neke od funkcijskih tipki:

- `F1` – za ispis uputa.
- `F2` – za ulazak u „*setup*“ s postavkama.
- `F3` – za inkrementalnu pretragu naredbenog retka.
- `F4` – za inkrementalno pretraživanje procesa (programa).
- `F5` – za ispis procesa (programa) u obliku stabla [roditeljski proces – proces dijete].
- `F6` – za odabir polja za filtriranje.
- `F7` – za povećanje ***nice*** vrijednosti odabranog procesa (samo kao root korisnik).
- `F8` – za smanjenje ***nice*** vrijednosti odabranog procesa (samo kao root korisnik).
- `F9` – za ubijanje odabranog procesa.
- `F10` – za izlazak iz programa.

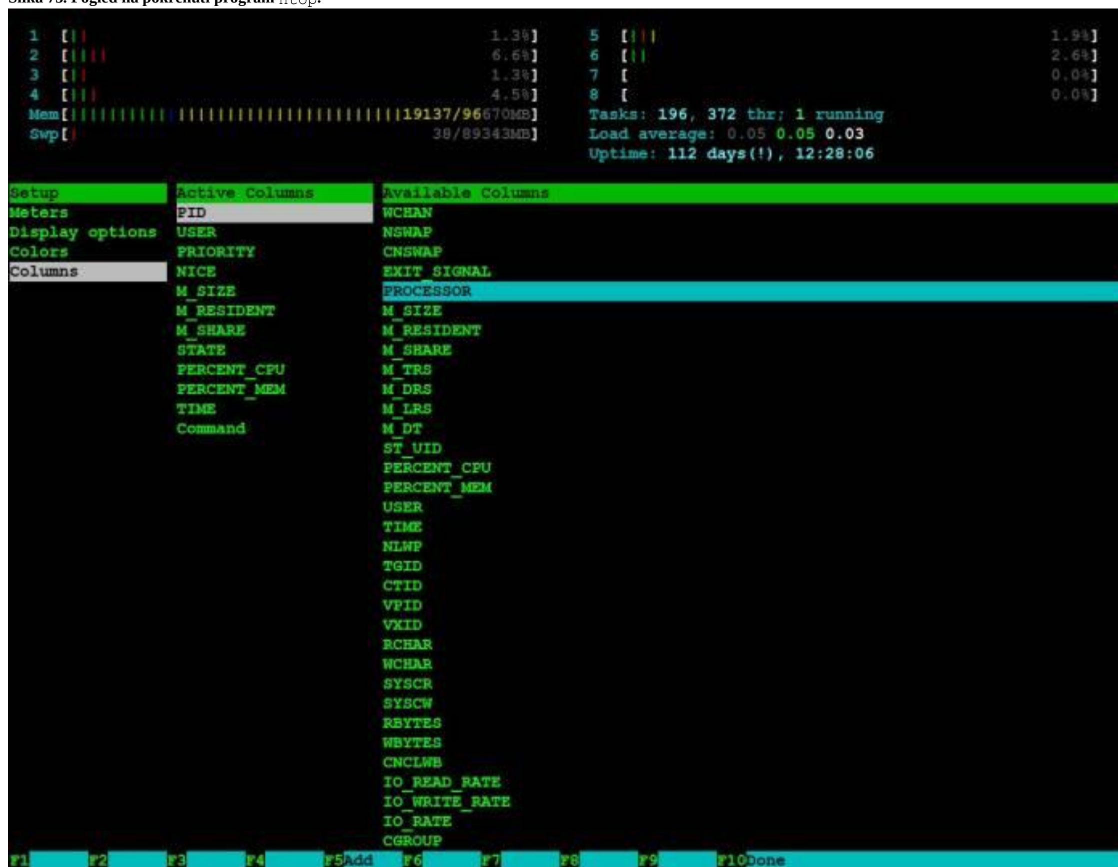
Pokrenimo naredbu `htop`:

```
htop
```

Ova naredba nam osim standardnog prikaza nudi i dodatne opcije prikaza, koje ćemo mi i koristiti.

Stoga stisnite `F2`, odaberite (s kursorskim tipkama: strelice gore/dolje):
**1.** Idite na `Setup`
**2.** Potom dolje na `Columns`
**3.** Idite desno na `Available Coumns` i idite dolje te označite polje `PROCESSOR` (pa stisnite tipku ENTER), kao na slici dolje.

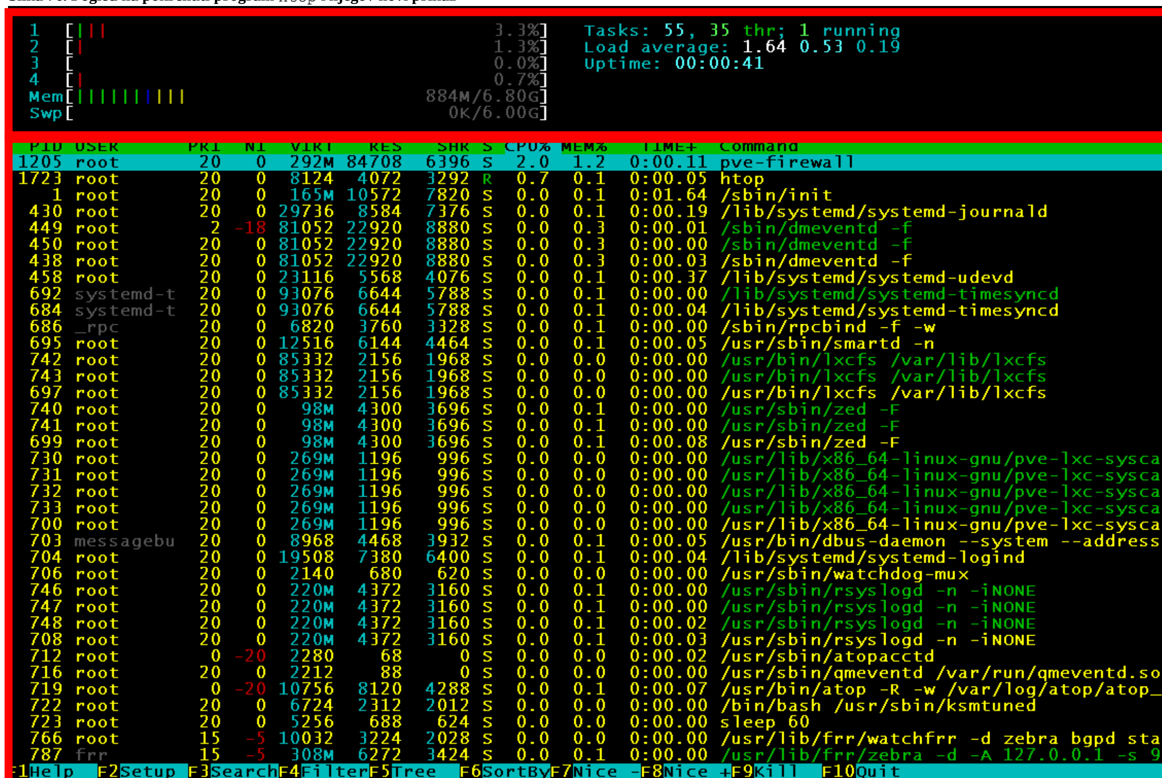
**Slika 75. Pogled na pokrenuti program** `htop`**.**




4. Sada stisnite tipku `ESC`

Odmah potom ćete vidjeti i statistike za sve procese (pokrenute programe), s tim da će se s lijeve strane u prvom stupcu vidjet i koja CPU jezgra se trenutno koristi za koji proces (stupac `CPU`) kao na slici dolje:

**Slika 76. Pogled na pokrenuti program** `htop` **i njegov novi prikaz**




U gornjem dijelu ekrana (1) je vidljiva i grafička reprezentacija upotrebe svake CPU jezgre kao i RAM memorije.
U donjem dijelu (2) su informacije o svakom pokrenutom procesu, slično kao kod naredbe `top`.

**Izvor informacija:** **`man htop`** **.**

### 10.7.2.3.4. Naredba *sar* (NUMA ili SMP)

**Slijedi napredna cjelina!**
Naredba `sar` dolazi nam u softverskom paketu `sysstat` koji je ponekad potrebno instalirati. Dakle ova naredba je jedna od komponenti *sysstat* podsustava. Vezano za njen rad, postoji i nekoliko njenih komponenti, koje su zadužene za sve njene funkcionalnosti:

- `sar` - ovo je naša naredba koja prikazuje podatke koje smo prikupili (takozvani *reporter* alat).
- `sadc` - ovo je sistemski servis koji pokreće naredba `sar` kada prikuplja podatke. Ovaj servis (`sadc`) zapravo prikuplja podatke za naredbu `sar`.
- `sa1` - ovo je interni mehanizam zadužen za spremanje podataka u željeno vrijeme, a koje prikuplja `sadc` naredba/servis, a zapisuje ih u binarnu datoteku. Ovo je obično skripta koja poziva naredbu `sadc` s određenim parametrima.
- `sa2` - je program koji zna dekodirati (prikazati) binarni zapis u datoteci, koju je snimio `sa1` mehanizam. Ovo je obično skripta koja poziva naredbu `sadc` s određenim parametrima te generira kompletan izvještaj.
- `sadf` - je program koji može prikazati statistike iz binarne datoteke, u drugim formatima (*CSV*, *XML* i sl.).

Naime kada pokrenemo naredbu `sar`, ona pokreće servis `sadc` koji kreće s prikupljanje statistika te ih zapisuje u datoteku unutar direktorija: `/var/log/sa/`. Ime datoteke je u formatu: sa**XX**, pri čemu je **XX** dan u mjesecu. Dakle, ako je dan u mjesecu 24, tada će se datoteka koja će se snimiti taj dan, zvati: `sa24`. Ovo je posebno formatirana binarna datoteka, koju program `sar` pomoću `sa2` mehanizma zna ispravno prikazati. Ako je potrebno moguće je kreirati i kompletni (dnevni) izvještaj, pomoću naredbe odnosno skripte: `sa2`, koji će biti snimljen u novu datoteku. Ova datoteka će se naći u direktoriju: `/var/log/sa/` imena: sar **XX**, pri čemu je **XX** dan u mjesecu. Uočite kako je ime datoteke *sar*, a ne *sa*. Glavna konfiguracijska datoteka sar-a se nalazi na lokaciji: `/etc/sysconfig/sysstat`. U njoj se mogu konfigurirati parametri poput arhiviranja svih statistika u direktoriju: `/var/log/sa/`, a standardno je to jednom mjesečno.

**Malo teorije i osnovna konfiguracija**
Prvo dobro proučite servis `crontab` u poglavlju: **7.5.1. *Crontab* servis**, jer se za snimanje analitike u željeno vrijeme, koristi *crontab* servis. Na svim sustavima koji koriste `sysstat` paket kreira se posebna *crontab* datoteka na razini cijelog sustava, i to datoteka imena: `/etc/cron.d/sysstat`.
Ona sadrži dva aktivna unosa:

```
*/10 * * * root /usr/lib/sa/sa1 1 1
53 23 * * * root /usr/lib/sa/sa2 -A
```

Pogledajmo opis unosa, jer se oni odnose na `sar` alat:
**1.** Prvi red odnosno unos: `*/10 * * * root /usr/lib/sa/sa1 1 1` – znači kako se svakih deset (10) minuta snimaju statistike sustava u standardnu datoteku: `/var/log/sa/sa`**XX** (**XX** je broj dana u mjesecu).

**2.** Drugi unos: `53 23 * * * root /usr/lib/sa/sa2 -A` – znači kako se jednom dnevno, u 23:53 h., kreira dnevna statistika iz kumulativnih statistika u standardnu datoteku: `/var/log/sa/sar`**XX** (**XX** je broj dana u mjesecu).

Dakle standardno se snimaju *sar* statistike rada cijelog sustava, svakih 10 minuta, te se kumulativno dopunjava dnevna datoteka sa statistikama: `/var/log/sa/sa`**XX** (**XX** je broj dâna u mjesecu). Nakon toga se, jednom dnevno u 23:53 h. izrađuje dnevna statistika, iz prethodne datoteke i zapisuje u novu statistiku, odnosno u novu datoteku imena: `/var/log/sa/sar`**XX** (**XX** je broj dâna u mjesecu).
Ono što se standardno od statistika snima je sljedeće:

- `%user` - opterećenje CPUa, od strane korisničkih programa.
- `%nice` - je opterećenje CPUa za korisničke programe koji imaju drugačiji (*NICE*) prioritet.
- `%system` - opterećenje CPUa, od strane sistemskih programa.
- `%iowait` - je opterećenje CPUa za I/O procese (prema diskovnom podsustavu), na koje se čeka i troši CPU vrijeme.
- `%steal` - je opterećenje CPUa kod virtualnih računala (ako se radi o virtualnom računalu), u slučajevima kada *Hipervizor* koji je preopterećen mora oduzeti dio CPU vremena (resursa) virtualnom računalu.
- `%idle` - je postotak koliko je CPU slobodan (ne radi ništa).

U svakom trenutku, možemo i ručno vidjeti zapisanu kumulativnu statistiku za trenutni dan, samo pozivanje naredbe:

```
sar
```

Ova naredba zapravo čita trenutnu dnevnu datoteku: `/var/log/sa/sa`**XX** (**XX** je broj dana u mjesecu), što možemo napraviti i ručno primjerice, ako je danas osmi (8) dan u mjesecu:

```
sar -f /var/log/sa/sa08
```

I dobit ćemo isti ispis.
Na isti način možemo gledati i kumulativne statistike za svaki dan unutar trenutnog mjeseca, koje se kako smo vidjeli, rade i zapisuju svakih deset (10) minuta. Skripte `sa1` i `sa2` ovisno radi li se o 32 bitnom ili 64 bitnom *Linuxu*, nalaze na različitim lokacijama, odnosno unutar direktorija (mapa):

- Za 32 bitni Linux je to: `/usr/lib/sa/`
- Za 64 bitni Linux je to: `/usr/lib64/sa/`

Ako želimo da se standardno svakih 10 minuta snimaju i druge statistike, potrebno je ručno urediti odnosno otvoriti skriptu: `sa1` i to dio u sljedećem retku (liniji):

```
exec ${ENDIR}/sadc -F -L
```

Dakle ovdje, uz prekidače: `-F` i `-L`, možemo dodati i druge, ali treba biti oprezan, kako s time ne bi usporili cijeli sustav.

Važno je znati i kako se nakon instalacije paketa `sysstat` on inicijalizira kao servis te se inicijalno pokreće sa sustavom. Doduše ne baš na način na koji smo navikli kod linux servisa. U svakom slučaj pogledajmo kako on radi.

**Za RedHat/CentOS 6.x.**
Provjerimo inicijalizira li se servis `sysstat` tijekom svakog pokretanja (i restarta) sustava:

```
chkconfig --list |grep sysstat
sysstat          0:off    1:on     2:on     3:on     4:on     5:on     6:off
```

Dakle `sysstat` se inicijalizira tijekom pokretanja sustava (*1:on 2:on 3:on 4:on 5:on*).
Ako smo ipak nešto poremetili te želimo da se ovaj servis ponovno uključi, to možemo napraviti na sljedeći način:

```
chkconfig sysstat on
```



Vezano za servise odnosno prema Unix/Linux terminologiji *daemone* odnosno rad s njima, pogledajte poglavlja:
**7.3.1 *Init* i sistemski servisi (*daemoni*)**
**7.3.2 *Systemd* i sistemski servisi (*daemoni*)**.

**Za RedHat/CentOS 7.x./8+**
Provjerimo inicijalizira li se servis (*daemon*) `sysstat` tijekom pokretanja sustava:

```
systemctl list-unit-files | grep enabled | grep sysstat

sysstat.service           enabled
```

Dakle `sysstat` se inicijalizira tijekom pokretanja sustava jer je u stanju `enabled`.

Ako smo nešto poremetili ili se on ne pokreće sa sustavom, možemo ga ponovno uključiti na sljedeći način:

```
systemctl enable sysstat.service
```

**Izrada trenutne analitike sustava**
Naredba `sar` nam nudi praćenje raznih statistika sustava, a mi ćemo se u ovoj cjelini fokusirati na dio u kojem možemo vidjeti opterećenja pojedinih CPU jezgri. Za ovu potrebu ćemo koristiti: `sar -P ALL`.
Dodatno ćemo pratiti statistiku svake sekunde (1) i to četiri (4) puta za redom.

```
sar -P ALL 1 4

12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal     %idle
12:10:01 AM     all      0.01      0.00      0.00      0.06      0.00     99.93
12:10:01 AM       0      0.00      0.00      0.00      0.12      0.00     99.87
12:10:01 AM       1     10.01      0.00      0.00      0.00      0.00     89.98
12:10:01 AM       2      0.00      0.00      0.00      0.12      0.00     99.87
12:10:01 AM       3      0.01      0.00      0.00      0.00      0.00     99.98
```

Skratili smo ispis, ali vidljivo je ukupno opterećenje sve četiri CPU jezgre: `0`-`3` (pod stupcem `CPU`) koje imamo na našem računalu, kao i opterećenje CPUa za korisničke programe (`%user`) te sistemske servise (`%system`).



Pogledajte i poglavlja:
**10.7.2.3.4. Naredba *sar* (NUMA ili SMP).**
**25.3. Statistike, analiza i praćenje mrežnih paketa.**



Za statistike `%iowait` i `%steal` pogledajte sljedeću naredbu (`vmstat`) odnosno opise u njoj.

**Izvor informacija:** `man sar`, `man systemctl`, `man 5 sysstat`.

#### 10.7.2.3.5. Naredba *vmstat* za CPU

**Slijedi napredna cjelina!**
Naredba `vmstat` se koristi za prikaz statistika sustava virtualne memorije te dolazi u paketu `sysstat`.
Međutim postoji i mala poveznica između sustava virtualne memorije, diskovnog podsustava i opterećenja *CPUa*.
Kako ovdje govorimo o *CPUu*, pogledati ćemo i što nam ova naredba od *CPU* statistika može prikupiti odnosno prikazati.



Za praćenje stanja diskovnog podsustava, pogledajte poglavlje:
**14.3 Praćenje performansi I/O sustava.**

Prvo ćemo pokrenuti ovu naredbu, i to tako kako bi se statistike prikupljale svake dvije sekunde (2) i to pet puta za redom (5). Preporuka je pratiti opterećenje ipak duže vrijeme (pr. `2 100`). Dodatno želimo ispis u MB (*MegaByte*), što ćemo dobiti korištenjem prekidača: `-S M`.

Moguće je koristiti i `-t` za prikaz vremena kada je statistika napravljena (to sada nećemo koristiti).

```
vmstat -S M 2 5

procs -----------memory---------- ---swap-- -----io---- -system--  ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs  us sy id wa
 0  1     10  13327   1939  61073    0    0    13    15    3    2   2  1 97  0
 1  1     10  13327   1939  61073    0    0     8    13 19423 37877  1  1 98  1
 0  1     10  13327   1939  61073    0    0     8    15 19201 37557  1  1 98  1
 0  1     10  13327   1939  61073    0    0     8     1 19250 37555  1  1 98  1
 1  1     10  13327   1939  61073    0    0     8     9 19836 38788  3  1 95  1
```

U ispisu ćemo se bazirati na stupcu `--cpu--` u kojem se prikazuju statistike vezane za opterećenje *CPUa*.

Ovdje imamo četiri pôd stupca koji označavaju sljedeće:

- `us` - procesorsko (*CPU*) vrijeme koje se troši na obradu aplikacija na razini korisnika (ne na kernel razini).
- `sy` - procesorsko (*CPU*) vrijeme koje se troši (*CPU* resursi) na obradu kernel zahtjeva i procesa na razini kernela, to su primjerice *kernel threadovi* odnosno posebni kernel servisi/programi.
- `id` - je takozvano *„idle"* vrijeme odnosno vrijeme kada je *CPU* neaktivan (kada/koliko ništa ne radi).
- `wa` - je takozvano *waiting for I/O* vrijeme, koje označava koliko *CPU* čeka na ulazne ili izlazne (I/O) operacije prema diskovnom sustavu. Visoke vrijednosti ovog polja indiciraju kako je diskovni sustav preopterećen zapisivanjem ili čitanjem te da sustav (*CPU*) mora čekati na operacije čitanja ili zapisivanja na disk. To će također biti vidljivo, ako je sva RAM memorija iskorištenja te se krenulo u korištenje *SWAP* memorije. Što je vidljivo pod statistikama `-swap-` i to za čitanje iz *SWAP* memorije (`si`) odnosno za zapisivanje na *SWAP* (`so`) memoriju.

Kod virtualnih računala (ako pokrećemo ovu naredbu na njima) se pojavljuje još jedan pôd stupac u *CPU* stupcu:

- `st` - ovo je vrijeme ukradeno od *CPUa* virtualnog računala. Dakle ukradeno od strane *Hypervizora* u slučajevima kada je *Hypervizor* preopterećen pa ovom (virtualnom) računalu sustav oduzima CPU resurse (pogledajte sljedeću naredbu).



Vezano za virtualizaciju pogledajte poglavlje:
**27.1. Virtualizacija.**

Objasnit ćemo i druga polja, koja nismo prethodno objasnili:

- `io` - ovo su statistike vezane za diskovni (I/O) sustav:
  - `bi` - označava broj blokova podataka zaprimljenih s blok (disk) uređaja.
  - `bo` - označava broj blokova podataka poslanih na blok (disk) uređaja.
- `System` - ovo su statistike sustava, vezane za:
  - `in` - označava trenutni broj *hardverskih signala prekida* (**IRQ**) u sekundi, uključujući „*clock*".
  - `cs` - označava trenutni broj *context switcheva* u sekundi.

**Izvor informacija: `man vmstat`.**

#### 10.7.2.3.6. Naredba *mpstat* za CPU

**Slijedi napredna cjelina!**
Naredba `mpstat` koristi se za statistike opterećenja *CPUa*, a dolazi u paketu *sysstat*.
Pokrenimo ju pa ćemo objasniti što sve prikazuje. Pokrenimo ju ovako:

```
mpstat

Linux 2.6.32-48-pve (Server-1)    02/01/2018      _x86_64_        (8 CPU)
01:35:29 PM  CPU  %usr  %nice   %sys  %iowait   %irq  %soft  %steal  %guest   %idle
01:35:29 PM  all  1.70  0.00    0.66     0.06   0.00   0.03    0.00    0.21   97.34
```

Ako ovu naredbu pokrenemo bez prekidača, dobivamo sumariziranu statistiku za sve CPU jezgre, pa je vidljivo sljedeće:

- `CPU` - prikazuje se `all` jer je statistika za sve CPU jezgre.
- `%usr` - je opterećenje CPUa za procese korisnika (*user space*).
- `%nice` - je opterećenje CPUa za korisničke programe koji imaju drugačiji (*NICE*) prioritet.
- `%sys` - je opterećenje CPUa za sistemske servise (*daemone*).
- `%iowait`- je opterećenje CPUa za I/O procese (prema diskovnom podsustavu), na koje se čeka i troši CPU vrijeme.
- `%irq` - je opterećenje CPUa za opsluživanje hardverskih signala prekida (**IRQ** / *interrupta*).
- `%soft` - je opterećenje CPUa za opsluživanje softverskih signala prekida (**Soft IRQ** / *interrupta*).
- Za virtualna računala odnosno za njihove **Hipervizore** imamo dodatne statistike:
    - `%steal` - prikazuje opterećenje CPUa kod virtualnih računala u slučajevima kada ***Hipervizor*** oduzima dio CPU vremena virtualnom računalu jer je hardverski preopterećen. Dakle ova statistika unutar virtualnog računala nam govori koliko CPU vremena nam je *ukrao hipervizor* za virtualizaciju jer je preopterećen (CPU), pa je morao (ovom) virtualnom računalu uzeti određeni postotak CPU resursa. Sve vrijednosti oko 0% su prihvatljive, a sve više od toga znači kako hipervizor znatnije oduzima resurse virtualnom računalu.
    - `%guest` - je opterećenje CPUa virtualnih računala gdje se prikazuje koliko CPU vremena je potrebno za opsluživanje virtualnih procesora. Naime na ***hipervizoru*** (engl. *Host*) ovo vrijeme znači CPU vrijeme potrošeno na sva virtualna računala (engl. *Guest*) koja se pokreću na konkretnom poslužitelju.
- `%idle` - prikazuje postotak koliko je CPU slobodan odnosno ne radi ništa.

Postoje i drugi korisni prekidači, a jedan od njih je `-P` s kojim možemo pratiti zauzeće prema svim gore navedenim parametrima, ali za pojedinu ili sve CPU jezgre. Pogledajmo primjer za sve CPU jezgre, na računalu koje ima 8 CPU jezgri:

```
mpstat -P ALL

Linux 2.6.32-48-pve (Server-1)    02/01/2018      _x86_64_        (8 CPU)
06:57:50 PM  CPU  %usr %nice   %sys %iowait   %irq   %soft  %steal  %guest   %idle
06:57:50 PM  all  1.70  0.00   0.66    0.06   0.00    0.03    0.00    0.21   97.34
06:57:50 PM    0  2.93  0.00   0.97    0.19   0.00    0.08    0.00    0.22   95.60
06:57:50 PM    1  2.17  0.00   0.72    0.21   0.00    0.03    0.00    0.22   96.64
06:57:50 PM    2  2.45  0.01   1.04    0.02   0.00    0.03    0.00    0.21   96.24
06:57:50 PM    3  1.80  0.01   1.01    0.02   0.00    0.03    0.00    0.25   96.89
06:57:50 PM    4  1.57  0.00   0.41    0.01   0.00    0.02    0.00    0.17   97.82
06:57:50 PM    5  0.91  0.00   0.43    0.01   0.00    0.02    0.00    0.21   98.42
06:57:50 PM    6  0.92  0.00   0.46    0.00   0.00    0.01    0.00    0.24   98.37
06:57:50 PM    7  0.83  0.00   0.25    0.00   0.00    0.01    0.00    0.16   98.74
```

Ovakva statistika može biti korisna, ako sumnjamo da su određene CPU jezgre preopterećenije u odnosu na druge, prema bilo kojem od vidljivih parametara. U našem slučaju nema odstupanja i sve CPU jezgre su vrlo malo i ujednačeno opterećene.
Slijedeća vrlo važna statistika je detaljnija slika o tome kako su hardverski **IRQ** (*interrupti*) odnosno signali prekida raspoređeni prema CPU jezgrama. To dobivamo naredbom: `mpstat -I CPU`, koju nećemo izlistati jer je dugačka.

Važno je razumjeti da su hardverski signali prekida popisani tako da je u svakom stupcu jedan signal prekida (*IRQ*), za pojedinu CPU jezgru, a koje su popisane u svakom redu po jedna. Prema principu vidljivom u tablici dolje:

| CPU JEZGRA | IRQ 0 | IRQ 1 | IRQ *XY* |
|---|---|---|---|
| 0 | vrijednost | vrijednost | vrijednost |
| 1 | … | … | … |

Dodatno, možemo vidjeti i sumiran broj signala prekida (*IRQ*) za sve CPU jezgre, sa sljedećom naredbom:

```
mpstat -I SUM

Linux 2.6.32-48-pve (Server-1)    02/01/2018      _x86_64_        (8 CPU)
07:23:23 PM  CPU    intr/s
07:23:23 PM  all   7178.16
```

Na prethodnom ispisu vidimo kako se trenutno odrađuje `7178` hardverskih signala prekida u sekundi, za cijeli poslužitelj odnosno računalo.

Sljedeća korisna stvar je informacija o **softverskim signalima prekida**[202] koje koriste razni podsustavi Linuxa poput:

- Diskovnog (I/O) i mrežnog podsustava.
- Aplikacija (programa/procesa) u interakciji sa sustavom ili s drugim aplikacijama.
- Mnogih drugih komponenti linuxa.

Pogledajmo kako vidjeti i statistike vezane za softverske signale prekida, s naredbom `mpstat`:

```
mpstat -I SCPU

Linux 2.6.32-48-pve (Server-1)    02/01/2018      _x86_64_        (8 CPU)

07:04:13 PM  CPU  HI/s TIMER/s NET_TX/s  NET_RX/s BLOCK/s BLOCK_IOPOLL/s TASKLET/s SCHED/s  HRTIMER/s  RCU/s
07:04:13 PM    0  0.00  121.86  0.20    163.58    2.41    0.00            0.45      20.83     1.19    128.93
07:04:13 PM    1  0.00   99.30  0.00     23.20    1.94    0.00            0.00      16.43     1.00    105.21
07:04:13 PM    2  0.00  117.87  0.00     28.16    0.00    0.00            0.00      17.34     0.62    127.18
07:04:13 PM    3  0.00  111.57  0.00     21.74    0.00    0.00            0.00      14.08     0.57    124.66
07:04:13 PM    4  0.00   72.38  0.00     16.76    0.00    0.00            0.00      14.73     1.11     74.29
07:04:13 PM    5  0.00   52.25  0.00     12.68    0.00    0.00            0.00       9.99     2.06     54.88
07:04:13 PM    6  0.00   35.56  0.00      6.92    0.00    0.00            0.00       7.35     0.91     36.28
07:04:13 PM    7  0.00   27.31  0.00      5.30    0.00    0.00            0.00       5.68     0.91     28.07
```

Pogledajmo opis navedenih stupaca:

- `CPU` – označava CPU jezgru za koju se rade statistike.
- `HI/s` – predstavlja softverske signale prekida (*soft IRQ*) visokog prioriteta (*HIgh Priority*).
- `TIMER/s` – predstavlja *timere* vezane za softverske signale prekida (*interrupte).*
- `NET_TX/s` – predstavlja *softverske signale prekida* vezane za slanje mrežnih paketa na mrežnu karticu.
- `NET_RX/s` – predstavlja *softverske signale prekida* vezane za primanje mrežnih paketa sa mrežne kartice.
- `BLOCK/s` – predstavlja *softverske signale prekida* vezane za operacije nad blok uređajima (diskovima).
- `BLOCK_IOPOLL/s` – predstavlja *softverske signale prekida* vezane za operacije nad blok uređajima (diskovima), gdje je moguće prebacivanje na drugu CPU jezgru.
- `TASKLET` – daju nam mogućnost odgode određene funkcionalnosti, za kasnije izvršavanje.
- `SCHED` – predstavlja mogućnost da se pokreće barem jedan `kthread` proces po pojedinoj CPU jezgri te da se on i izvršava (za razne funkcionalnosti).
- `HRTIMER` – daje mogućnost da se određene funkcionalnosti, koje bi se trebala odraditi (izvršiti) nakon isteka *timera*, mogle prebaciti na drugu CPU jezgru.
- `RCU` – **RCU** funkcionalnost (engl. *Read-copy update*) je sinkronizacijski mehanizam pomoću kojeg se operacije čitanja, kopiranja i osvježavanja podataka mogu bolje skalirati na veći broj istovremenih procesa/programa i na više paralelnih CPU jezgri.



Vezano za ***softverske signale prekida*** pogledajte poglavlje:
**10.5.1.3 Servis *ksoftirqd i softverski signali prekida*.**



Naredba `mpstat` svoje statistike izvlači i iz mnogih datoteka unutar posebnog `/proc` datotečnog sustava poput: `/proc/interrupts`, `/proc/softirqs` te datoteke `/proc/stat`.

**Izvor informacija: (201),(202),** `man mpstat`,`man 5 proc`.

### 10.7.2.3.7. Naredba *pidstat* za CPU (ali i memoriju i disk prema potrebi)

**Slijedi napredna cjelina!**
Naredba `pidstat` koristi se za statistike opterećenja procesora, memorije ili diskovnog podsustava za željeni proces (pokrenuti program), a i ona dolazi u paketu *sysstat*. Pokrenimo ju pa ćemo objasniti što sve prikazuje:

```
pidstat

Linux 3.10.0-862.11.6.el7.x86_64 (svr101)      10/21/2019      _x86_64_        (32 CPU)

19:22:18 AM   UID       PID    %usr %system  %guest    %CPU   CPU  Command
19:22:18 AM     0         1    0.01    0.02    0.00    0.03    11  systemd
19:22:18 AM     0         2    0.00    0.00    0.00    0.00    12  kthreadd
19:22:18 AM     0         3    0.00    0.00    0.00    0.00     0  ksoftirqd/0
19:22:18 AM     0         8    0.00    0.00    0.00    0.00     0  migration/0
19:22:18 AM     0        10    0.00    0.18    0.00    0.18    24  rcu_sched
```

Ako ovu naredbu pokrenemo ovako, bez prekidača, dobivamo sumariziranu statistiku za sve jezgre procesora (*CPU*) te vidimo:

- `UID` – prikazuje ***UID*** (ime korisnika) koji je pokrenuo određeni program (proces).
- `PID` – prikazuje ***PID*** (*process ID*) pokrenutog programa (procesa).
- `%usr` – prikazuje koliko CPU resursa za određeni program (proces) se koristi u korisničkom prostoru Linuxa.
- `%system` – prikazuje koliko se CPU resursa za određeni program (proces) koristi u kernel prostoru Linuxa.
- `%guest` – prikazuje koliko CPU resursa za određeni program (proces) se koristi u virtualnom prostoru; pokretanjem virtualnog procesora koji se iskorištava unutar virtualnog računala, a gleda s razine hipervizora (fizičkog računala).
- `%CPU` – prikazuje koliko se ukupno CPU resursa za određeni program (proces) koristi (postotak CPU jezgri).
- `CPU` – prikazuje na kojoj CPU jezgri se trenutno izvršava program.

Međutim, moguće je pratiti i statistike opterećenja procesa prema diskovnom podsustavu, pokrenemo li ovu naredbu na slijedeći način.

```
pidstat -d
```



Za detalje ovakve primjene, pogledajte primjere u poglavlju:
**14.2. Praćenje performansi I/O sustava**.

Moguće je pratiti i statistike opterećenja procesa prema zauzeću memorije (`-r`) pokrenemo li ovu naredbu na sljedeći način:

```
pidstat -r
Linux 3.10.0-862.11.6.el7.x86_64 (svr101)    10/21/2019    _x86_64_        (32 CPU)

19:36:34 AM   UID   PID  minflt/s  majflt/s     VSZ    RSS  %MEM  Command
19:36:34 AM     0     1      1.14      0.00  193244   2820  0.00  systemd
19:36:34 AM     0   547      2.84      0.00  153872  59300  0.10  systemd-journal
19:36:34 AM     0   572      0.00      0.00   44332    576  0.00  systemd-udevd
```

Ovdje u ispisu imamo dodatne stupce koje ćemo sada objasniti:

- `minflt/s` – prikazuje takozvani „***Minor page fault***“ [(250),(252)].



Pogledajte poglavlje: **12.4. Anatomija programa u memoriji**.

- `majflt/s` – prikazuje takozvani „***Major page fault***“ *koji raste samo primjerice tijekom učitavanja programa (podataka) s diska u (RAM) memoriju te se treba smanjiti nakon toga* [(250),(252)].



Pogledajte poglavlje: **12.4. Anatomija programa u memoriji**.

- `VSZ` – prikazuje zauzeće virtualnog dijela memorije.



Pogledajte poglavlje: **12.3.3.4. Naredbe *ps* i *top* za memoriju.**

← Zauzeće se iskazuje u kilobajtima (kB).

- `RSS` – prikazuje stvarno zauzeće memorije.



Pogledajte poglavlje: **12.3.3.4. Naredbe *ps* i *top* za memoriju.**

← Zauzeće se također iskazuje u kilobajtima (kB).

- `%MEM` – prikazuje postotak ukupnog zauzeća memorije.

Dodatno, možemo tražiti statistike samo za određene procese na nekoliko načina. Primjerice želimo statistike i diskovnog i memorijskog (-r) i CPU zauzeća za proces s ***PID*** brojem 43904, sve u jednom redu (-h) svake sekunde (1), 100 puta:

```
pidstat 1 100 -h -u -r -p 43904
```

Ili ako želimo iste statistike za sve procese koji se zovu „mysql" ili imaju u nazivu imena procesa tu riječ:

```
pidstat 1 100 -h -u -r -C mysql
```

Ili ako želimo iste statistike za sve procese koji se zovu „mysql" ili „apache" ili imaju u nazivu imena procesa te riječi:

```
pidstat 1 100 -h -u -r -C "mysql|apache"
```

Za ljepši prikaz možemo koristiti i prekidač -h:

```
pidstat -h

Linux 5.16.9-1.el8.elrepo.x86_64 (localhost.localdomain)   02/24/2022   x86_64_    (2
CPU)

# Time         UID   PID    %usr %system  %guest   %wait    %CPU   CPU  Command
05:54:38 AM      0     1    0.00    0.01    0.00    0.00    0.01     0  systemd
05:54:38 AM      0    12    0.00    0.00    0.00    0.00    0.00     0  ksoftirqd/0
05:54:38 AM      0    13    0.00    0.00    0.00    0.01    0.00     0  rcu_preempt
05:54:38 AM      0    19    0.00    0.00    0.00    0.00    0.00     1  ksoftirqd/1
05:54:38 AM      0    33    0.00    0.00    0.00    0.00    0.00     1  khugepaged
05:54:38 AM      0   319    0.00    0.00    0.00    0.00    0.00     1  scsi_eh_0
05:54:38 AM      0   447    0.00    0.00    0.00    0.00    0.00     1  systemd-journal
05:54:38 AM      0   479    0.00    0.00    0.00    0.00    0.00     0  systemd-udevd
05:54:38 AM     81   547    0.00    0.00    0.00    0.00    0.00     1  dbus-daemon
05:54:38 AM      0   549    0.00    0.00    0.00    0.00    0.00     0  irqbalance
05:54:38 AM    993   562    0.00    0.00    0.00    0.00    0.00     1  chronyd
05:54:38 AM      0   565    0.00    0.00    0.00    0.00    0.00     1  ksmtuned
05:54:38 AM      0   596    0.00    0.00    0.00    0.00    0.00     0  irq/18-vmwgfx
05:54:38 AM      0   609    0.00    0.00    0.00    0.00    0.00     0  sssd_be
```



Naredba pidstat svoje statistike izvlači i iz mnogih datoteka unutar posebnog /proc datotečnom sustava poput: /proc/stat te posebnih datoteka za svaki proces (***PID***) u strukturi: /proc/***PID***/status.

Pogledajmo koje još korisne opcije i prekidače nudi naredba pidstat:

- -d – za prikaz diskovnih (I/O) statistika svih pokrenutih procesa.
- -h – za horizontalni i ljepši prikaz većine statistika svih pokrenutih procesa.
- -I – za horizontalni prikaz statistika upotrebe prema pojedinim jezgrama procesora (za ***SMP*** ili ***NUMA*** sustave).
- -l – za prikaz statistika prema pojedinim jezgrama procesora (za ***SMP*** ili ***NUMA*** sustave) uz prikaz s kojim opcijama i parametrima je pojedini proces pokrenut.
- -p PID – za prikaz statistika pokrenutog procesa prema njegovom PID broju.
- -R – za prikaz statistika pokrenutih *realtime* procesa prema njihovim *realtime* prioritetima (brojčano).
- -r – za prikaz statistika o memoriji, svih pokrenutih procesa.
- -s – za prikaz statistika o *stack* memoriji, svih pokrenutih procesa.
- -t – za prikaz statistika o *programskim nîtima*, svih pokrenutih procesa, uz prikaz ***TGID*** i ***TID*** brojeva.
- -u – za prikaz statistika o zauzeću procesora (*CPU utilization*), svih pokrenutih procesa.
- -w – za prikaz statistika o *programskim nîtima*, svih pokrenutih procesa, prema svakom procesu.

**Izvor informacija: (250),(252),** **man pidstat**,**man 5 proc**.

### 10.7.2.4. Izolacija jezgri procesora

Iako smo vidjeli kako je moguće pokretati željene programe na ***SMP*** ili ***NUMA*** sustavima, na samo odabranim jezgrama procesora, moguća je izolacija i na još nižoj razini sustava. CPU izolacija je način da se sistemski planer (*task scheduler*) prisili da koristi samo određene logičke CPU jezgre za programe, dok su ostale logičke jezgre slobodne za specijalizirane primjene to jest programe. Važno je razumjeti da Linux kernel sadrži podsustav za raspodjelu izvršavanja *procesa* to jest pokrenutih programa, koji je odgovoran i za premještanje izvršavanja *procesa* s jedne jezgre procesora na drugu. U normalnim uvjetima, navedeni podsustav za raspodjelu *procesa*, premješta *procese* po CPU jezgrama dostupnima sustavu, u nastojanju da osigura jednako vrijeme izvršavanja za sve *procese* koji se aktivno izvršavaju te ravnomjerno opterećenje svih jezgri. Ipak on može i zadržati proces koji se izvršava na određenoj CPU jezgri, ako nema potrebe da ga migrira na neku drugu CPU jezgru. Postoje i primjene u kojima ovakvo (standardno) prebacivanje na izvršavanje *procesa* s jedne jezgre na drugu i sâmo prebacivanje između programskih nîti unutar *procesa* (tzv. *context switch*) nije dobro što se tiče performansi. To su najčešće mrežne aplikacije koje moraju obrađivati ekstremno veliki broj mrežnih konekcija ili operacija nad mrežnim paketima. Izolacija CPU resursa se najčešće koristi i u virtualizaciji gdje je preporučeno izolirati procesorske jezgre za zadatke koje obavlja *hipervizor* (fizičko računalo) i/ili njegov centralni mrežni servis (pr. ***Open vSwitch***), a preostale jezgre propustiti za upotrebu unutar virtualnih računala. Tako smo praktično rezervirali procesorske resurse za zadatke koje odrađuje *hipervizor* i njegove komponente (pr. mrežni i diskovni servisi i slično). Dostupna su nam dva načina kako to postići:

- Kroz ***isolcpus*** opciju na razini kernela, tijekom pokretanja sustava, u **GRUB (GRUB2)** konfiguraciji.
- Pomoću ***cpuset*** mehanizma, preko ***cgroups*** komponente sustava (obično se koristi za linux kontejnere).

Prva metoda na kojoj je naš fokus je pomoću ***isolcpus***, koja specifične CPU jezgre uklanja iz sistemskog raspodjeljivača *procesa* (planera) znanog kao *task scheduler*. Naime *task scheduler*, osim standardnog konstantnog pauziranja svakog *procesa* kada mu se završi vremenski okvir za izvršavanje, te ponovnog pokretanja, mora povremeno prekidati regularno izvršavanje *procesa* (programa) na određenim jezgrama procesora kako bi utvrdio je li potrebno rebalansiranje ili migracija *procesa* na neku drugu jezgru procesora (CPUa). Osim toga procesor je zadužen i za obradu signala prekida.

Važno je razumjeti da čak i ako koristimo ***isolcpus*** i izoliramo određene CPU jezgre od *task schedulera*, na tim izoliranim jezgrama i dalje možemo ručno pokrenuti programe, i to samo i isključivo pomoću naredbi: `taskset` ili `numactl`.



Za više detalja o ***task scheduleru*** i prebacivanju između izvršavanja više procesa na sustavu pogledajte poglavlje: **9.3.1. Context switching.**

Popis *isolcpus* jezgri konfigurira se statički u vrijeme pokretanja operativnog sustava. Možete ga promijeniti samo ponovnim pokretanjem cijelog sustava (restartom) s nekom drugom vrijednošću. Ova konfiguracija radi se unutar *GRUB2* konfiguracije sustava. Međutim treba biti svjestan toga da **isolcpus** izolira programe koji se izvršavaju u korisničkom prostoru (*user space*), ali se neće izolirati CPU jezgre od aktivnosti kernela, poput primjerice servisa kernela: *[watchdog], [kworker], ….*
Dakle procesa vidljivih (pr. s naredbama `ps` ili `top`) unutar uglatih zagrada, kao **[..xy..]**. To znači da će sve komponente sustava na najnižoj razini to jest one koje se izvršavaju u izoliranom prostoru kernela (*kernel space*) i dalje moći koristiti i koristit će ove jezgre procesora, jer se na to ne može utjecati. Ipak to u praksi ne predstavlja problem. U ovoj priči važno je da postoji mehanizam izolacije CPU jezgri za *normalne* programe to jest one koji se pokreću u korisničkom prostoru, a to su praktično svi servisi i programi koje kao bilo koji korisnik na sustavu i pokrećemo.



Za više detalja o ***GRUB2 boot loaderu***, postavkama i opcijama koje nudi, pogledajte poglavlje: **11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

U *GRUB2* konfiguracijskoj datoteci, u sekciji s kernel opcijama, potrebno je dodati opciju `isolcpus=` u argumente koji se šalju samom kernelu, ali unutar navodnika, na mjestu gdje se spominju argumenti (`args=`). Drugi način je modifikacija standardnih postavki *GRUB2* u datoteci: `/etc/default/grub`, ali u parametru: `GRUB_CMDLINE_LINUX=`.
Još lakši način je to sve napraviti s programom `grubby`. Stoga ćemo to i napraviti s programom `grubby` na sljedeći način.
Primjer je za izolaciju jezgri **0** i **1**, za sve kernele koje imamo konfigurirane na sustavu:

```
grubby --update-kernel=ALL --args=isolcpus=0-1
```

Sada je potrebno restartati sustav, a potom provjeriti kako se kernel učitao to jest s kojim opcijama i parametrima se pokrenuo:

```
cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos2)/vmlinuz-5.14.0-70.22.1.el9_0.x86_64 root=UUID=bd9a0762-0159-470f-a55f-497bcd6a0bb0 ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M resume=UUID=1b1d653b-65d3-4dca-85b9-6edc7ba4d3fe isolcpus=0-1
```

Vidimo zadnju opciju "`isolcpus=0-1`" što smo i očekivali. Provjerimo jesmo li izolirali CPU jezgre **0** i **1** od *task schedulera*. To ćemo uraditi tako da ćemo prvo instalirati pa pokrenuti program `stress` koji opterećuje sve navedene CPU jezgre. Pošto naš testni sustav ima osam CPU jezgri, to ćemo postići sa sljedećim naredbama:

```
yum install stress
stress -c8
```

U drugom terminalu možemo pratiti opterećenje svih jezgri, uz očekivanje da će CPU jezgre **0** i **1** biti neopterećene:

```
sar -P ALL 2 200
```

**Izvori informacija: (1314),(1315),(1316),(1317),** `man grubby`, `man sched_setaffinity`, `man stress`, `man sar`

### 10.7.2.5. Ograničavanje CPU resursa i cgroup mehanizam

**Cgroups** (skraćeno od kontrolne grupe) značajka je kernela Linuxa koja može ograničavati i izolirati upotrebu resursa računala poput: CPU, memorije, mreže ili diskovnog (I/O) podsustava, za skupinu željenih procesa (pokrenutih programa).

Jedan od ciljeva dizajna *cgroups* mehanizma je osigurati jedinstveno sučelje za mnoge različite slučajeve upotrebe:
- Od kontrole pojedinačnih procesa (primjerice upotrebom naredbe `nice`).
- Preko virtualizacije na razini operativnog sustava odnosno upotrebe Linux kontejnera poput: ***OpenVZ***, ***Linux-VServer***, ***LXC***, ***Docker***, ***Podman*** i drugih.
- Sve do upotrebe za virtualizaciju (***KVM+QEMU***); prilikom pokretanja virtualnog računala, pa čak i promjene tijekom njegovog rada.

*Cgroups* mehanizam pruža sljedeće značajke:
- **Ograničavanje resursa**: može se postaviti da ne premaši konfigurirano ograničenje memorije (što također uključuje predmemoriju datotečnog sustava), ograničenje propusnosti diskovnih i mrežnih resursa, ograničenje CPU resursa poput CPU kvote ili ograničenje na korištenje CPU resursa (CPU set) zadanog procesa.
- **Prioritizacija**: određivanjem prioriteta određene grupe, koja može dobiti veći udio u iskorištenju procesora (CPU-a ili I/O propusnosti diska ili mreže.
- **Mjerenje** korištenja resursa, što se može koristiti, na primjer, za potrebe naplate iskorištenih resursa.
- **Kontroliranje** rada procesa: zamrzavanje grupa procesa, njihovo provjeravanje i ponovno pokretanje prema potrebi.

Naime *cgroup* se najčešće primjenjuje na skup procesa koji su vezani istim kriterijima povezanima sa skupom parametara ili ograničenja. Ove grupe mogu biti hijerarhijske, što znači da svaka grupa nasljeđuje ograničenja od svoje nadređene grupe. Kernel omogućuje pristup višestrukim kontrolerima (također zvanim podsustavima) kroz sučelje *cgroup*; na primjer, kontroler "*memorije*" ograničava korištenje memorije, "*cpuset*" ograničava izvršavanje odnosno korištenje CPU-a i tako dalje.



Za više detalja o imeničnim prostorima Linuxa i ***cgroup***, pogledajte poglavlje:
**9.4.5. Izolirani prostori Linuxa (*Linux namespaces*).**

**Kernel resource controllers**
Kontrolori resursa kernela omogućuju upotrebu funkcionalnost kontrolnih grupa odnosno navedeni *cgroups*. Počevši od *Red Hat 9* Linuxa podržani su različiti kontroleri za kontrolne grupe inačice 1 (*cgroups-v1*) i kontrolne grupe inačice 2 (*cgroups-v2*). Kontroler resursa, koji se naziva i podsustav kontrolne grupe, je podsustav kernela koji predstavlja jedan resurs, kao što je primjerice CPU vrijeme za obradu (programa), sustav memorije, propusnosti mreže ili (I/O) diska.

Pri tome Linux kernel pruža niz kontrolera resursa koje automatski montira **systemd** servis. U datoteci `/proc/cgroups` možete pronaći popis trenutno montiranih kontrolera resursa sustava.
Direktorij `/sys/fs/cgroup/`, također nazvan *root* kontrolna grupa, prema zadanim postavkama sadrži datoteke sučelja (počevši s *cgroup*) i datoteke specifične za kontroler kao što je primjerice `cpuset.cpus.effective`. Osim toga, postoje neki direktoriji vezani uz **systemd** servis, kao što su `/sys/fs/cgroup/init.scope`, `/sys/fs/cgroup/system.slice` i `/sys/fs/cgroup/user.slice`.

Mi ćemo koristiti *cgroup2* što znači da za *cgroup2* mora biti montiran poseban (pseudo) *cgroup2* datotečni sustav:
```
mount -l | grep cgroup
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime)
```
Što znači da unutar vršnog direktorija `/sys/fs/cgroup/` postoji stablo poddirektorija i konfiguracijskih (i drugih) datoteka.

*Cgroup* v.1. nam vezano za CPU resurse nudi sljedeće kontrole:

| | |
|---|---|
| `cpu` | Prilagođava parametre *Completely Fair Scheduler planera* [task menadžera] (**CFS**) za zadatke kontrolne grupe. CPU kontroler je montiran zajedno s `cpuacct` kontrolerom na istoj točki montiranja. |
| `cpuacct` | Stvara automatska izvješća o CPU resursima koje koriste zadaci u kontrolnoj grupi. `cpuacct` je montiran zajedno s `cpu` kontrolerom na istoj točki montiranja. |
| `cpuset` | Ograničava izvršavanje zadataka kontrolne grupe samo na određenom podskupu CPU-a i usmjerava zadatke da koriste memoriju samo na određenim memorijskim čvorovima. |

Dok *cgroup* v.2. to pojednostavljuje te nam po pitanju CPU resursa daje sljedeće kontrole:

| | |
|---|---|
| `cpu` | Prilagođava parametre *Completely Fair Scheduler planera* [task menadžera] (**CFS**) za zadatke kontrolne grupe i generira automatska izvješća o CPU resursima koje koriste zadaci u kontrolnoj grupi. |
| `cpuset` | Ograničava izvršavanje zadataka kontrolne grupe samo na određenom podskupu CPU-a i usmjerava zadatke da koriste memoriju samo na određenim memorijskim čvorovima. Podržava samo osnovne funkcije (`cpus{,.effective}, mems{,.effective}`). |

Kontrolne grupe (*cgroups*) mogu se koristiti na nekoliko načina:

- Direktnim pristupom i promjenama na pseudo datotečnom sustavu (*cgroups*).
- Upotrebom specijaliziranih naredbi, poput: `cgcreate`, `cgexec` i `cgclassify`.
- Indirektno, preko drugih programa ili servisa koji imaju metode rada s *cgroups* sustavom, poput: ***Systemd*** (indirektno ***Proxmox VE***), **LXC**, **LibVirt** (te indirektno ***OpenStack***, ***Red Hat/Oracle Virtualization***, … ), ***Docker***, ***Podman***…

Ako ste ručno željeli raditi sa *cgroup* mehanizmima[(746)], instalirajte si softverski paket koji sadrži potrebne programe za rad s njima:

```
yum install -y libcgroup-tools
```

S njim ćete dobiti i slijedeće naredbe (navest ćemo ih samo nekoliko):`cgcreate`, `cgexec`, `cgdelete`, `cgset`, `cgclear` i druge.

Od **Red Hat 7.7.** paket **`libcgroup-tools`** se više ne koristi jer je sada ***systemd*** jedini zadužen za ***cgroup*** mehanizme!

Mi ćemo dalje nastaviti s novijim primjerima kontrole ***cgroups*** mehanizama preko ***systemd*** servisa.
Prema zadanim postavkama, **systemd** stvara novu **cgroup(u)** pod ***system.slice*** strukturom za svaki servis koji nadzire.
Zadano ponašanje je moguće promijeniti rekonfiguracijom servisa ili upotrebom naredbe: `systemctl set-property`.
Ispis kontrolnih grupa prema servisima možemo vidjeti s naredbom (skratili smo ispis):

```
systemd-cgls

Control group /:
-.slice
├─user.slice (#1075)
│ → user.invocation_id: 5fb7d842056d4358979d6e3bf959b058
│ → trusted.invocation_id: 5fb7d842056d4358979d6e3bf959b058
│ └─user-0.slice (#3140)
│   → user.invocation_id: 43982ccefdc04d11a2fcc13d7c6f319c
│   → trusted.invocation_id: 43982ccefdc04d11a2fcc13d7c6f319c
│   ├─session-1.scope (#3392)
│   │ ├─4150 sshd: root [priv]
│   │ ├─4164 sshd: root@pts/0
│   │ ├─4165 -bash
│   │ ├─4287 systemd-cgls
│   │ └─4288 less
│   └─user@0.service … (#3212)
```

Primjerice želimo li za neki servis, primjerice **rsyslog** ograničiti CPU resurse preko *cgroup* mehanizma s `CPUShares` opcijom, to možemo napraviti ovako:

```
systemctl set-property rsyslog.service CPUShares=1024
```

Tada će se automatski kreirati *unit* datoteka `50-CPUShares.conf` unutar direktorija:
`/etc/systemd/system.control/rsyslog.service.d/`. Ona će sadržavati samo:

```
[Service]
CPUShares=1024
```

Nakon toga je samo potrebno restartati servis:

```
systemctl restart rsyslog.service
```

Pogledajmo samo neke *systemd* opcije vezana za *cgroup* mehanizme, koje ograničavaju CPU resurse:

| Systemd (systemctl) opcija | |
|---|---|
| **`CPUShares=`** | Zadana vrijednost je 1024. Povećanjem broja jedinica dodjeljujete više CPU vremena (u odnosu na druge servise u istoj *cgroup* grupi). Primjerice ako imamo tri servisa u istoj grupi i ako svaki ima postavljenu vrijednost 1024, to znači da će svaki od njih dobiti 33.33% CPU vremena za obradu. |
| **`CPUWeight=`** | Ako je postavljena na cjelobrojnu vrijednost, dodijelite navedeni težinski koeficijent procesorskog vremena izvršenim procesima, ako se u sustavu koristi jedinstvena hijerarhija kontrolne grupe. Ove opcije kontroliraju atribut kontrolne grupe "**cpu.weight**". Dopušteni raspon je od 1 do 10000. Dostupno CPU vrijeme podijeljeno je između svih servisa (*unit-a*) unutar jednog odsječka vremena u odnosu na njihov težinski koeficijent CPU vremena. Već težinski koeficijent znači više CPU vremena, manji znači manje vremena (za obradu). |
| **`CPUQuota=`** | Dodjeljuje određenu CPU vremensku kvotu izvršenim procesima (servisima). Uzima vrijednost u postotku, sa sufiksom "%". Postotak određuje koliko CPU vremena koji servis mora dobiti maksimalno, u odnosu na ukupno CPU vrijeme dostupno na jednoj jezgri CPU-a. Koristite vrijednosti > 100% za dodjelu CPU vremena na više od jednog CPU-a (1 CPU=100%, 2 CPU=200%, …) |
| **`AllowedCPUs=`** | Za ograničavanje procesa (servisa) koji se izvršavaju samo na određenim jezgrama procesora. Uzima popis CPU indeksa ili raspona odvojenih razmakom ili zarezima. |

Za opis svih dostupnih *cgroup* opcija pokrenite:

```
man systemd.resource-control
```

Sada još provjerimo podržava li naše računalo cgroups sustav:

```
cat /proc/cgroups

#subsys_name    hierarchy       num_cgroups     enabled
cpuset     0          783        1
cpu        0          783        1
cpuacct    0          783        1
blkio      0          783        1
memory     0          783        1
devices    0          783        1
freezer    0          783        1
net_cls    0          783        1
perf_event      0          783        1
net_prio        0          783        1
hugetlb    0          783        1
pids       0          783        1
rdma       0          783        1
misc       0          783        1
```

🡐 U ovom slučaju to znači da podržava.

U slučaju želje za ručnim radom (ako se ne želimo oslanjati na ***systemd*** što je preporučeno), možemo primjerice napraviti novu strukturu direktorija za naš program:

```
mkdir /sys/fs/cgroup/moja_grupa
```

Nakon što smo napravili ovaj direktorij, sustav će u njemu kreirati sve cgroup2 datoteke za ograničavanje resursa:

```
ls
cgroup.controllers       cpu.max             cpu.stat               memory.stat
cgroup.events            cpu.weight             memory.swap.current    memory.reclaim
cgroup.freeze            cpu.weight.nice        memory.swap.events     memory.peak
cgroup.kill              cpu.max.burst          memory.current         memory.swap.high
cgroup.max.depth         memory.events          memory.swap.max
cgroup.max.descendants   memory.events.local    memory.swap.peak
cgroup.procs             memory.high            memory.zswap.current
cgroup.stat              memory.low             memory.zswap.max
cgroup.subtree_control   memory.max             pids.current
cgroup.threads           memory.min             pids.events
cgroup.type              memory.numa_stat       pids.max
cpu.idle                 memory.oom.group       pids.peak
```

Vezano za ograničenje CPU resursa imamo dostupne datoteke:

- `cpu.max` - za *cgroups 2* je zadano: `max 100000`. Broj 100000 predstavlja period u mikrosekundama koji određuje trajanje vremenskog intervala tijekom kojeg se provjerava je li proces iskoristio svoju kvotu. Ako dodajemo neku vrijednost ona će biti prva; primjerice ako dodamo 50000 , druga vrijednost će ostati 100000, što bi značila kvota koja određuje koliko dugo proces smješten u toj kontrolnoj grupi može koristiti CPU resurse tijekom određenog vremenskog intervala: konkretno 50.000 : 100.000 znači 50% CPU vremena.

Postavimo upravo ovakvo ograničenje na 50% CPU vremena (resursa):

```
echo 50000 > /sys/fs/cgroup/moja_grupa/cpu.max
```

I dobili smo zapisane vrijednosti: `50000 100000`. Sada naš program (recimo da mu je ***PID*** broj **1540**) dodajemo u *cgroup*:

```
echo 1540 > /sys/fs/cgroup/moja_grupa/cgroup.procs
```

I nakon toga program s ***PID*** brojem **1540** je ubačen u našu grupu nad kojom je aktivirano ograničenje CPU resursa na 50%.

- `cpu.weight` - zadana vrijednost je 100 (težinski koeficijent) [opisan u **`CPUWeight`**].

Kako provjeriti *cgroup* ograničenje za program prema njegovom ***PID*** broju (primjerice za ***PID*** **1540**):

```
cat /proc/1540/cgroup
0::/moja_grupa
```

🡐 Vidimo da pripada našoj grupi (**`/sys/fs/cgroup/moja_grupa/`**)

To možemo vidjeti i ovako:

```
ps -o pid,cgroup -p 1540
    PID CGROUP
   1540 0::/moja_grupa
```

🡐 Vidimo pripadnost našoj grupi.

Isto možemo vidjeti i sa:

```
systemd-cgls
Working directory /sys/fs/cgroup/moja_grupa:
└─1540 /usr/bin/mc -P /var/tmp/mc-root/mc.pwd.1517
```

Dok sve *cgroup-e* možemo pratiti sa:

```
systemd-cgtop
```

**Izvori informacija: (1544),(1545),(1546),(1548),(1549),(1550),** **`man systemctl`**, **`man systemd-cgls`**, **`man systemd.resource-control`**, **`man systemd-cgtop`**.

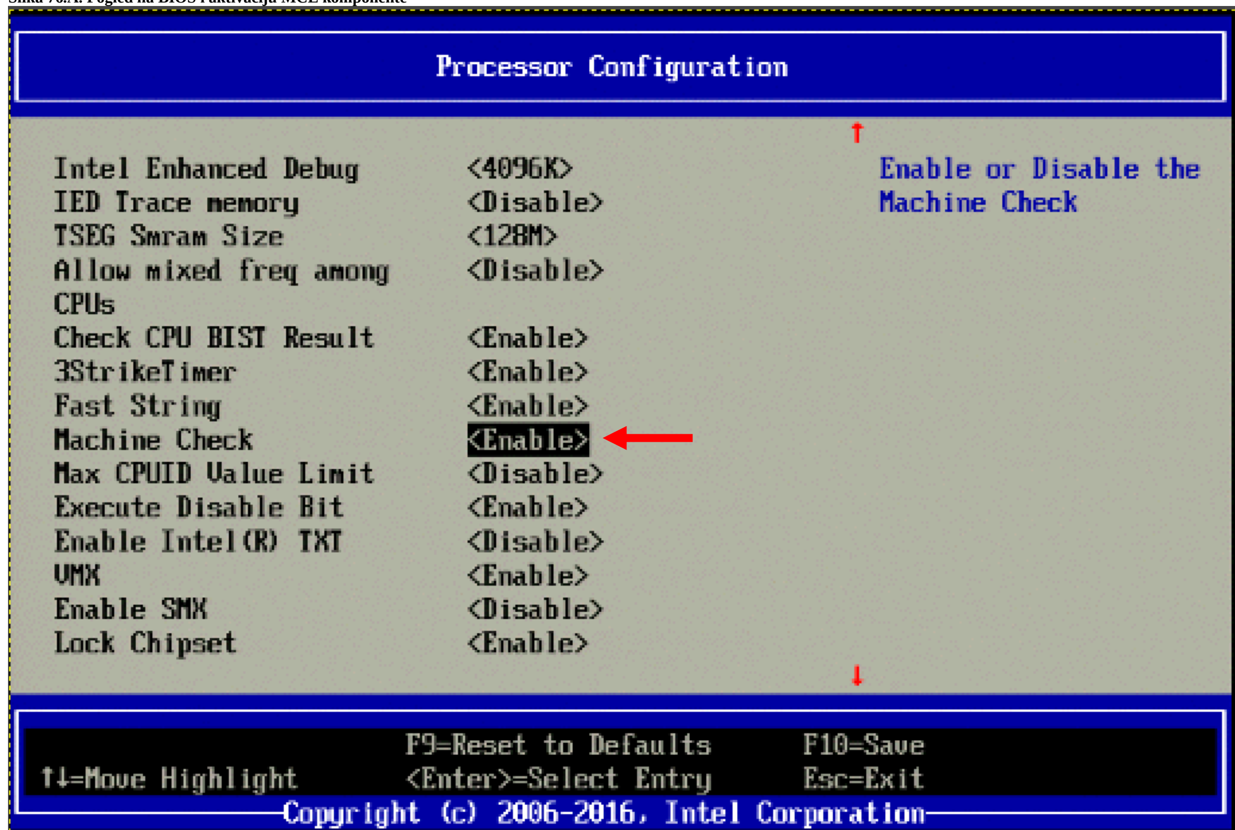
## 10.7.3. Machine Check Architecture (MCA)

*Machine Check Architecture* (**MCA**) je *Intelov* i *AMDov* mehanizam pomoću kojeg CPU može prijaviti hardverske pogreške operativnom sustavu. *Intelovi* procesori počevši od obitelji P6 (negdje od 1995.g.) i Pentium 4 te AMD-ovi procesori počevši od obitelji K7 i K8, kao i *Itanium* arhitektura implementiraju ove mehanizme provjere koji omogućuju detekciju i prijavu hardverskih (strojnih) grešaka, kao što su: greške sistemske sabirnice, ECC greške RAM memorije, pogreške pariteta, pogreške predmemorije procesora te pogreške **TLB** međumemorije. *MCA* radi tako da unutar procesora postoji skup registara odnosno *model specific register* (tzv. **MSR**-ova) koji se koriste za ovu značajku te dodatnih MSR-ova koji se koriste za bilježenje grešaka koje su otkrivene.
Unutar *MCA* arhitekture, ako se pojavi neka greška u hardveru našeg računala, aktivira se takozvani *machine check exception* (**MCE**). Tada se generira signal prekida (*interrupt*) koji se zatim obrađuje od strane operativnog sustava. Operativni sustav prikuplja informacije o pogrešci iz navedenih MSR registara procesora te ih može zabilježiti u sistemskim logovima. Pri tome postoje dvije glavne kategorije MCE pogrešaka: pogreška obavijesti ili upozorenja i fatalna pogreška. Upozorenje će biti zabilježeno porukom "***Machine Check Event logged***" u log datotekama našeg sustava, a kasnije se poruke mogu vidjeti putem nekih od uslužnih programa Linuxa. Pri tome će fatalni MCE uzrokovati nemogućnost odziva sustava ili čak prestanak rada računala, a detalji o MCE-u bit će ispisani i na konzoli sustava.
Često je *machine check exception* (**MCE**) pogreška koja se javlja kada CPU računala otkrije neki hardverski problem, mada ovakva poruka može biti samo informativna i ne previše kritična. U slučaju fatalne greške sustav će pokrenuti proceduru panike kernela kako bi se zaštitio od oštećenja podataka. MCE se oslanja na poseban uređaj (`10, 227`) `/dev/mcelog`.

Da bi podrška za MCE uopće radila potrebna su dva preduvjeta:

- Podrška od strane procesora i BIOS-a računala. Pogledajmo sliku aktivacije MCE u BIOS-u:

**Slika 76.A. Pogled na BIOS i aktivaciju MCE komponente**




Za provjeru, podržava li naš procesor **MCE**, pokrenite sljedeću naredbu:

```
dmidecode -t processor | grep MSR

MSR (Model specific registers)
```

← pošto ovdje imamo vidljiv **MSR** (`MSR`), sve je u redu odnosno podrška postoji.

- Podrška unutar kernela, što možemo provjeriti sa sljedećom naredbom:

```
grep -i CONFIG_X86_MCE /boot/config-`uname -r`
CONFIG_X86_MCE=y
CONFIG_X86_MCELOG_LEGACY=y
CONFIG_X86_MCE_INTEL=y
CONFIG_X86_MCE_AMD=y
CONFIG_X86_MCE_THRESHOLD=y
CONFIG_X86_MCE_INJECT=m
```

← Navedene opcije s kojima je kompiliran kernel su važne za rad MCE-a.
*MCE* sustav koristi signale prekida da bi kreirao poruku o greškama, što je vidljivo iz liste signala prekida (IRQ) [naše računalo ima četiri procesorske jezgre].

Pogledajmo signale prekida vezane za *MCE*:

```
cat /proc/interrupts | grep MCE
           CPU0       CPU1       CPU2       CPU3
MCE:          0          0          0              0       Machine check exceptions
```

← u listi signale prekida (IRQ) vidimo (`MCE:`) što znači da je *MCE* dostupan.

Sumirajmo uzroke *MCE* grešaka:

- Greške u radu procesora poput lošeg/slabog/nestabilnog napajanja.
- Pogreške predmemorije procesora (CPU *cache*: L1, L2 ili L3), **TLB** memorije ili drugih grešaka unutar procesora. Naime procesori koji podržavaju poboljšano izvješćivanje o pogreškama predmemorije sadrže hardver koji prati radni status određenih predmemorija procesora i daje pokazatelj njihovog "zdravlja".
- Pogreške u RAM memoriji ili problemi s ECC memorijom. MCE pohranjuje informacije o greškama ovisno o fizičkom procesoru (*NUMA*), memorijskom kanalu, memorijskom utoru, memorijskom modulu i čak po stranicama memorije (što je posebno korisno kod ECC memorija). Nadalje, ako imamo ECC memoriju, greške možemo otkriti i sa:

```
cat /sys/devices/system/edac/mc/mc?/ce*count
0
0
0
0
```

← Ako imamo sve nule u ispisu, stanje ECC je uredno. Međutim ako nemamo ECC memoriju i želimo testirati memoriju, sustav možemo restartati i pokrenuti primjerice memtest86+ to jest testiranje RAM memorije.

- Neadekvatno hlađenje ili  pregrijavanje procesora.
- Greške sistemskih sabirnica: neki noviji procesori Intel Xeon podržavaju izvješćivanje o neispravljenim pogreškama putem nove značajke **IOMCA**. Dakle ovdje se mogu detektirati oštećeni podaci na *PCI express* sabirnicama. Ove pogreške prijavljuje *mcelog* za neispravljene pogreške. Za ispravljene pogreške i pogreške koje prijavljuje uređaj koristi se **PCI AER**, koji se prijavljuje putem uobičajene log datoteke `/var/log/syslog` (ne u *mcelogu*).
- Greške u BIOS-u: preporuka je napraviti BIOS/*firmware* ili nadogradnju mikrokoda procesora.

Za čitanje i spremanje MCE poruka unutar Linuxa zadužen je `mcelog` servis (i naredba), koji moramo instalirati:

```
yum -y install mcelog
```

Nakon instalacije, potrebno je definirati gdje će se *MCE* poruke spremati, pa je potrebno urediti datoteku: `/usr/lib/systemd/system/mcelog.service` koja mora sadržavati sljedeći redak (za ***Red Hat 8+***):

```
ExecStart=/usr/sbin/mcelog --ignorenodev --daemon --syslog --foreground --logfile=/var/log/mcelog
```

Nakon što snimimo ovu promjenu, pokrenimo rekreiranje servisa koji smo upravo rekonfigurirali:

```
systemctl daemon-reload
```

Tek sada možemo restartati servis:

```
systemctl start mcelog
```

A potom ga i trajno aktivirati:

```
systemctl enable mcelog
```

Centralna konfiguracijska datoteka *mcelog* servisa je: `/etc/mcelog/mcelog.conf`.

## Greške i MCE poruke

MCE poruke o greškama odnosno detalje možemo vidjeti u dvije datoteke:

- `/var/log/messages` – ovo je standardna log datoteka za cijeli sustav (koja će sadržavati i *MCE* poruke).
- `/var/log/mcelog`– ovo je nova log datoteka za *MCE* poruke.

Ako sustav ima neke od hardverskih grešaka, u navedenim datotekama ćemo vidjeti nešto poput sljedećih.
U slučaju samo slanja informativne *MCE* poruke, vidjet ćemo nešto poput:

```
…  hostname kernel: Machine check events logged
…  hostname kernel: Machine check events logged
```

Dok u slučaju hardverske greške, možemo imati ovakve poruke:

```
… hostname kernel: mce_notify_irq: 12 callbacks suppressed
… hostname kernel: mce: [Hardware Error]: Machine check events logged
… hostname kernel: mce: [Hardware Error]: Machine check events logged
… hostname kernel: mce_notify_irq: 13 callbacks suppressed
… hostname kernel: mce: [Hardware Error]: Machine check events logged
… hostname kernel: mce: [Hardware Error]: Machine check events logged
```

Odnosno u ekstremno slučaju možemo imati ovakve poruke koje uzrokuju *kernel panic* stanje:
```
Pid: 0, comm: swapper Tainted: G   M --- 5.14.0-284.11.1.el9_2.x86_64
Call Trace:
 <#MC>  [<ffffffff814ec341>] ? panic+0x78/0x143
...
```
← Za takozvni *tainted* kernel i **M** oznaku, pogledajmo njen opis:
*(M) →: processor reported a Machine Check Exception (MCE)*
U slučaju fatalnih grešaka sustav će se restartati, a prije restarta će se prema zadanim postavkama sustava kreirati datoteka **vmcore**.



Za više detalja o *Tainted kernel* porukama pogledajte (tablicu):
**11.1.1.6. Tainted kernel.**

Ovako izgleda poruka kod problema s **ECC RAM** memorijom:
```
Hardware event. This is not a software error.
Corrected error
Transaction: Memory scrubbing error
Memory ECC error occurred during scrub
...
```

Distribucije Linuxa bazirane na *Debian* distribuciji, od novijih inačica koriste noviji servis imena `rasdaemon` te pripadajuću naredbu: `ras-mc-ctl`. Ovaj servis i naredba dostupni su i za ***Red Hat* 8+** te ih se može instalirati sa:
```
yum install rasdaemon
```

Zatim ga možemo pokrenuti i permanentno aktivirati.
```
systemctl start rasdaemon
systemctl enable rasdaemon
```

***Pazite da ne koristite istovremeno rasdaemon i mcelog servis (ili jedan ili drugi).***

Servis *rasdaemon* i naredbu `ras-mc-ctl` nećemo detaljno opisati ovdje, iako on nudi i neke napredne mogućnosti detekcije hardverskih grešaka.



Naziv **RAS** dolazi od: *Reliability, Availability and Serviceability*.

Ako nas zanimaju detalji o *MSR* registrima, možemo instalirati softverski paket naziva `msr-tools`:
```
yum -y install msr-tools
```
Softverski paket koji smo upravo instalirali omogućuje nam čitanje registara (naredba `msr-cpuid`) odnosno naredba `rdmsr` omogućuje nam čitanje točno određenog registra. Međutim, ako nije učitan, potrebno je učitati i poseban kernel modul za MSR:
```
modprobe msr
```

Nakon toga pojavit će se posebne datoteke `/dev/cpu/{CPU_ID}/msr`.

Ako je i to potrebno, moguće je u ***GRUB boot*** menadžeru u opciji `mce=` uključiti ili isključiti *MCE* (ili noviji *LMCE*). Primjerice za isključivanje *LMCE* bi to bila opcija: `mce=no_lmce`. Odnosno ako želimo isključiti *MCE*, tada bi to bila opcija: `mce=off`.



Za više detalja *o GRUB boot menadžeru* i opcijama tijekom pokretanja računala pogledajte poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**
**11.1.1.5.1. Rad s GRUB-om prilikom pokretanja sustava (*rescue*).**

**Izvori informacija: (1502),(1503),(1504),(1505),(1506),(1507),(1508),(1509),(1510), `man mcelog`,`man rasdaemon`, `man ras-mc-ctl`,`man 4 msr`.**

# 10.8. AHCI i ACPI

## Advanced Configuration and Power Interface (ACPI)

Sučelje za naprednu konfiguraciju i napajanje odnosno *Advanced Configuration and Power Interface* (**ACPI**) definira otvoreni standard koji operativni sustavi mogu koristiti za otkrivanje i konfiguraciju hardverskih komponenti računala te za upravljanje napajanjem: poput stavljanja neiskorištenih hardverskih komponenti ili cijelog sustava u stanje mirovanja i slično.
***ACPI*** se koristi za automatsku konfiguraciju uređaja (hardvera), primjerice tzv. ***Plug and Play*** ili tijekom zamjene uređaja u radu (tzv. ***Hot swapping)***, te za praćenje statusa rada hardvera. ***ACPI*** je prvi put objavljen u prosincu 1996. godine, a imao je za cilj zamijeniti: *Advanced Power Management* (**APM**), *MultiProcessor Specification*, ***PCI BIOS*** specifikaciju te *Plug and Play BIOS* (**PnP**) specifikacije.



**ACPI** otkrivanje i konfiguraciju hardvera te upravljanje napajanjem, stavlja pod kontrolu operativnog sustava, za razliku od prethodnog sustava kod kojeg je tu zadaću odrađivao **BIOS** koji se oslanjao na svoj *firmware*. Dakle ***ACPI*** definira sučelje za apstrakciju hardvera između *firmwarea* uređaja (primjerice ***BIOS*** ili **UEFI**) to jest komponenti računalnog hardvera i operativnog sustava. To znači kako se **ACPI** komponenta (softver) nalazi unutar operativnog sustava i ona je zadužena za komunikaciju s **ACPI BIOS**-om. Čak se i primjerice slanje signala za gašenje računala (ili virtualnog računala) ili stavljanje sustava u stanje mirovanja, odrađuje slanjem posebnih **ACPI** poziva prema operativnom sustavu to jest **ACPI** komponenti.

Interno, *ACPI* oglašava dostupne komponente računala i njihove funkcije kernelu operativnog sustava, koristeći popise uputa koje se dostavljaju putem *firmware-a* sustava (**UEFI** ili **BIOS**), koje kernel potom analizira. *ACPI* zatim izvršava željene operacije napisane u *ACPI* strojnom kôdu, pomoću ugrađenog minimalnog virtualnog stroja. Kada se ***ACPI*** sustav pokreće, prije nego što se operativni sustav učitao, ***ACPI BIOS*** postavlja početne **ACPI tablice**** u memoriju, koje kasnije kernel linuxa može koristiti. ***ACPI BIOS**** je mali BIOS koji izvodi osnovne operacije upravljanja na niskoj razini, na hardveru.
Te operacije uključuju kôdove za pomoć pri pokretanju sustava i za stavljanje sustava u stanje mirovanja ili buđenja. Imajte na umu da je **ACPI BIOS** puno manji od **APM BIOS**-a, jer se većina upravljačkih funkcija preselila u operativni sustav i **ACPI** tablice. **ACPI tablice**** su središnja struktura podataka sustava koji se temelji na *ACPI*-ju.

One sadrže blokove definicija koji uključuju i podatke i strojno neovisan bajt kôd koji se koristi za izvođenje operacija upravljanja hardverom. **ACPI** tablice pod Linuxom možemo pronaći u strukturi direktorija: `/sys/firmware/acpi/tables/`. Ovdje se nalazi po jedna datoteka za svaki pojedini objekt to jest funkcionalnost. Pogledajmo opise samo nekih od ovih datoteka:

- `APIC` – tablica vezana za **APIC** (*Interrupt* kontroler).
- `BERT` – tablica s greškama pri pokretanju računala.
- `BDAT` – *BIOS Data ACPI Table* (tablica s ACPI podacima).
- `DSDT` – *Differentiated System Description Table.*
- `DMAR` – *DMA remapping unit* i `EINJ` – *Error Injection* tablica.
- `FACP` – fiksne informacije o ACPI hardveru.
- `FACS` – *Firmware ACPI Control Structure* (kontrolna struktura).
- `HEST` – *Hardware Error Source Table*
- `FPDT` – *Firmware Performance Data Table* (sadrži informacije o inicijalizaciji *firmware*-a i vremenu inicijalizacije).
- `HPET` – *High Precision Event Timer* i `MCFG` – *PCI Express Memory-mapped Configuration*
- `MSCT` – informacije o hardverskoj topologiji računala.
- `SLIT` – informacije o latenciji memorije na NUMA sustavu.
- `SPMI` – Server **IPMI** informacije.
- `SRAT` –informacije o memoriji i CPU jezgrama na *NUMA* sustavima.
- `SSDT` (**1**, **2**, **3**, **4**) – proširenje je na `DSDT` s podacima o uređajima.
- `UEFI` – sadrži **UEFI** postavke.

Da bi **ACPI** u potpunosti radio na računalu, a pogotovo na virtualnom računalu, to jest da bi se virtualno računalo moglo uredno ugasiti ili staviti u stanje mirovanja [*snapshot/suspend*], potrebno je na Linuxu imati instaliran i pokrenut `acpid` servis:

```
yum -y install acpid
systemctl start acpid
systemctl enable acpid
```

Sve druge komponente **ACPI** sustava su obično već ugrađene u kernel (jedna po jedna), a može ih se pregledati s naredbom:

```
grep -i acpi /boot/config-`uname -r`
```

## Advanced Host Controller Interface (AHCI)

*Advanced Host Controller Interface* (**AHCI**) tehnički je standard definiran od strane tvrtke *Intel*, koji specificira rad serijskih ATA (**SATA**) diskovnih kontrolera na način koji nije specifičan za implementaciju u *čipsetima* matične ploče. Specifikacija opisuje strukturu sistemske memorije za razmjenu podataka između memorije računala i priključenih uređaja za pohranu podataka (pr. diskova).

***AHCI*** daje programerima softvera i dizajnerima hardvera standardnu metodu za otkrivanje, konfiguriranje i programiranje **SATA/AHCI** adaptera. Iako je **AHCI** odvojen od **SATA 3** standarda, on omogućuje napredne **SATA** mogućnosti tako da ih računalo bezbolno može koristiti, što vrijedi i za virtualna računala. Neke od ***AHCI*** značajki su:

- Zamjena diskova u radu (engl. *hot swap*).
- **NCQ** mehanizam redova čekanja/izvršavanja SATA naredbi. *Native Command Queuing* (***NCQ***) tvrdim diskovima omogućuje da interno optimiziraju redoslijed kojim se izvršavaju primljene naredbe za čitanje i pisanje. To može smanjiti količinu nepotrebnog pomicanja glave diskovnog pogona, što rezultira povećanjem performansi diskovnog sustava.



Za više detalja pogledajte poglavlje:
**13.2.1.7. Native Command Queuing (NCQ).**

- **TRIM** funkcionalnost. Naredba TRIM koja je poznata kao TRIM u skupu ATA naredbi odnosno UNMAP u skupu naredbi SCSI, omogućuje operativnom sustavu da obavijesti *solid-state* disk (**SSD**) koji se blokovi podataka više ne smatraju "korištenima" i stoga se mogu interno brisati. *Trim* je predstavljen ubrzo nakon što su predstavljeni SSD diskovi. Budući da se na najnižoj razini rada SSD diskovi značajno razlikuje od mehaničkih tvrdih diskova, tipičan način na koji operativni sustavi rukuju operacijama poput brisanja i formatiranja rezultirao je neočekivanim progresivnim smanjenjem performansi operacija pisanja na SSD diskove. TRIM u konačnici omogućuje SSD-u da učinkovitije upravlja brisanjem podataka što bi inače usporilo buduće operacije pisanja u uključene blokove.



Za više detalja pogledajte poglavlja:
**13.3. SSD diskovi.**
**13.3.1.3. TRIM funkcionalnost.**



Važno je napomenuti da u **BIOS**-u računala kod odabira parametara rada diskova (pr. ***Storage config*** sekcija) trebamo odabrati načina rada diska. Uobičajeni izbori su: **ATA (IDE)** ili **AHCI**, a ponekad i **RAID**.

***AHCI*** je standardno kompiliran u Linux kernelu (*obično kao kernel moduli*), što možete provjeriti sa sljedećim nîzom naredbi:

```
grep -i ahci /boot/config-`uname -r`
```

Nakon inicijalizacije sustava *AHCI* će biti vidljiv ovako:

```
dmesg | grep -i ahci
ahci 0000:00:1f.2: version 3.0
ahci 0000:00:1f.2: AHCI 0001.0300 32 slots 6 ports 6 Gbps 0x3 impl SATA mode
ahci 0000:00:1f.2: flags: 64bit ncq pm led clo pio slum part ems apst
scsi host0: ahci
```

Ako pogledamo oznaku uređaja na **PCI** sabirnici, koji smo sada vidjeli, odnosno oznaku `00:1f.2` vidjet ćemo sljedeće:

```
lspci -s 00:1f.2 -vv
00:1f.2 SATA controller: Intel Corporation 8 Series/C220 Series Chipset Family 6-port
SATA Controller 1 [AHCI mode] (rev 04) (prog-if 01 [AHCI 1.0])
Kernel driver in use: ahci
```

← Iako smo ispis drastično skratili, vidljivo je da je PCI oznaka `00:1f.2`, uređaja oznaka SATA disk kontrolera, za koji vidimo da radi u *AHCI* načinu rada `[AHCI mode]` te da za rad koristi kernel modul imena `ahci`. Naime bazni kernel modul s podrškom za ***AHCI*** je kernel modul imena: `ahci` dok je za njegove rutine na najnižoj razini zadužen kernel modul imena: `libahci`.

Pogledajmo jesu li učitani ovi kernel moduli, s naredbom: `lsmod`:

```
lsmod |grep -i ahci
ahci                   49152  3
libahci                61440  1 ahci
```

← Svi navedeni kernel moduli su učitani, što znači da je naš sustav sposoban koristiti *AHCI* značajke.

Za moderne **SSD** pogone (primarno **NVMe**), **AHCI** sučelje je zamijenio **NVMe** standard.



Za više detalja o **NVMe** diskovima pogledajte poglavlje:
**13.3.2. NVMe diskovi.**

**Izvori informacija (998),(999),(1071),(1072),(1073),(1074),(1075),(1076),(K-8) te poglavlja: 13.3, 13.3.1.3 i 27.1.3.**

# 11. Od kojih komponenti se sastoji Linux

U ovom poglavlju i sljedećim poglavljima upoznat ćemo se s važnijim dijelovima svakog Unix/Linux sustava, a to su dijelovi koje ima i bilo koji drugi operativni sustav (primjerice ***Windows***, ***MacOS***, ***FreeBSD***, ***iOS***, ***Android***, …).

**Što je uopće operativni sustav?**
Operativni sustav (OS) je sistemski softver koji upravlja računalnim hardverom, softverskim resursima i pruža zajedničke usluge za računalne programe. Prisjetimo se da se svako računalo sastoji od centralnog procesora (CPU), radne memorije (RAM), diskovne memorije odnosno tvrdih (SSD) i drugih diskova (pr. CD/DVD/BD-ROM, USB), grafičke, zvučne mrežne i drugih kartica, kao i svih drugih ulazno-izlaznih komponenti. Zadaća operativnog sustava je upravljanje sa svim tim sklopovljem (hardverom) odnosno njegovim resursima. Operativni sustav za pristup hardverskim funkcijama poput ulazno-izlaznih operacija, dodjelu memorije, pristup disku ili mreži i drugima, djeluje kao posrednik između programa i računalnog hardvera. Dakle bez operativnog sustava ne bi mogli instalirati niti pokretati naše programe (aplikacije).
Povijesno gledano, prvi operativni sustavi su mogli pokretati (izvršavati) samo jedan program. Tek kada bi se taj program završio, mogli smo pokrenuti sljedeći. Ovakvi sustavi se nazivaju sustavima koji su u stanju izvršavati samo jednu zadaću (Eng. *Single task*). Razvojem prvih ***Unix*** operativnih sustava, krajem 1960-ih, pojavili su se moderniji operativni sustavi koji su mogli izvršavati više programa istovremeno, a nazivaju se *višezadaćnim* sustavima (engl. *Multi-tasking*).
To se postiže dijeljenjem vremena, gdje se raspoloživo vrijeme procesora dijeli između više programa (tzv. *procesa*).

Svaki od tih procesa višestruko se prekida u vremenskim odsječcima, od strane podsustava za planiranje zadataka operativnog sustava (engl. *Task manager*). Više zadataka se obično izvršava u tzv. preventivnom *višezadaćnom* radu. U preventivnom *višezadaćnom* radu (engl. *Preemptive multitasking*), operativni sustav svaki program pokreće samo u jednom djeliću odnosno odsječku vremena, a potom se prebacuje na izvršavanje drugog programa, u njegovom odsječku vremena za izvršavanje i tako dalje. Dakle svaki program se izvršava samo djelić vremena, a pošto se sustav ekstremnom brzinom prebacuje između pokrenutih programa, mi dobivamo dojam da se programi izvršavaju istovremeno odnosno paralelno.
Druga važna značajka modernih operativnih sustava je višekorisnički rad, što znači da nam ovakvi sustavi omogućavaju rad više korisnika istovremeno. Ovakav rad se naziva engl. *multi user* način rada. U ovakvom radu se praktično proširuje ideja i način *višezadaćnog* rada jer sada svaki korisnik u svom izoliranom okruženju može pokretati svoje programe, i to više programa istovremeno. Kako bi to uopće radilo, prošireni su osnovni mehanizmi *višezadaćnog* rada sustava.



Za više detalja o mehanizmima zaduženim za višezadaćnost, pogledajte poglavlja:
**9. Proces menadžment.**
**9.3. Task/process scheduler.**

Pošto je operativni sustav zadužen za upravljanje sklopovljem (hardverom), jedna od njegovih važnih komponenti je i ona zadužena za rad sa signalima prekida (engl. Interrupts [***IRQ***]). Naime kada neka komponenta računala treba pristup centralnom procesoru, ona ga o tome obavještava slanjem signala prekida. Primjerice kada mrežna kartica zaprimi mrežni paket, ona tada preko signala prekida signalizira centralnom procesoru, da mu je potrebna njegova pomoć. Ona konkretno traži od njega da prihvati mrežni paket, obradi ga i pošalje ga na aplikaciju (program) koji očekuje podatke s mreže. Za to je centralnom procesoru potreban i pristup radnoj (RAM) memoriji, ali i privremeno pauziranje programa koji se trenutno izvršava.
Nakon što je procesor pauzirao trenutni program, on pokreće posebnu rutinu za obradu tog signala prekida i u konačnici se obrađuje zaprimljeni mrežni paket, a podaci iz njega se prosljeđuju programu koji ih očekuje. Potom sustav ponovno pokreće pauzirani program i nastavlja s radom i daljim prebacivanjem na izvršavanje drugih programa (*višezadaćni* rad).



Za više detalja o (hardverskim) signalima prekida (**IRQ**), pogledajte poglavlje:
**10.5.1. IRQ (*Interrupt request*).**

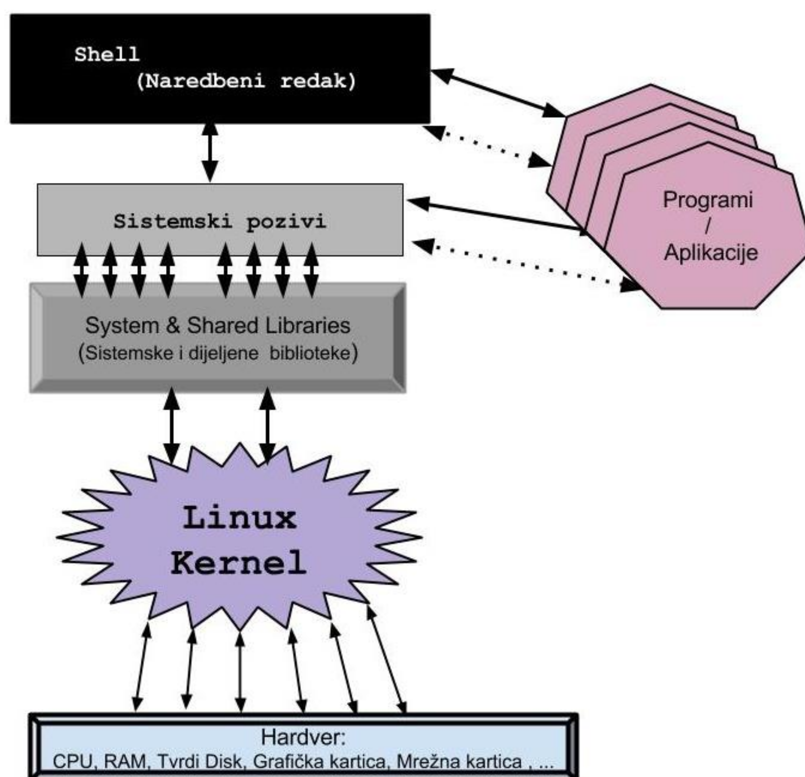
Sljedeća važna zadaća operativnog sustava je rad s radnom memorijom (RAM) koji se odvija preko sustava virtualne memorije.
Naime kako bismo uopće mogli pokretati programe s nekog diskovnog uređaja (tvrdi, SSD, USB ili neki drugi disk), oni se prvo moraju učitati u radnu memoriju računala. Ova memorija je ekstremno važna za rad programa, jer osim što se u nju učitavaju programi, u nju se zapisuju i međurezultati obrade programa, kao i drugi podaci važni za njihov rad.
Tijekom pristupa diskovnom sustavu, operativni sustav omogućava programima rad s diskovima, zbog čitanja ili zapisivanja na njih, što nam omogućava posebna komponenta zadužena za rad s diskovnim sustavom. Naime pomoću ove komponente i učitavamo programe s diska. Osim toga, preko nje naši (i drugi) programi podatke mogu i zapisivati na disk ili pristupati nekim drugim podacima (datotekama) s njega.
I na kraju, operativni sustav je taj koji se brine o cijelom životnom vijeku naših programa, kao i o interakciji među programima, ali i s nama, u slučajevima kada moramo unositi neke podatke u programe ili jednostavno raditi s njima. Naša interakcija s programima može osim ručnog unosa podataka, biti i prema diskovnom sustavu (učitavanje ili snimanje datoteka), mrežnom sustavu (slanje ili primanje podataka s mreže), ili prema nekim drugim vanjskim komponentama računala (pr. ispis na pisač).

Linux, kao i bilo koji drugi operativni sustav, sastoje se od sljedećih dijelova odnosno komponenti:

- **Kernela** odnosno jezgre operativnog sustava, koji sadrži:
  - Podsustav za pokretanje operativnog sustava (engl. *boot code*).
  - Upravljačke programe (engl. *device drivers*) za sav hardver.
  - Memorijski management za upravljanje, rad i pristup virtualnoj i fizičkoj memoriji.
  - Menadžment za baratanje s datotekama i direktorijima, mrežnim i sigurnosnim sustavom.
  - Kontrolne i upravljačke mehanizme potrebne za pokretanje i rad svih aplikacija (tzv. *proces/task scheduler*), mehanizme za komunikaciju između programa i mnoge druge mehanizme i komponente.
- **Razvojnog okruženja** koje se sastoji od:
  - Jezičnog odnosno programskog prevoditelja (engl. *Compiler*), povezivača (engl. *Linker*), …
  - Sistemskih biblioteka (engl. *System library*).
  - Programa za pronalaženje pogrešaka (engl. *Debugger*) i drugih pomoćnih programa.
- **Korisničkog sučelja (***engl. User interface***)** koje obično sadrži**:**
  - Naredbeni redak odnosno *ljusku* (engl. *Shell*) i/ili grafičko sučelje (takozvani *X Window* sustav).
- **Ostalih programa** poput servisnih programa sustava, ali i mnogih kategorija korisničkih programa te nîza programa i dokumentacije za iste.

Prema svemu navedenom, logička shema svakog Unix/Linux operativnog sustava je prikazana na slici 77**.**

**Slika 77. Pogled na komponente Linuxa**




Ako krenemo od vrha slike, vidljivo je kako naredbeni redak (engl. *Shell*) kao i svi programi komuniciraju s linux kernelom, preko sistemskih poziva. Kernel dalje preko svojih upravljačkih programa pristupa svôm hardveru: disk kontrolerima i diskovima, grafičkoj, mrežnoj i drugim karticama te drugim komponentama računala. Dakle svaki program, pa tako i *shell,* odnosno naredbeni redak ili ljuska, preko sistemskih poziva šalju određene naredbe sâmom kernelu. U trenutku kada primjerice iz naše ljuske želimo pokrenuti neku naredbu, šalju se sistemski pozivi za pokretanje te naredbe odnosno programa. Kada se naredba pokrene (od strane kernela) i mora nešto zapisivati na disk, ponovno se šalju sistemski pozivi za pristup disku, i tako dalje…
U cjelinama koje slijede upoznat ćemo se s nekoliko važnih komponenti Linuxa.



Linux koristi takozvani **monolitni** kernel čija osnovna značajka je da su sve navedene komponente: proces menadžer, sustav virtualne memorije, mrežna, diskovna i druge komponente, kao i upravljački programi, integrirane u sâm kernel.
To ga čini prilično velikim; primjerice Linux kernel iz 2020.g. imao je 27.8 milijuna linija (redaka) programskog kôda.



Za više detalja o kernelu pogledajte sljedeće poglavlje, ali i poglavlje:
**2.7.1. Što je operativni sustav.**

**Izvori informacija: (88),(967),(968),(969),(1114),(K-9)**,`man 7 sched`.

# 11.1. Kernel

Linux kernel je, kako mu i ime govori (engl. *kernel*=jezgra), sâma jezgra operativnog sustava Linux, koja je zadužena za:

- Pristup, kontrolu i upravljanje hardverom: od upotrebe centralnog procesora (CPU), memorije (RAM), *chipseta i sabirnica(e)*, do upravljanja signalima prekidima (*IRQ*) i direktnim pristupom memoriji (*DMA*) te drugim elementima ključnim za rad sveg hardvera unutar računala.
- Upravljanje i korištenje upravljačkih programa to jest ***kernel modula*** (tako se u Linuxu zovu upravljački programi).
- Upravljanje i rad s memorijom (RAM) te u konačnici radom sa sustavom virtualne memorije.
- Pokretanje, kontrolu i rad cijelog operativnog sustava, što obuhvaća:
  - Pokretanje, upravljanje, rad, ali i zaustavljanje svih programa/aplikacija (procesa).
  - Upravljanje vremenskim okvirima za izvršavanje svih programa/aplikacija (procesa):
    - Prebacivanje između pokrenutih procesa/programa ili programskih nîti unutar pojedinog programa, za što je zadužen tzv. *context switch* mehanizam.
- Komunikaciju između programa (procesa) unutar jednog računala: ovdje se ne misli na mrežnu komunikaciju između programa.
- Upravljanje i rad s datotečnim sustavom te upravljanje i rad s mrežom i mrežnim sustavom.
- Sigurnost, kontrolu prava pristupa i druge sigurnosne radnje sustava.

Jezgru linuxa čini njegov ***kernel***, kao i nîz sistemskih i drugih programa potrebnih za njegov rad. Linux kernel osim klasičnih računala koriste i razni uređaji poput usmjerivača, uređaja za pohranu podataka (engl. *NAS* ili *SAN*), vatrozida (engl. *Firewalls*) kao i drugi specijalizirani uređaji, poput *pametnih* mobitela ili satova kao i ***IoT***, ali i drugih uređaja.
Oznaka generacije odnosno vršne inačice kernela (kao i softvera) se naziva engl. ***Major*** i odnosi se na značajne promjene sustava (pr. promjene načina rada, ***API*** poziva i sl.). Oznake pôd inačice kernela (i softvera) se nazivaju *minorne* engl. ***Minor***, te se odnose na nadogradnje ili nove funkcionalnosti, ali kompatibilne unutar vršne inačice.
Za kernel i softver imamo i još jednu oznaku koja označava tzv. inačicu zakrpe (engl. ***Patch***), pa u konačnici možemo imati primjerice sljedeću inačicu tj. naziv: ***kernel 4.18.0-240***. Pri tome je **4** oznaka *Major* inačice, **18** je oznaka za *Minor* a **0-240** je oznaka za razinu zakrpi.



Pogledajte i uvodno poglavlje o razvoju distribucija Linuxa:
**2.7. Povijest otvorenog koda (*Open Sourcea)* i Linuxa.**

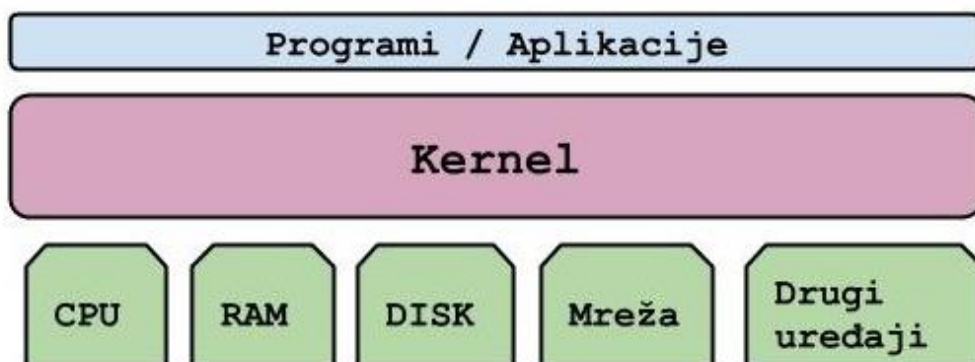
Linux kernel je inicijalno razvio student računalnih znanosti, *Helsinškog* Sveučilišta iz *Finske*: ***Linus Tovalds*** u toku 1991 godine, za svoje osobno računalo. Ovo je vijest koju je 25.8.1991 ***Linus Torvalds*** poslao na *News* grupu: "***comp.os.minix***":

```
I'm doing a (free) operating system
(just a hobby, won't be big and professional like gnu) for 386(486) AT clones.
This has been brewing since April, and is starting to get ready. I'd like any feedback
on things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).
```

Vrlo brzo nakon ove vijesti dodani su i razni sistemski i pomoćni programi iz GNU projekta, te je nastao operativni sustav Linux. Već u 12.mj. 1991 izašla je prva inačica Linuxa: 0.11. Ubrzo nakon što je javno objavljen izvorni kôd Linux kernela, kernel je prihvaćala sve veća i veća zajednica ljudi te se počeo razvijati i za druge arhitekture procesora: ***ARM***, ***MIPS***, ***SPARC***, ***RISC-V i*** druge. Danas ga aktivno razvija zajednica od nekih 12.000+ aktivnih programera, a od toga je aktivno i oko 1.200 tvrtki: od proizvođača hardvera (***Intel, AMD, Broadcom, Samsung***, …) do softverskih divova (***RedHat***, ***Google***, ***IBM***, ***Facebook***, …).

Slika 78. prikazuje osnovnu logičku shemu rada Linuxa:

**Slika 78. Logička shema rada Linuxa.**



Linux kernelu se pristupa preko kernel **API** poziva (engl. *Application Programming Interface*) preko kojih aplikacije imaju interakciju s kernelom. To se konkretno odrađuje preko takozvanih sistemskih poziva (engl. *syscall*). Sistemski pozivi su procedure koje osiguravaju sučelje između programa (procesa) i operativnog sustava. Naime, to je način na koji računalni program traži uslugu od kernela (jezgre) operativnog sustava. Kada vaš program želi pisati ili čitati iz datoteke, osluškivati ili ostvariti mrežnu vezu, izbrisati ili stvoriti direktorij ili čak završiti svoj rad, program koristi sistemske pozive. Sistemski pozivi općenito se ne pozivaju izravno, već putem funkcija, obično u ***glibc*** ili eventualno nekoj drugoj biblioteci. Dakle sistemski pozivi su funkcije *kernel prostora* koju programi *korisničkog prostora* pozivaju da se obradi neki zahtjev prema sustavu. Linux kernel pruža skup ovih funkcija, a svaka arhitektura računala odnosno procesora (pr. ***x86***,***x86-64***, ***ARM***, ***RISC-V***, ...) nudi svoj skup funkcija. Na primjer x86_64 arhitektura pruža 322 sistemska poziva, a x86 arhitektura pruža 358 različitih sistemskih poziva. Sistemski pozivi su pri tome podijeljeni o nekoliko kategorija:

- ***Process control*** – ova kategorija sistemskih poziva zadužena je za pokretanje, terminaciju i sl., procesa/programa.
- ***File management*** – ova kategorija sistemskih poziva zadužena je za manipulaciju datotekama i direktorijima (pr. kreiranje, čitanje ili zapisivanje).
- ***Device management*** – ova kategorija sistemskih poziva zadužena je za čitanje ili zapisivanje prema/iz uređaja.
- ***Information maintenance*** – ova kategorija sistemskih poziva barata s informacijama (pr. alarmi, dohvaćanje informacija o PID brojevima i drugo).
- ***Communication*** – ova kategorija sistemskih poziva namijenjena je komunikaciji među programima (procesima).

S druge strane i kernel i kernel moduli mogu slati signale korisničkim programima (o tim signalima nešto kasnije).
U svojoj osnovnoj funkcionalnosti Linux kernel uključuje i upravljačke programe za raznovrstan hardver:

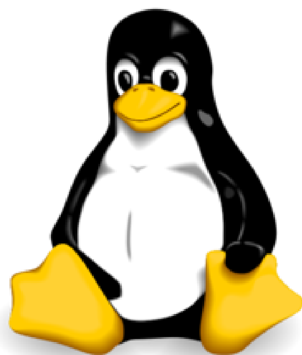
- Od matične ploče i svih njenih dijelova (CPU, memorijski kontroler, sabirnice, …).
- Disk kontrolera, mrežnih i grafičkih kartica te zvučnih i drugih kartica.

Upravljački programi, u Linux terminologiji znani su kao ***kernel moduli***, koji mogu biti ugrađeni u kernel koji koristimo i/ili mogu biti dostupni kao zasebni upravljački programi, koje možemo pozivati odnosno učitavati prema potrebi. Tada govorimo o takozvanim kernel modulima koji se mogu učitavati odnosno o tzv. ***LKMs*** (engl. *Loadable Kernel Modules (LKMs)*).

***Važno je razumjeti da upravljački programi (kernel moduli) u pravilu dolaze s kernelom.***

Vezano za kernel module, oni ne moraju isključivo sâmo imati funkcionalnost upravljačkih programa, već mogu imati bilo koju drugu funkcionalnost, poput funkcionalnosti vatrozida, nekog dodatnog mrežnog protokola ili funkcionalnosti, funkcionalnosti (dijela) datotečnog sustava i slično. Službena maskota Linuxa je pingvin ***Tux***, koji je vidljiv na slici **79**.

**Slika 79. Pingvin TUX**



**Pogledajmo Linux kernel malo detaljnije.**

Kernel se logički nalazi između aplikacija i hardverskih resursa: CPU, RAM, tvrdi disk, mreža, … . On je stoga zadužen za rad cijelog operativnog sustava i svih njegovih procesa odnosno pokrenutih programa/aplikacija. Sâm kernel se sastoji od nekoliko logičkih cjelina, koje ćemo sada sâmo spomenuti, a detaljnije ćemo ih upoznati kasnije u radu.

**Kernel se sastoji od sljedećih cjelina:**

- **`Memory Manager`** odnosno memorijski menadžment je zadužen za sve operacije koje se tiču rada sa sustavom virtualne memorije. Dakle svaki pristup memoriji: radilo se o čitanju ili zapisivanju u RAM memoriju ili *Swap* disku kao proširenju RAM memorije, prolazi kroz ovu komponentu. Ova komponenta je u konačnici zadužena za pristup i korištenje RAM memorije. Dodatno i sâmo dijeljenje memorije između procesa odnosno programa, uz sve zaštitne mehanizme, također prolazi kroz ovu komponentu. Više o memorijskom menadžeru i virtualnoj memoriji pogledajte u poglavlju: **12.Sustav virtualne memorije i memorijski menadžment**.

- **`Inter Process Manager`** je zapravo logički najviše povezan sa **`Memory managerom`**, a on je konkretno zadužen za komunikaciju između procesa (programa), kao i dio funkcionalnosti dijeljenja memorije između procesa. Osim toga njemu pristupa i **`Proces Management & Scheduler`** za kontrolu i upravljanje procesima.
Za više detalja o komunikaciji između procesa/programa, pogledajte poglavlje: **9.4 Komunikacija između procesa.**

- **`Proces Management & Scheduler`** je zadužen za cijeli životni ciklus svakog procesa. Proces management znan i kao "*proces scheduler*" odnosno "*task scheduler*" pokreće svaki program, brine se o njegovom prioritetu i nîtima unutar procesa (engl. *Thread*). Dakle on prati i upravlja svakim procesom (programom) od početka do kraja njegovog životnog vijeka. Za više detalja pogledajte cijelo poglavlje**: 9. Proces menadžment.** S obzirom na njegovu funkcionalnost, on komunicira s procesorom (*CPU*om) i njegovim registrima, kao i s radnom (RAM) memorijom.

- **`Virtual Filesystem (VFS)`** komponenta je jednim dijelom zadužena za pristup sustavu virtualne memorije, pa je povezana i isprepletena s **`Memory managerom`**. Međutim njena primarna unutarnja komponenta je zadužena za komunikaciju s diskovnim podsustavom te s datotečnim sustavom. Naime ona kroz ulazno/izlazni (**I/O**) sustav komunicira (na slici prema dolje) preko upravljačkih programa, s disk kontrolerima te u konačnici s tvrdim diskovima. Za više detalja o ovoj komponenti pogledajte poglavlje: **14. Diskovni (I/O) podsustav**.

- **`I/O Interface`** je zadužen za ulazno izlazni pristup (engl. *I/O*) raznim uređajima, bilo fizičkim ili logičkim; primjerice: disk kontroler ili disk, blok uređaji, "*character*" uređaji, … .

- **`Virtual Network`** komponenta je zadužena za rad s mrežom odnosno za svu mrežnu komunikaciju preko ulazno/izlaznog (**I/O**) sustava mreže, sve do upravljačkih programa za mrežne kartice ili druga mrežna sučelja.

Sve navedene komponente imaju komunikaciju s vanjskim svijetom s jedne strane, preko sistemskih poziva prema procesima/programima (aplikacijama), a s druge strane direktnu vezu s hardverom, preko upravljačkih programa odnosno takozvanih Linux *kernel modula*. Za detalje o sistemskim pozivima pogledajte poglavlje: **9.2.2. Signali i sistemski pozivi.**

Pogledajmo logičku shemu linux kernela i njegovih osnovnih komponenti (slika 80.).

**Slika 80. Pogled na logičku shemu Linux kernela.**




Sada ćemo se fokusirati na donji dio slike s prethodne stranice, odnosno na dio s kernel modulima odnosno *upravljačkim programima*.

**Slika 81. Pogled na logičku shemu dijela Linux kernela s detaljima vezanim za kernel module (upravljačke programe).**

Na slici s prethodne stranice, vidljivo je kako su u grubo kernel moduli podijeljeni u dvije kategorije:

- Oni koji predstavljaju upravljačke programe za određeni hardver ili hardverske komponente.
- Oni koji predstavljaju neke dodatne funkcionalnosti (značajke) sustava.

Za one kernel module koji su zapravo upravljački programi za određeni hardver, imamo više kategorija, odnosno po jednu kategoriju za svaki pojedini uređaj. Tako da imamo (skratili smo listu i sliku zbog lakšeg razumijevanja):

- **`CPU`** - podrška za procesor (*CPU*) i većinu njegovih komponenti se već nalazi unutar kernela, dok su upravljački programi za neke komponente CPUa, dostupne kao vanjski kernel moduli, poput ovih koje smo naveli:
  - `procesor` - daje nam **ACPI** (*Advanced Configuration and Power Interface*) funkcionalnosti CPUa.
  - `aesni-intel` - daje nam (za *Intel* CPU) hardversku akceleraciju ***AES*** kriptografskih funkcija, koje su ugrađene u sâm procesor (ako to konkretan model procesora podržava).
  - `rng-core` - daje nam hardversku akceleraciju (ubrzanje) funkcija generiranja slučajnih brojeva, koja je ugrađena u procesor (ako to konkretan model procesora podržava).

- **`Chipset`** - ovdje se nalaze razne komponente *chipseta*, poput:
  - `pcieport` - upravljački program za **PCI Express** sabirnicu, ovaj modul je često unutar sâmog kernela.
  - `uhci_hcd` i `ehci_hcd` - ovo su upravljački programi za **USB** kontrolere.
  - `agpgart-intel` - ovo je upravljački program za **AGP** sabirnicu.
  - `i801_smbus` - ovo je upravljački program za **SMBUS** (*System Management Bus*) sabirnicu.

- **`Disk kontroleri`** - ovdje se nalaze upravljački programi za razne disk kontrolere (ATA/SATA/SCSI/SAS …):
  - `ata_piix` - ovo je upravljački program za (P)ATA i SATA disk kontrolere (primjerice za *Intel*: PIIX3/4, ICH/2/3/4/5/6/7/8/9/10, …).
  - `hpsa` je upravljački program za *Smart Array* RAID kontrolere (P212/410/411/700M/712m/…) tvrtke **HP**.
  - `megaraid_sas` je upravljački program za *Mega RAID SAS* RAID kontrolere (1078/2108/2008/2208/…) tvrtke **LSI**.

- **`Mrežne kartice`** - u ovom dijelu su upravljački programi za razne mrežne kartice:
  - `e1000e` - ovo je upravljački program za 1Gbps mrežne kartice (82571, 82572, 82573, 82574, 82583, ICH8, ICH9, ICH10, PCH, PCH2, I217, I218, I21) tvrtke **Intel**.
  - `tg3` - ovo je upravljački program za 1Gbps mrežne kartice (BCM 57xx) tvrtke **Broadcom**.
  - `igb` - ovo je upravljački program za 1Gbps mrežne kartice (82575, 82576, 82580, I210, I211, I350, I354, DH89xx) tvrtke **Intel**.

Druga kategorija su kernel moduli koji nisu upravljački programi, a daju nam nove funkcionalnosti.
Pogledajmo nekoliko njih, prema logičkim kategorijama kojima pripadaju:

- **`Diskovni podsustav`** - u ovoj kategoriji su kernel moduli koji su zaduženi za određene funkcionalnosti diskovnog podsustava. Za detalje o diskovnom podsustavu, pogledajte poglavlje: **14. Diskovni (I/O) podsustav** i to cjelinu: "**Dolazimo do nižih slojeva**". Ukratko u priči o kernel modulima i diskovnom podsustavu, kernel moduli koji su vidljivi ovdje, brinu se o cjelini koja je logički iznad upravljačkih programa. To znači da se brinu o:
  - `libata` - ovo je kernel modul zadužen za pristup *ATA* funkcionalnostima i vrsti disk kontrolera, preko *ATA* upravljačkih programa (kernel modula).
  - `sd` - ovo je vršni kernel modul koji je zadužen za pristup *SCSI* funkcionalnostima, preko *SAS/SATA/SCSI* upravljačkih programa (kernel modula). U tu kategoriju pripadaju i *SAS* i *SATA* disk kontroleri odnosno u konačnici *SAS* ili *SATA* diskovi.
  - `sg` - ovo je također *SCSI* među sloj za komunikaciju pomoću *SCSI* naredbi, prema i od diskovnog sustava.
- **`Datotečni sustavi`** - u ovoj kategoriji su kernel moduli koji su odgovorni za određene funkcionalnosti datotečnih sustava. Naime dio funkcionalnosti datotečnih sustava, koji zbog brzine rada trebaju biti što bliže kernelu, izrađuje se kao kernel modul, dok se drugi dio funkcionalnosti odrađuje na višoj razini operativnog sustava.
  Ovdje nalazimo kernel module poput:
  - `ext3` ili `ext4` - ovo su standardni Linux datotečni sustavi ***Ext3*** i ***Ext4*** odnosno dio njihovih funkcionalnosti.
  - `xfs` - ovo je dio funkcionalnosti ***XFS*** datotečnog sustava, kao što je primjerice `zfs` dio funkcionalnosti ***ZFS*** datotečnog sustava.
  - `cifs` i `nfs` su funkcionalnosti za mrežne datotečne sustave, redom za *Windows* ***SMB/CIFS*** (***cifs***) protokol te je za *Unix/Linux* to ***NFS*** protokol (***nfs***).
- **`Mrežni protokoli i funkcionalnosti`** - ovdje imamo kernel module koji odrađuju funkcionalnosti raznih mrežnih protokola ili posebnih mrežnih funkcija, poput:
  - `8021q` - ovaj kernel modul odrađuje funkcionalnosti potrebne za rad sa standardom **802.1Q**, odnosno za rad s Virtualnim LAN mrežama (***VLAN***). Pogledajte poglavlje: **20.6.2.**
  - `bonding` - ovo je kernel modul koji odrađuje funkcionalnost agregiranja mrežnih kartica, koje je još poznato pod imenima: *Teaming*, *Bonding* ili *Link aggregation*, a koji podržava nekoliko načina rada odnosno protokola (pogledajte poglavlje**: 20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).):**
    - *Active-backup (1), Balance-xor (2),*
    - *Broadcast (3), 802.3ad* (**LACP**) (4),
    - *Balance-tlb (5) i Balance-alb (6)*
  - `bridge` - kernel modul je zadužen za *bridge* funkcionalnosti (OSI Sloj 2 odnosno *Layer 2*).
    **Pogledajte poglavlje: 20.6.1.**
  - `ip_tables` – on je centralni kernel modul zadužen za *linux* ***iptables*** *vatrozid (firewall)* mogućnosti, a koji za dodatne funkcionalnosti treba i druge kernel module.

- Neki od modula koje koristi modul `ip_tables` su:
    - `ip_table_nat` - zadužen je za ***NAT*** funkcionalnost.
    - `nf_conntrack` - zadužen je za napredne funkcionalnosti praćenja stanja svake mrežne konekcije, a važan je i dodatno jer se preko njega definira maksimalan broj aktivnih konekcija.



Za opis rada **vatrozida**, pogledajte: **26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije.**

- **`Druge funkcionalnosti`** – ovdje su navedene neke od dodatnih funkcionalnosti:
    - `edac_core` - zadužen je za **EDAC** (*Error Detection And Correction*) funkcionalnost unutar dijela memorijskog kontrolera *CPUa*, a oslanja se na upravljački program za *EDAC* sklop *(chip).*
    - `ac` - zadužen je za *ACPI AC adapter*- dio za napajanje (paljenje, gašenje i slično).
    - `fan` - on je zadužen za mjerenja i kontrolu sistemskih ventilatora (preko *ACPI* funkcija).

Vjerujemo kako ste sada shvatili gore navedenu podjelu kernel modula. Sada krećemo s radom vezanim za kernel.

**Izvori informacija: (88),(1115),(1197),(1391),(1392),(K-2),(K-12), `man 2 syscalls`,`man 7 signal`.**

---

## 11.1.1. Rad s kernelom

***Slijedi napredno poglavlje (11.1.1.x)!***

**Koje su sve generacije Linux kernela u upotrebi danas (2023.g.)?**

Danas su u upotrebi sljedeće generacije Linux kernela

- **3.x**
- **4.x**
- **5.x**
- **6.x (***od kraja 2022. godine***)**

Važno je znati da je Linux kernel odnosno jezgra operativnog sustava Linux besplatna jer se razvija prema licenci otvorenog izvornog koda, a prema unutarnjoj arhitekturi, ona je ***monolitna***, ***modularna*** i ***višezadaćna***.
Linux kernel je *monolitan* što znači da se sve osnovne funkcije, poput upravljanja memorijom, upravljanja procesima (pokretanjem/radom i zaustavljanjem rada programa), upravljanja uređajima te diskovnim, mrežnim i drugim podsustavima, izvršavaju u istom adresnom prostoru kernela. Sve ove funkcije su implementirane unutar jednog velikog bloka programskog koda, čineći sâm kernel jedinstvenom komponentom, a u konačnici i jednom posebnom (izvršnom) *monolitnom* datotekom. *Monolitni* model kernela koristi sistemske pozive kao mehanizam za interakciju između programa koji se izvršavaju u takozvanom korisničkom (izoliranom) prostoru i kernela. Korisnički procesi (programi) pri tome za rad trebaju određene funkcije (npr. učitavanje datoteke, pristup mreži ili upravljanje procesima) do kojih dolaze pozivajući odgovarajući sistemski poziv. Ovi sistemski pozivi stoga prelaze granicu između korisničkog prostora i kernela kako bi omogućili izvršavanje takozvanih privilegiranih operacija. Sistemske pozive koristi monolitni kernel kao način za pružanje standardiziranog sučelja korisnicima odnosno korisničkim programima, čime se osigurava kontrolirana i sigurna komunikacija s kernelom i u konačnici s hardverom. Ovakva monolitna arhitektura olakšava brzu komunikaciju između različitih komponenti sustava, ali može povećati složenost i smanjiti otpornost na greške u odnosu na ***mikroservisnu*** ili ***mikrokernelnu*** arhitekturu.
Pojam ***modularan*** se odnosi na činjenicu da se upravljački programi uređaja mogu učitavati uz kernel i kao moduli. Osim toga se i određene nove značajke (pr. mrežni protokoli, određene funkcije i sl.) mogu koristiti kao kernel moduli odnosno vanjske komponente koje nisu direktno ugrađene u sâm kernel. Naime u sam kernel su osim navedenih osnovnih funkcija rada svakog operativnog sustava poput upravljanja memorijom, procesima, diskovnim, mrežnim i drugim podkomponentama, već prilikom kompiliranja ugrađeni mnogi upravljački programi, dodatne funkcije i protokoli. To još dodatno naglašava pojam ***monolitnosti***.

Nadalje, postoji i nekoliko vršnih stâbala kernela, od kojih svako stablo ima svoju namjenu:

- **Mainline** - ovo stablo održava *Linus Torvalds*. To je stablo na kojem se uvode sve nove značajke, upravljački programi, novi protokoli i slično i na kojem se odvija sav novi razvoj. Novi "*mainline*" kerneli izlaze svakih 9-10 tjedana.
- **Stable (***stabilni***)** - nakon što se svaka glavna grana ("*mainline*") kernela objavi, a nakon prolaska nekoliko ***RC*** (engl. *Release Candidate*) iteracija i testiranja, ona se smatra stabilnom odnosno postaje ***"Stable***". Svi popravci grešaka za ovaj stabilni kernel dohvaćaju se iz glavnog stabla ("*mainline*") a primjenjuju ih ovlašteni održavatelji (ovog) stabilnog kernela. Obično postoji samo nekoliko izdanja kernela s ispravcima pogrešaka dok sljedeći "*mainline*" kernel ne postane dostupan, osim ako nije označen kao kernel za dugoročno održavanje odnosno **LTS**. Ažuriranja ovog stabla kernela objavljuju se prema potrebi, obično jednom tjedno.
- **LTS** - ako je stabilna inačica kernela označena kao kernel s dugoročnom podrškom to jest takozvani **LTS** (engl *Long Term Support*), ona će se održavati dodatnih nekoliko godina. Dakle unutar svake generacije odnosno stabla kernela razvijaju se i njegove pôd inačice od kojih se neke starije slabije razvijaju i održavaju dok se daje naglasak na novije. Zbog toga je unutar svake generacije kernela odlučeno kako će se određene pôdinačice proglasiti dugoročnim odnosno takozvanim ***LTS*** (engl. *Long Term Support*) inačicama. Na tim *dugoročnim* inačicama su bazirane razne poslužiteljske distribucije Linuxa. Naime poslužitelji se u većini slučajeva ne reinstaliraju svako malo, jer želimo pouzdan i provjeren sustav na kojem se neće svaki tren mijenjati krucijalni dijelovi sustava, a koji bi mogli uzrokovati razne nestabilnosti i probleme. Zbog toga je potrebno neko dugoročno rješenje za višegodišnje sigurne i pouzdane zakrpe i poboljšanja za takve inačice Linuxa. Upravo zbog toga i postoje ***LTS*** inačice Linux kernela.

**LTS** odnosno dugoročno podržane inačice linux kernela (***LTS***) su:

| LTS inačica | Tko je odgovoran za razvoj | Kada je objavljena | Kraj podrške (EOL) |
|---|---|---|---|
| 6.6 | Greg Kroah-Hartman & Sasha Levin | 2023-10-29 | 2026-12 |
| 6.1 | Greg Kroah-Hartman & Sasha Levin | 2022-12-11 | 2026-12 |
| 5.15 | Greg Kroah-Hartman & Sasha Levin | 2021-10-31 | 2026-12 |
| 5.10 | Greg Kroah-Hartman & Sasha Levin | 2020-12-13 | 2026-12 |
| 5.4 | Greg Kroah-Hartman & Sasha Levin | 2019-11-24 | 2025-12 |
| 4.19 | Greg Kroah-Hartman & Sasha Levin | 2018-10-22 | 2024-12 |
| 4.14 | Greg Kroah-Hartman & Sasha Levin | 2017-11-12 | 2024-01 |

*Oznaka **EOL** označava kraj životnog vijeka projekta (engl. End of Life).*

Za one inačice kernela koje su označene kao **LTS** odnosno one koje imaju produljenu podršku od strane zajednice programera koja ih razvija, u većini slučajeva, naznačeni **EOL** datum označava kraj ciklusa njihovog razvoja odnosno njihovog životnog vijeka. Međutim, samo određene **LTS** inačice kernela odabiru se i potom označavaju kao inačice sa *Super-Long-Term Support* (**SLTS**), a to znači da će ih se održavati još narednih nekoliko godina, ali od strane ***Civil Infrastructure Platform*** (*CIP*) projekta, koji je dio Linux fondacije. Takve (**SLTS**) inačice koriste se za iznimno važne poslužitelje, ugrađene i druge kritične sustave.

Pogledajmo koje generacije Linux kernela su označene kao **SLTS**, a koje održava **CIP** platforma:

| Inačica | Tko je odgovoran za održavanje | Datum prve objave | Projicirani kraj životnog vijeka (EOL) |
|---|---|---|---|
| SLTS v6.1 | Nobuhiro Iwamatsu & Pavel Machek | 2023-07-14 | 2033-08 |
| SLTS v6.1-rt | Pavel Machek | 2023-07-16 | 2033-08 |
| SLTS v5.10 | Nobuhiro Iwamatsu & Pavel Machek | 2021-12-05 | 2031-01 |
| SLTS v5.10-rt | Pavel Machek | 2021-12-08 | 2031-01 |
| SLTS v4.19 | Nobuhiro Iwamatsu & Pavel Machek | 2019-01-11 | 2029-01 |
| SLTS v4.19-rt | Pavel Machek | 2019-01-11 | 2029-01 |
| SLTS v4.4 | Ulrich Hecht | 2017-01-17 | 2027-01 |
| SLTS v4.4-rt | Pavel Machek | 2017-11-16 | 2027-01 |

## Kako saznati koju inačicu Linux kernela imamo na računalu?

Važno je znati da naša distribucija Linuxa (***RedHat/CentOS/Rocky)***, standardno dolazi s dvije starije inačice (verzije) kernela, i jednom najnovijom. Odabir inačice kernela je automatski takav, da će se u pravilu sustav uvijek pokrenuti sa zadnjom (najnovijom) inačicom kernela.

I nakon nadogradnje sustava, uvijek ćemo imati automatski pokrenutu najnoviju inačicu te dostupne i prve dvije starije. Starije dvije inačice se čuvaju, kako bi sustav ipak mogli ručno pokrenuti u slučaju kada nešto eventualno pođe po zlu. Provjerimo koju inačicu Linux kernela imamo pokrenutu na računalu, s naredbom: `uname -r`

```
uname -r

2.6.32-573.3.1.el6.x86_64
```

Što znače oznake koje vidimo?.

Opisat ćemo oznaku, odnosno broj, po broj:

- Prvo vidimo vršnu inačicu odnosno generaciju (engl. *Major version*) kernela, što vidimo na početku kao: `2.`
- Zatim slijedi (poslije točke), koja je nîža inačica (engl. *Minor version*) kernela, kod nas je to: `6.`
- Potom vidimo i pôd inačicu `32` - dakle pôd inačica: `2.6.32` je označena kao inačica s produženom podrškom odnosno **LTS** (*Long Term Support*), kako vidimo iz popisa ***LTS*** inačica kernela (gore) **[u vrijeme pisanja dok je 2.6.32 bila LTS inačica]**.
- Nadalje, nakon znaka – slijedi: `573.3.1` što označava nadogradnju odnosno „update/fix" inačicu.
- Potom je vidljiva oznaka `el6` koja označava "*Enterprise Linux*" v.6. a odnosi se na ***RedHat Enterprise Linux*** generacije 6.x iako se u našem primjeru radi o *CentOS* Linuxu 6.6. koji je baziran na *RedHat Enterprise Linuxu* 6.6.
- Oznaka `x86` pred kraj označava da je arhitektura procesora x86 te `64` na kraju znači kako se radi o 64 bitnom kernelu i sâmom Linuxu.

**Izvori informacija: (203),(1564),(1565),(1566).**

### 11.1.1.1. Kako vidjeti koji kernel moduli su trenutno pokrenuti

Ako želimo vidjeti koji su sve kernel moduli odnosno upravljački programi (engl. *driveri*) učitani, pokrenimo naredbu `lsmod` Ovo se odnosi na one kernel module koji nisu bili kompilirani direktno u kernel odnosno koji nisu već ugrađeni u sâm kernel već su učitani naknadno u procesu pokretanja operativnog sustava, kao zasebni kernel moduli.
Sada i pogledajmo što je sve učitano od kernel modula (ispis je skraćen zbog potrebe lakšeg razumijevanja):

```
lsmod

Module                  Size  Used by
iptable_filter          2793  1
ip_tables              17831  2 iptable_filter
e1000e                230782  0
ext4                  378683  1
cdrom                  39085  1 sr_mod
ata_piix               24409  2
pata_acpi               3701  0
ata_generic             3837  0
i915                 1015889  2
video                  21654  1 i915
output                  2409  1 video
```

Objasnit ćemo sâmo nekoliko kernel modula s ove liste. S lijeve strane (stupac: `Module`) je ime kernel modula, potom imamo veličinu kernel modula [*u bajtima*] u memoriji (`Size`), a s desne strane u stupcu `Used by` je vidljivo koji (drugi) kernel modul ga koristi, u slučajevima kada jedan kernel modul ovisi o drugome. Pogledajmo neke od gore ispisanih kernel modula:

- `ip_tables` - ovaj kernel modul nam daje podršku za funkcionalnost vatrozida. Njega koristi: `iptable_filter`.
- `e1000e` - ovo je upravljački program za ***Intelovu*** gigabitnu mrežnu karticu.
- `ext4` - ovo je upravljački program koji donosi podršku za Linux ***ext4*** datotečni sustav.
- `cdrom` - ovo je upravljački program koji donosi podršku za CD/DVD-ROM uređaje.
- `ata_piix` i `pata_acpi` - ovo su upravljački programi za disk kontrolere (SCSI/ATA/PATA).
- `i915` - ovo je upravljački program za *Intel i915* integriranu grafičku karticu.
- `video` i `output` su upravljački programi koji koriste grafičke kartice - u ovom slučaju: ***Intel*** **i915 (**kernel modul: `i915`**).**

*Lista svih učitanih kernel modula zapisuje se u posebnu datoteku* **`/proc/modules`**

**Izvori informacija: (203),(204),(1115), `man lsmod`, `man modinfo`, `man proc`, `modinfo XY` (`XY`=*ime kernel modula*).**

### 11.1.1.2. Napredno: Rad s kernel modulima

## Gdje se nalaze kernel moduli?

Kernel moduli se nalaze unutar direktorija `/lib/modules/` unutar pôddirektorija čije ime odgovara trenutnoj aktivnoj inačici kernela. Iz ovoga je jasno kako je moguće imati više inačica Linux kernela od kojih je sâmo jedna aktivna, a to je ona u trenutnoj upotrebi, odnosno ona s kojom je trenutno pokrenut operativni sustav.
Pogledajmo koja je naša trenutno aktivna inačica kernela, s naredbom `uname`, ako ju pokrenemo na sljedeći način:

```
uname -r

2.6.32-573.3.1.el6.x86_64
```

Kernel moduli se za našu inačicu kernela tada nalaze unutar sljedećeg direktorija (mape):
`/lib/modules/2.6.32-573.3.1.el6.x86_64` točnije unutar direktorija (mape):
`/lib/modules/2.6.32-573.3.1.el6.x86_64/kernel/`
Za više informacija o pojedinom kernel modulu, pogledajte stranicu: https://www.kernel.org/doc/Documentation/. Kategorije navedene ovdje su identične onima koje imate unutar vašeg: `/lib/module/`**`KERNEL`**`/kernel/` direktorija (mape).

## Što je s kernel modulima u sâmom kernelu?

Kao što smo rekli i sâm kernel može imati unutar sebe ugrađene kernel module, koji su uglavnom upravljački programi za hardver, odnosno takozvani „*Driveri*“. Spomenuli smo i kako se sâm kernel (mislimo na datoteku) nalazi unutar `/boot/` direktorija. Unutar istog tog direktorija se nalazi i datoteka u kojoj su definirane sve dodatne funkcionalnosti i upravljački programi (kernel moduli), s kojima je kompiliran konkretan kernel odnosno koji su već ugrađeni u konkretan kernel, kao i oni koji su dostupni kao kernel moduli. Dakle ako je naš kernel: `2.6.32-573.3.1.el6.x86_64` tada je sâm kernel zapravo datoteka: `/boot/vmlinuz-2.6.32-573.3.1.el6.x86_64`.
Na istoj putanji unutar `/boot/` direktorija (mape) nalaze se: inicijalni RAM disk to jest datoteka imena prema principu: `initramfs-`***`INAČICAKERNELA`***`.img` (pogledajte poglavlje: **10.7.1.2.1.**) te konfiguracijska datoteka sâmog kernela, koja ima ime, prema principu: `config-`**`INAČICAKERNELA`**.

Konfiguracijska datoteka kernela u našem primjeru je datoteka imena: `/boot/config-2.6.32-573.3.1.el6.x86_64`

Unutar ove datoteke su navedene sve funkcionalnosti i kernel moduli koje je moguće koristiti (ili ne koristiti) na sustavu, i to:

- Unutar sâmog (konkretnog) kernela.
- Kao (vanjske) kernel module odnosno vanjske kernel modul datoteke.
- Ili se uopće ne koriste odnosno ne mogu se koristiti.

**Kako to izgleda?**
Pogledajmo jedan manji dio konfiguracijske datoteke našeg kernela: `/boot/config-2.6.32-573.3.1.el6.x86_64`:

```
CONFIG_X86_64=y
CONFIG_X86=y
#
# SCSI device support
CONFIG_SCSI=y
#
# SCSI Transports
CONFIG_SCSI_AIC7XXX=m
CONFIG_ATA_ACPI=y
CONFIG_SATA_AHCI=m
CONFIG_ATA_PIIX=m
#
# USB Network Adapters
CONFIG_USB_RTL8150=m
CONFIG_USB_NET_AX8817X=m
```

Vidljivo je kako su neki kernel moduli ili funkcionalnosti uključene direktno u kernelu, a neke su dostupne kao kernel moduli.



U ekstremnim slučajevima, moguće je i zabraniti učitavanje kernel modula (kao datoteka), postavljanjem vrijednosti **`1`** u varijablu: **`/proc/sys/kernel/modules_disabled`**

Važno je znati sljedeće:

- Svi parametri koji imaju vrijednost `=y` su uključeni direktno u kernel. Ovo znači kako su kompilirani zajedno s kernelom i nalaze se u sâmom kernelu, a koji je u našem slučaju datoteka: `/boot/vmlinuz-2.6.32-573.3.1.el6.x86_64`.
- Svi parametri koji imaju vrijednost `=m` su dostupni kao kernel moduli (datoteke) na disku i mogu se učitavati prema potrebi[(204)], s naredbama `modprobe` ili `insmod` (o njima malo kasnije).
- Svi parametri koji imaju vrijednost `=n` **nisu** dostupni niti kao kernel moduli (datoteke), niti unutar sâmog kernela.

## Informacije o konkretnom kernel modulu

Informacije o željenom *kernel* modulu dobivamo s naredbom `modinfo`. Recimo da nas zanima *kernel* modul za USB Ethernet (RJ-45) mrežnu karticu baziranu na *sklopu* **Asix AX88772B**. Njen kernel modul je (datoteka) imena: `asix.ko`
Ovo nas zanima jer smo kupili USB mrežnu karticu te želimo učitati njen upravljački program (*kernel modul*) kako bi uopće radila. Provjerimo koje informacije ćemo dobiti o željenom kernel modulu, koji je došao kao datoteka uz naše kernel:

```
modinfo asix

filename:       /lib/modules/2.6.32-573.3.1.el6.x86_64/kernel/drivers/net/usb/asix.ko
license:        GPL
description:    ASIX AX8817X based USB 2.0 Ethernet Devices
author:         David Hollis
srcversion:     41AB1B1334631410071EA9C
alias:          usb:v17EFp7203d*dc*dsc*dp*ic*isc*ip*
depends:        mii,usbnet
vermagic:       2.6.32-573.3.1.el6.x86_64 SMP mod_unload modversions
```

Sada možemo vidjeti gdje se točno ovaj kernel modul (datoteka) nalazi, a što je vidljivo u prvom redu, nakon `filename:`.
Ovdje je još vidljivo i to da je ime datoteke na disku, koja predstavlja kernel modul: `asix.ko`.



Svi Linux kernel moduli (datoteke) imaju ekstenziju `.ko` koja znači "*Kernel Object*". Dodatno, kernel moduli mogu biti i komprimirani, što se odrađuje u trenutku kompiliranja sâmog kernela, pa će tada kernel moduli imati dodatnu ekstenziju:

- `.ko.xz` – ako su komprimirani sa XZ algoritmom za kompresiju.
- `.ko.gz` – ako su komprimirani sa GZIP algoritmom za kompresiju.

Nadalje je vidljivo, ovisi li naš kernel modul o nekom drugom kernel modulu, što je opet vidljivo u redu `depends:`.
Naime za neke kernel module je potrebno da se neki dugi kernel moduli učitaju prije njih, za što se kaže da za njih postoji neka ovisnost o drugim modulima (engl. *dependencies*). U našem slučaju, vidljivo je kako se prije učitavanja našeg kernel modula moraju učitati kernel moduli: `mii` i `usbnet`.

Dodatno za kernel module koji to dozvoljavaju, moguće im je poslati i neke parametre. Kako bismo provjerili koje sve parametre podržava određeni kernel modul, to možemo saznati pomoću naredbe `modinfo -p`. Pogledajmo primjer za kernel modul `e1000e`, a on je upravljački program za određene gigabitne mrežne kartice tvrtke ***Intel***.
Njega smo uzeli za primjer je on konkretno podržava razne dodatne parametre rada (ispis smo skratili):

```
modinfo -p e1000e

debug:Debug level (0=none,...,16=all) (int)
copybreak:Maximum size of packet that is copied to a new buffer on receive (uint)
TxIntDelay:Transmit Interrupt Delay (array of int)
TxAbsIntDelay:Transmit Absolute Interrupt Delay (array of int)
RxIntDelay:Receive Interrupt Delay (array of int)
RxAbsIntDelay:Receive Absolute Interrupt Delay (array of int)
InterruptThrottleRate:Interrupt Throttling Rate (array of int)
IntMode:Interrupt Mode (array of int)
SmartPowerDownEnable:Enable PHY smart power down (array of int)
WriteProtectNVM:Write-protect NVM [WARNING: disabling this can lead to corrupted NVM]
CrcStripping:Enable CRC Stripping, disable if your BMC needs the CRC (array of int)
```

Parametri koji su trenutno postavljeni vidljivi su unutar `/proc/` datotečnog sustava. Konkretno za `e1000e` modul su vidljivi unutar direktorija: `/sys/module/e1000e/parameters/` i to po jedna datoteka za po jedan parametar.
Dakle putanja je: `/sys/module/`***`IME-KERNEL-MODULA`***`/parameters/`.
Čisto za informaciju, nakon kompiliranja kernela, kreira se i datoteka imena: `modules.dep` u koju se zapisuju sve međusobne ovisnosti kernel modula (*dependencies*).
Ova datoteka se nalazi u direktoriju (mapi): `/lib/modules/`***`INAČICA-KERNELA`***`/`


U slučaju kada ste nešto nehotice promijenili na datoteci **`modules.dep`** sustav vam nakon prvog restarta potencijalno neće raditi kako treba. To se može dogoditi jer ne može učitati takozvane *dependencies* module (one o kojima ovisi) pa sâmim time i one koji dodatno ovise o njima. Srećom postoji jednostavno rješenje.
Naime tijekom kompiliranja kernela kreira se i posebna tablica sistemskih simbola, koja se snima u datoteku imena: **`System.map-`*`INAČICA-KERNELA`***, a koja se i kod instalacije novog kernela kopira s kernelom u direktorij **`/boot/`** u koji se obično kopira i sâm kernel.

Navedena datoteka sa sistemskim simbolima je dostupna pomoću sljedeće poveznice:
`/boot/System.map-$(uname -r)`
U ovoj tablici simbola se nalazi sve potrebno kako bi se ponovno, za trenutno aktivni kernel, kreirala datoteka s tablicom ovisnosti kernel modula odnosno, da bi se ponovno kreirala datoteka: `modules.dep`
S naredbom `depmod` možemo ponovno kreirati datoteku `modules.dep` na sljedeći način:

```
depmod -F /boot/System.map-$(uname -r)
```

**Izvori informacija: (204),(205),(206),(1115), `man depmod`,`man 5 depmod.d`,`man modinfo`.**

## Kako ispisati listu svih dostupnih kernel modula na našem Linux sustavu?

S obzirom da znamo lokaciju (putanju) na kojoj se nalaze svi kernel moduli (iako su grupirani u pôddirektorije), možemo ih izlistati na jednostavan način. Pokrenite sljedeću naredbu odnosno nîz naredbi:

```
ls -R /lib/modules/$(uname -r)
```

Što radi ova naredba?

- `ls -R` rekurzivno izlistava sve direktorije i pôddirektorije te sve datoteke unutar njih. Pri tome je: `/lib/modules/` početna putanja (direktorij) odakle treba krenuti s izlistanjem, proširena je sa: `$(uname -r)` koji nam daje točnu inačicu linux kernela koja je aktivna unutar ovog vršnog direktorija.

Ono što će zapravo Linux pokrenuti u našem slučaju *(za našu konkretnu inačicu kernela)* je :

```
ls -R /lib/modules/2.6.32-573.3.1.el6.x86_64
```

Nakon gore navedene i pokrenute naredbe, dobili smo potpuno izlistanje (koju nećemo ovdje ispisati zbog veličine).
Nadalje, sve možemo i filtrirati, kako bismo izvukli samo imena `.ko` datoteka (`grep` naredba), a potom kako bismo maknuli ekstenziju `.ko` koristimo `cut` naredbu. Potom ćemo sve poredati po abecedi s naredbom `sort`. Stoga pokrenite sljedeće:

```
ls -R /lib/modules/$(uname -r) | grep "\.ko" | cut -f1 -d"." | sort
```

Druga i znatno jednostavnija metoda pronalaska svih kernel modula bi bila pomoću naredbe `find`:

```
find /lib/modules/$(uname -r) -type f -name '*.ko*'
```

S njom pronalazimo sve datoteke čija ekstenzija počinje sa `.ko` a nakon te ekstenzije može slijediti bilo koja dodatna ekstenzija jer je moguće da su neki kernel moduli komprimirani pa će imati dodatnu ekstenziju (mogu biti: `.ko.xz` ili `.ko.gz`).

S navedenom naredbom `find` se sve pretražuje počevši od navedenog direktorija na početku (*/lib/modules/XY*).

**Kako vidjeti listu svih kernel modula koji su kompilirani u sâm kernel odnosno nalaze se unutar kernela**
```
cat /lib/modules/$(uname -r)/modules.builtin
```

Odnosno ta ista lista je vidljiva u konfiguracijskoj datoteci, s kojom je trenutni kernel kompiliran (opcija =y):
```
grep "=y" /boot/config-`uname -r`
```

## Binarna kompatibilnost kernel modula (upravljačkih programa)

Linux ne pruža stabilan **API** ili **ABI** za kernel module odnosno upravljačke programe. To je namjerno napravljeno, da bi se kernel moduli morali kompilirati s novim kernelom, koji u pravilu nudi nove, stabilnije, sigurnije i poboljšane mehanizme rada.
Iz toga je jasno da postoje razlike u unutarnjoj strukturi i funkcijama između različitih inačica kernela, što može uzrokovati probleme s kompatibilnošću. Pošto sa svakom inačicom linux kernela dolaze kernel moduli namijenjeni za točno određenu inačicu kernela, to znači da u većini slučajeva kernel modul koji je napravljen za jednu inačicu kernela neće raditi na drugoj.
Drugi operativni sustavi, kao što su ***Solaris***, ***FreeBSD***, ***macOS*** i ***Windows***, održavaju **API** i **ABI** relativno stabilnim, čime se izbjegava ovaj problem. Na primjer, ***FreeBSD*** kernel moduli kompilirani s inačicom kernela 6.0 će raditi bez ponovne kompilacije na bilo kojoj drugoj vršnoj inačici ***FreeBSD*** 6.x., primjerice 6.4. Međutim, oni nisu kompatibilni s drugim vršnim inačicama i moraju se ponovno kompilirati za korištenje s primjerice inačicom *FreeBSD* **7.x**, budući da se kompatibilnost **API**-ja i **ABI**-ja održava samo unutar vršne grane inačica (pr. unutar 6.x ili unutar 7.x i slično).

U pokušaju borbe protiv tih problema, podaci o inačicama simbola unutar kernela smješteni su i unutar `.modinfo` odjeljka učitanih (***ELF)*** modula u binarnom zapisu kernel modula. Popis svih trenutno aktivnih simbola (funkcija ili varijabli) ovisi o trenutnom kernelu, nalazi se u `System.map` datoteci, koju možemo vidjeti kao: `/boot/System.map-`uname -r``.
Isto tako se tablica sa simbolima nalazi u datoteci koja se kreira prilikom kompiliranja kernel modula (`Module.symvers`), a koja se kasnije povezuje s datotekom: `/boot/symvers-`uname -r`.gz`, za konkretan kernel (vidljiv s ``uname -r``).

U svakom slučaju, ovdje uz svaki kernel modul, u prvom stupcu, stoji i provjerni zbroj (**CRC**), koji ovisi o inačici Linux kernela za koji je kompiliran. ***Symvers*** tablica se praktično koristi kao jednostavna provjera konzistencije s **ABI** pozivima.



Za detalje o učitavanju kernel modula pogledajte poglavlje:
**11.1.1.7. Mehanizam prepoznavanja hardvera i učitavanje kernel modula.**
Pogledajte i poglavlje (posebno dio „**Tablica svih podržanih simbola (Symvers)**“):
**11.1.1.4. Kompiliranje novog kernela.**

Navedene informacije o inačici kernel modula se uspoređuju s onima u aktivnom kernelu, prije učitavanja kernel modula.
Ako su inačice nekompatibilne (kernela i kernel modula), kernel modul se neće moći učitati.
S naredbom `modinfo`, tražeći pojam `vermagic`, možemo vidjeti za koji kernel je određeni kernel modul kompiliran.

Pogledajmo što možemo vidjeti, primjerice za kernel modul imena: `virtio_net`, što se tiče inačice kernela za koji je konkretni kernel modul kompiliran:
```
modinfo virtio_net | grep vermagic
vermagic:       5.15.7-1.el8.elrepo.x86_64 SMP mod_unload modversions
```
Vidimo da je konkretni kernel modul kompiliran uz kernel inačice: `5.15.7-1`.



Za detalje o **ABI** i **API**, pogledajte poglavlje:
**10.4.4.2. API i ABI.**

**Slijedi napredna cjelina!**

## Drugi slučajevi binarne (ne)kompatibilnost kernel modula.

U praksi postoje slučajevi u kojima netko može imati potrebu razviti kernel modul bez stavljanja njegovog izvornog kôda uz programski kôd konkretne inačice kernela, barem kroz određeno vremensko razdoblje. Ovu situaciju nazivamo modelom razvoja kernel modula izvan stabla kernela (engl. *out-of-tree kernel module model*).

U ovom modelu, pretpostavimo da je kernel modul u početku razvijen prema sučelju izloženom u kernelu s vršnim brojem inačice 3.10. Međutim kada se prelazi na kernel inačice 3.11, autori tog kernel modula bi morali nekako znati je li se promijenio takozvani binarni **KMI** (*Kernel Module Interface*) odnosno **ABI** prema kojem je modul razvijen. A ako jest, koje su se promjene dogodile. S tim znanjem, autori kernel modula mogu procijeniti potrebu za ponovnim kompiliranjem svog kernel modula kako bi ga učinili kompatibilnim s novom inačicom kernela 3.11. To znači i to da u teoriji kernel modul koji je kompiliran i radi s jednom inačicom kernela, potencijalno može raditi i s drugom inačicom, pod uvjetom da se **ABI** odnosno **KMI** nije promijenio, barem za one varijable i funkcije koje naš kernel modul koristi.

U praksi, skup funkcija i globalnih varijabli (kao i njihovi tipovi) definiraju binarni **KMI** (**ABI**) koji se obično razlikuje od inačice do inačice kernela odnosno njih je u ovom slučaju potrebno usporediti. Stoga prvo instalirajmo novi softverski paket:

```
yum -y install libabigail
```

U ovom paketu ćemo dobiti novu naredbu `abipkgdiff` koja se koristi za usporedbu dva softverska paketa kernela za usporedbu **KMI**-ja odnosno **ABI**-ja. S njom ćemo zapravo usporediti kompatibilnost dvije inačice kernela po pitanju **ABI**-ja.

Pogledajmo koju inačicu **RedHat/CentOS/Rocky** distribucije linuxa trenutno koristimo:

```
cat /etc/redhat-release
CentOS Linux release 7.8.2003 (Core)
```

Vidimo da koristimo **CentOS** inačice **7.8.2003**. Sada ćemo kreirati jedan direktorij i u njega skinuti trenutnu inačicu kernela i jednu noviju. Međutim prvo provjerimo koju inačicu kernela sada imamo:

```
uname -r
3.10.0-1127.el7.x86_64
```

Sada kopirajmo *RPM* softverske pakete starog kernela (**3.10.0-1127**): *kernel*, *debug info kernel* i **ABI** (lista **ABI** simbola)

```
mkdir /root/kernel
cd /root/kernel
wget https://vault.centos.org/7.8.2003/os/x86_64/Packages/kernel-3.10.0-1127.el7.x86_64.rpm
wget http://debuginfo.centos.org/7/x86_64/kernel-debug-debuginfo-3.10.0-1127.el7.x86_64.rpm
wget https://vault.centos.org/7.8.2003/os/x86_64/Packages/kernel-abi-whitelists-3.10.0-1127.el7.noarch.rpm
```

Zatim kopirajmo *RPM* softverske pakete novog kernela za noviji *CentOS/RedHat* 7.9, koji koristi kernel **3.10.0-1160**:

```
wget http://ftp.bme.hu/centos/7.9.2009/os/x86_64/Packages/kernel-3.10.0-1160.el7.x86_64.rpm
wget http://debuginfo.centos.org/7/x86_64/kernel-debug-debuginfo-3.10.0-1160.el7.x86_64.rpm
wget http://ftp.bme.hu/centos/7.9.2009/os/x86_64/Packages/kernel-abi-whitelists-3.10.0-1160.el7.noarch.rpm
```

*Napomena: adrese (URL) koje navodimo ovdje, mogu se s vremenom mijenjati, tako da pripazite na to.*

Sada imamo po tri softverska paketa za svaku inačicu kernela, koje ćemo iskoristiti za usporedbu **ABI**-ja to jest **KMI**-ja:

```
abipkgdiff --d1 kernel-debug-debuginfo-3.10.0-1127.el7.x86_64.rpm \
--d2 kernel-debug-debuginfo-3.10.0-1160.el7.x86_64.rpm \
--wp kernel-abi-whitelists-3.10.0-1160.el7.noarch.rpm \
kernel-3.10.0-1127.el7.x86_64.rpm \
kernel-3.10.0-1160.el7.x86_64.rpm > kernel.ABIdiff.txt
```

Nakon nekog vremena (dok se sve raspakira i usporedi), dobivamo analizu usporedbe kompatibilnosti **ABI**-ja u tekstualnoj datoteci: **`kernel.ABIdiff.txt`**. Pogledajmo što smo mi dobili kao rezultat usporedbe (izrazito smo skratili ispis):

```
2 Changed variables:
  [C] 'efi efi' was changed at efi.c:25:1:
    size of symbol changed from 264 to 272; type of variable changed:
  [C] 'task_struct init_task' was changed at init_task.c:18:1:
    size of symbol changed from 11464 to 11480
...
'struct blk_mq_aux_ops at blk-mq.h:169:1' changed: type size changed from 256 to 320
'struct inet_peer at inetpeer.h:32:1' changed: type size changed from 1472 to 1536
```

Ovo su korisne informacije, koje upućuju na to da kernel modul koji je kompiliran s trenutnom inačicom kernela (**3.10.0-1127**) potencijalno (ako se koriste navedene funkcije ili varijable koje su se mijenjale) neće raditi na novom kernelu (**3.10.0-1160**).

Za detalje možemo koristiti i naredbu `readelf`, ali prvo dekomprimirajmo jedan kernel modul, primjerice: `virtio_net`:

```
cp /lib/modules/5.15.7-1.el8.elrepo.x86_64/kernel/drivers/net/virtio_net.ko.xz /root
xz -d /root/virtio_net.ko.xz
cd /root
```

Sada ga možemo analizirati, primjerice sa: `readelf -s virtio_net.ko` ili s prekidačima: **`-S`** ili **`-e`** ili **`--syms`**.

**Izvori informacija: (204),(205),(206),(566),(1115),(1116),(1117),(1118),(1119),(1120), `man depmod`,`man modinfo`, `man abipkgdiff`,`man readelf`,`man xz`.**

## Kako se (ručno) učitavaju kernel moduli?

Postoje dvije metode ručnog učitavanja kernel modula, odnosno dvije naredbe za tu namjenu.

- `insmod` naredba učitava navedeni kernel modul ali se ne brine o tome traži li taj kernel modul za rad, prethodno učitavanje nekog drugog kernel modula (tzv. *Dependency check*).
- `modprobe` je slična kao i prethodna naredba s time da ona prije pokušaja učitavanja primarnog kernel modula provjerava je li potrebno prethodno učitati neke druge kernel module, te ih ona po potrebi i učitava (koliko god ih bilo), a na kraju učitava i naš (primarni) kernel modul.

U našem primjeru spajanja nove USB mrežne kartice, poput donje (bijele) na slici, koja koristi ***ASIX AX88772B*** integrirani sklop, za koji je potrebno učitati kernel modul (upravljački program/driver) `asix`. Međutim prije toga je potrebno učitati kernel module (ako već nisu učitani): `mii` i `usbnet`. Kernel moduli se učitavaju prema imenu datoteke bez ekstenzije. Primjerice datoteka za navedeni kernel modul se zove `asix.ko`, a kernel modul učitavamo kao `asix`. Da bismo to napravili koristit ćemo naredbu `modprobe`. Međutim, prvo pogledajmo informacije o ovom kernel modulu (upravljačkom programu) za koji smo saznali da je zadužen za ovu USB mrežnu karticu:

```
modinfo asix -d
ASIX AX8817X based USB 2.0 Ethernet Devices
```

← Vidimo da je u opisu kernel modula vidljivo da je namjenjen za **ASIX AX8817x** čipove (kompatibilne s našim).

**Slika 82. USB LAN odnosno USB mrežne kartice: *ASIX AX88772B i REALTEK RTL 8153***



Na slici su vidljive dvije USB mrežne kartice: gornja (crna) sa Realtek **RTL8153** sklopom (čipom) te donja (bijela) sa ASIX **AX88772B** čipom. Mi ćemo se ovdje fokusirati na ASIX USB mrežnu karticu sa **AX88772B** sklopom (čipom).
Učitajmo navedeni kernel modul odnosno upravljački program naše (**AX88772B**) mrežne kartice, imena `asix`, na sljedeći način:

```
modprobe asix
```

Sada možemo provjeriti jesu li se učitali i svi potrebni kernel moduli koji su se morali učitati prije našeg kernel modula (tzv. *module dependencies*):

```
lsmod |grep asix

asix                   15796  0
usbnet                 36510  1 asix
mii                     5376  2 asix,usbnet
```

Vidimo kako su se prvo učitali kernel moduli `mii` i `usbnet` te potom i `asix`. Vidljivo je i sljedeće: kernel modul `usbnet` koristi kernel modul `asix` odnosno taj kernel modul mu je potreban za rad. Kernel modul `mii` koristi kernel module `asix` i `usbnet`. To znači da je sve u redu i naša nova USB mrežna kartica bi sada trebala raditi te biti vidljiva kao neko od `eth`**XY** sučelja.



Pogledajte i duge primjere učitavanja i rada s kernel modulima i komponentama kernela, u sljedećim poglavljima:
**10.7.1.2. CPU Mikrokod.**
**10.7.1.2.1. initrd i/ili initramfs i inicijalizacija sustava.**
**11.1.1.3. Parametri koje možemo poslati kernel modulima te automatsko učitavanje modula.**
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**
**20.6.2.3. Rad s VLANovima pod Linuxom.**
**20.6.6.2. Konfiguracija *bondinga* u Linuxu.**
**24.2.9. Nadzor zagušenja (*Congestion control*).**
**25.2.1. *Receive Side Scaling* (*RSS*).**
**26.7.3.1. Nova funkcionalnost: *ipset*.**
**26.7.3.2. Nftables.**
**26.8.3. Intelligent Platform Management Interface (IPMI).**

**Izvori informacija: (204),(205),(206),(1115),(K-12),** `man modprobe`, `man lsmod`, `man modinfo`, `man depmod`.

### 11.1.1.3. Parametri koje možemo poslati kernel modulima te automatsko učitavanje modula

Kako smo vidjeli u prijašnjoj cjelini, neki kernel moduli imaju mogućnost definiranja određenih parametara.

**Kako definirati parametre rada kernel modulima?**
**1.** Definirati parametre možemo ručno tijekom ručnog učitavanja kernel modula, upotrebom naredbe `modprobe` prema principu:
*`modprobe IME-MODULA IME-PARAMETRA=VRIJEDNOST-PARAMETRA`*

Tako primjerice za upravljački program za *Intelovu pro 1000* mrežnu karticu (kernel modul `e1000e`), ako želimo povećati posebnu međumemoriju odnosno tzv. *buffer copybrake* na vrijednost *256*, to bi izgledalo ovako:

```
modprobe e1000e copybreak=256
```

Sve dostupne parametre za konkretni kernel modul **`e1000e`** možemo vidjeti s naredbom:

```
modinfo e1000e -p
```



Druge primjere definiranja parametara rada kernel modula možete vidjeti u poglavljima:
**12.6.2. Upotreba RAM diska.**
**13.3.2.1. NVMe diskovi i Linux.**
**14.1.4.1. Nove tehnologije na donjim slojevima I/O podsustava.**
**17.3.1. Kratka analiza problema na sustavu.**
**25.2.2. Multiqueue funkcionalnost.**
**25.5.1. Large receive offload (*LRO*) i Generic receive offload (*GRO*).**

**2.** Moguće je željene parametre pohraniti i u datoteci koja će se čitati u trenutku inicijalizacije kernel modula. Ova datoteka mora se nalaziti u direktoriju: `/etc/modprobe.d/`. Pravilo je da se za svaki kernel modul, za koji želimo trajno postaviti neki parametar, kreiramo datoteku imena istog kao i kernel modul, ali s nastavkom (ekstenzijom) `.conf`.

U gore navedenom primjeru bi stoga morali kreirati datoteku imena: `/etc/modprobe.d/e1000e.conf`, koja bi sadržavala sljedeći redak konfiguracije:

```
options e1000e copybreak=256
```

Dakle redak mora započeti s ključnom riječi `options` nakon koje slijedi ime kernel modula (u ovom slučaju `e1000e`) te dio s parametrom i njegovom vrijednosti. U ovom primjeru je to parametar `copybreak=256` koji želimo trajno postaviti.

## Druge opcije unutar datoteka, koje se nalaze u direktoriju: `/etc/modprobe.d/`

U navedenom direktoriju, moguće je kreirati i datoteke koje će imati drugačiju namjenu, osim dodavanja parametara kernel modulima, u trenutku pokretanja cijelog operativnog sustava. Moguće je:

- Započeti novi redak s ključnom riječi `alias` iza koje slijedi skraćeno ime modula, a potom stvarno ime kernel modula. Naime aliasi za kernel module su skraćena imena kernel modula, koja se također mogu koristiti, zbog lakšeg snalaženja.

  To može izgledati ovako:

```
alias gre0 ip_gre
```

Pri čemu je `gre0` *alias* na pravi kernel modul imena `ip_gre`, a ovaj konkretni modul se koristi za neke **VPN** tunele.
On nam konkretno daje funkcionalnosti za GRE protokol (*enkapsulaciju*).



Za promjene naziva uređaja, i to tijekom inicijalizacije uređaja (***udevd***); pogledajte poglavlje:
**11.1.2.1.1. *Udev* pravila (*udev rules*).**

- Započeti novi redak s ključnom riječi: `blacklist` nakon koje mora slijediti ime kernel modula za koji **ne želimo da se učita** u trenutku pokretanja operativnog sustava. Pogledajmo jedan primjer, u kojem ne želimo učitavanje kernel modula zaduženog za rad PC zvučnika; onog koji se nalazi na matičnoj ploči, a što ćemo dobiti dodavanjem:

```
blacklist pcspkr
```

Sve *aliase* i *blacklist-ane* (onemogućene) kernel module možemo izlistati s naredbom:

```
modprobe -c
```

**Izvori informacija: (207),(208),(209)**, **`man modprobe.d`**, **`man modprobe`**.

---

#### 11.1.1.3.1. Automatsko učitavanje željenih kernel modula

**Udev servis** je u Linuxu takozvani *device manager*. Naime, kad god se uređaj odnosno novi hardver spoji (inicijalizira) ili ukloni iz sustava, Linux kernel šalje takozvane *uevents* poruke. Tada aplikacija koja je upravitelja uređaja (engl. *device manager*), kao što je *udev*, upravlja ovim *udev* događajima stvaranjem posebnih datoteka (u `/dev/` direktoriju) koje predstavljaju taj uređaj te učitavanjem pripadajućih kernel modula i povezivanjem s tim datotekama. Dakle njegova svrha je otkriti i konfigurirati uređaje koji su prisutni od samog početka rada sustava pa na dalje. Na primjer, kada se operativni sustav pokreće, ***udevd*** otkriva prvo osnovne komponente hardvera računala, poput CPU-a, matične ploče i drugih ključnih komponenata, a potom i drugi hardver i pokreće odgovarajuće upravljačke programe koji se prema Linux terminologiji nazivaju *kernel moduli*.

**Kako to radi ?**
Primjerice kada se neki uređaj poveže s računalom, ***udev*** prati događaje vezane uz uređaje (npr. spajanje novog uređaja ili njegovo uklanjanje). ***Udev*** potom analizira različite identifikatore uređaja, uključujući **PCI ID** (ako je riječ o PCI uređaju), **USB ID** (ako je riječ o USB uređaju) i **DMI ID** (ako je riječ o DMI informacijama na računalu) kako bi prepoznao konkretni uređaj. Na temelju prepoznatih identifikatora i pravila definiranih u konfiguracijskim datotekama **udev** izvršava različite radnje.
To može uključivati automatsko učitavanje odgovarajućih *kernel modula* kako bi uređaj uopće bio upotrebljiv, dodjelu uređaja (npr. dodjela imena uređajima) i druge konfiguracijske radnje.
Međutim od ***Red Hat Enterprise Linux v.8.*** **(RHEL 8)** s potpunim prelaskom na *systemd* uveden je i novi servis imena: `systemd-udevd` koji je zamijenio stariji ***udevd*** servis i njegovu namjenu.
Njegova globalna konfiguracijska datoteka je: `/etc/udev/udev.conf`.



Pogledajte i poglavlje:
**11.1.1.7. Mehanizam prepoznavanja hardvera i učitavanje kernel modula.**

S druge strane, osim **udev** servisa, postoji i servis koji je zadužen za *kernel module* (upravljačke programe), a to je `kmod`.
Servis ***Kmod*** rukuje učitavanjem sâmih *kernel modula*, ali funkcionira i tako da kada god kernel pokuša pristupiti određenim vrstama resursa koji su nedostupni, on upućuje poseban sistemski poziv kernelu to jest ***kmod*** podsustavu umjesto da vrati poruku o pogrešci. Ako ***kmod*** uspije učiniti resurs (obično hardver) dostupnim, učitavanjem jednog ili više *kernel modula*, kernel nastavlja s radom; u suprotnom, vraća pogrešku. Praktično se bilo koji resurs (preko *kernel modula*) može zatražiti na ovaj način, upotreba: *karakter* ili *blok* uređaja, datotečni sustavi, mrežni protokoli ili posebni algoritmi i tako dalje.
Tako je bilo sve do ***Red Hat Enterprise Linux v.7. (RHEL 7)***

Međutim od ***Red Hat Enterprise Linux v.8.*** **(RHEL 8)**, i intenzivnijim prelaskom na *systemd* uveden je i novi servis imena: `systemd-modules-load` koji je zamijenio prethodni `kmod` servis koji se koristio u starijim verzijama *Red Hat Enterprise Linuxa* za upravljanje *kernel modulima*. To znači da je servis ***systemd-modules-load*** u *systemd* sustavu odgovoran za automatsko učitavanje *kernel modula* prilikom pokretanja računala. Ovaj servis služi za rukovanje učitavanjem *kernel modula* koji su konfigurirani za automatsko učitavanje putem datoteka (***.conf**) u direktorijima: `/etc/modules-load.d/` ili `/usr/lib/modules-load.d`.

Nakon što je sustav već pokrenut, pa na dalje, automatsko učitavanje kernel modula se u većini slučajeva odvija bez intervencije korisnika, zbog navedenih servisa koji se brinu o tome kako bi se učitao svaki potrebni kernel modul (*upravljački program*).

## Dodatne komplikacije

U nekim slučajevima, učitavanje *kernel modula* moramo ili želimo napraviti sâmi. Dodatno, u slučajevima kada želimo automatsko učitavanje *kernel modula*, koji mogu i ne moraju biti upravljački programi, već dodatne funkcionalnosti, tada unutar direktorija `/etc/modprobe.d/` možemo kreirati običnu tekstualnu datoteku ekstenzije `.conf` i u nju ubacivati, redak po redak, željene *kernel module* za koje želimo da se učitavaju tijekom pokretanja operativnog sustava.
Za primjer, možemo kreirati datoteku: `/etc/modprobe.d/nasi-moduli.conf` i u nju prvo dodati primjerice upravljački program za našu *ASIX USB* mrežnu karticu. Tada bi ova datoteka izgledala ovako:

```
# USB Mrezna
asix
```

Potom u novi redak možemo dodati modul za recimo *Linux vatrozid (firewall)* [*iptables]*, a pomoću kojeg ćemo dobiti funkcionalnost vatrozida (*firewall*) pod Linuxom:

```
# Mrezni firewall
ip_tables
```

Redci koji počinju sa znakom `#` se ignoriraju, odnosno služe samo kako komentar.

U ***Red Hat*** baziranim Linuxima, postoji još jedan direktorij, u koji možemo ubacivati datoteke, koje sadrže *kernel modul*, za koje želimo da se učitaju tijekom pokretanja operativnog sustava, ali prema principu *shell* skripte.
Radi se o direktoriju `/etc/sysconfig/modules/` u kojem možemo kreirati datoteku bilo kojeg imena, ali isključivo ekstenzije `.modules`.

Ova datoteka mora zadovoljiti nekoliko uvjeta:

1. Prvi red datoteke mora sadržavati: `#!/bin/bash` ili neku drugu ljusku odnosno *shell* (**zsh**/**csh**/**sh** …).
2. Svaki novi redak u datoteci mora biti pisan kao da se radi o *shell* skripti, jer ovo i je *shell* skriptna datoteka.
3. Ova datoteka mora biti izvršna (mora imati postavljen ***x*** *bît* u ovlastima), primjerice s naredbom `chmod`:

```
cd /etc/sysconfig/modules/
chmod +x IME-DATOTEKE.modules
```

Ova datoteka može u potpunosti izgledati ovako, ako u nju želimo upisati iste stvari kao iz primjera sa stranice prije:

```
#!/bin/bash

# Ucitajmo USB ASIX mreznu karticu (driver)
/sbin/modprobe asix

# Ucitajmo kernel modul za firewall
/sbin/modprobe ip_tables
```

U direktoriju: `/etc/sysconfig/modules/` kreiramo datoteke **sâmo**, ako želimo postaviti neke uvjete, petlje ili slično, a koje možemo riješiti samo sa *shell* skriptom, a ne sâmim učitavanjem *kernel modula* (kao na primjeru sa stranice prije).



Standardno mjesto sâmo za učitavanje kernel modula je direktorij: `/etc/modprobe.d/`

Konkretan primjer upotrebe učitavanja *kernel modula* baš u ovom direktoriju bi mogla biti datoteka koja sadrži:

```
#!/bin/bash

if [ ! -c /dev/input/uinput ] ; then
        exec /sbin/modprobe uinput >/dev/null 2>&1
fi
```

U ovom primjeru se provjerava postoji li uređaj: `/dev/input/uinput`, a ako ne postoji tada se pokreće naredba: `/sbin/modprobe uinput` koja učitava *kernel modul* imena `uinput`.

**Izvori informacija: (210),(211),(1552), `man modprobe`,`man modprobe.d`,`man chmod`, `man systemd-modules-load`,`man modules-load.d`.**

### 11.1.1.3.2. Dodavanje novih kernel modula

Ako nam nedostaje neki upravljački program (*kernel modul*), moguće je naći izvorni kôd upravljačkog programa te ga kompilirati i instalirati. Ovaj proces ćemo objasniti malo kasnije, ali u slučaju kada ste to već napravili, novi upravljački program odnosno kernel modul će se sâm ubaciti negdje unutar strukture direktorija `/lib/modules/$(uname -r)/` te će se upisati/nadodati u datoteku: `modules.dep` odnosno u datoteku na lokaciji: `/lib/modules/$(uname -r)/modules.dep`

Ova datoteka je zapravo sâmo lista s popisom dostupnih *kernel modula* i putanjom do njih.

U slučaju kada smo recimo morali kompilirati i dodati *kernel modul* imena `asix.ko`, on bi u datoteku: `/lib/modules/$(uname -r)/modules.dep` dodao sljedeći redak:

```
kernel/drivers/net/usb/asix.ko: kernel/drivers/net/usb/usbnet.ko
kernel/drivers/net/mii.ko
```

Vidljivo je da osim putanje do sâmog *kernel modula*, slijede putanje do *kernel modula* o kojima on ovisi odnosno koji se moraju učitati prije njega.

Nakon toga potrebno je osvježiti bazu svih *kernel modula* s naredbom `depmod` na sljedeći način:

```
depmod -a
```

I to je to, novi *kernel modul* je spreman za buduće korištenje.

**Izvori informacija (212),(213),(726),(1028), `man depmod`,`man kmod`,`man yum`.**

#### 11.1.1.3.3. Kernel moduli (upravljački programi) koje je potrebno dodatno instalirati

Postoje i posebni slučajevi kada određeni kernel moduli ne dolaze već instalirani sa sustavom ili se ne nalaze unutar kernela niti kao kernel moduli. Ovakav primjer su upravljački programi za grafičke kartice poput ***Nvidia***, neke mrežne kartice, posebne funkcionalnosti i slično. Ovdje obično imamo dvije mogućnosti: sâmi kompilirati kernel modul ili ga instalirati kako dodatni softverski (***rpm***) paket. U slučaju kada smo se odlučili za instalaciju i već nam je softverski paket koji sadrži naš kernel modul dostupan u repozitoriju koji smo već dodali, ponovno imamo dvije opcije. Koristiti standardni softverski paket koji sadrži naš kernel modul ili koristiti takozvanu `kmod` varijantu softverskog (*rpm*) paketa o kojoj ćemo ovdje govoriti. Naime `kmod` je naredba i komponenta sustava koja se brine o tome da kada se instalira ili nadograđuje kernel modul, a potom u bilo kojem trenutku i nadograđuje cijeli sustav s novim kernelom, da se ponovno nadogradi i instalira i (naš) potrebni kernel modul.

Naime, ako recimo instaliramo ***Nvidia*** upravljačke programe za grafičku karticu (a to su kernel moduli) za naš trenutni radni kernel, a potom (bilo kada) nadogradimo kernel na novu inačicu, morali bi ponovno ručno instalirati nove upravljačke programe za našu grafičku karticu. Stoga, kako ovo ne bi morali stalno raditi nakon svake nadogradnje kernela, možemo koristiti posebnu inačicu upravljačkog programa, koji će automatski biti nadograđen kada nadogradimo i naš kernel.

Ove inačice upravljačkih programa (kernel modula) u nazivu obično imaju naziv ***kmod***-*. Konkretno bi to za *Nvidia* grafičku karticu to bio (*rpm*) softverski paket imena `kmod-nvidia`.
U slučaju kada imamo ***Nvidia*** grafičku karticu, možemo ju instalirati s naredbom:

```
yum install kmod-nvidia
```

I sada smo sigurni, da će se sa svakom nadogradnjom kernela nadograditi i upravljački program za našu grafičku karticu.

**Vratimo se na početak ove priče odnosno dio priče o kompiliranju upravljačkih programa (kernel modula)**
U slučajevima kada vam nedostaje unaprijed napravljeni upravljački program (*kernel modul*), na sustavu se nalazi zastarjela inačica upravljačkog programa ili postoji bilo koji drugi razlog zbog kojega želite sami kompilirati upravljački program, postoje dvije opcije kako to napraviti.

**1.** Prva opcija je ručno kompiliranje *kernel modula* za trenutnu inačicu kernela i instaliranje na sustav. Ne zaboravimo da Linux namjerno ne pruža stabilan **API** ili **ABI** za *kernel module* odnosno upravljačke programe. Stoga se upravljački programi moraju kompilirati za točno određenu inačicu kernela!



Vezano za **API** i **ABI** kompatibilnost i *kernel module*, pogledajte poglavlje:
**11.1.1.2. Napredno: Rad s kernel modulima.**

Posebno pogledajte cjelinu: „***Binarna kompatibilnost kernel modula (upravljačkih programa)***“

**Međutim, što će se dogoditi s upravljačkim programom ako kasnije nadogradite sustav na noviji kernel?**
Ne zaboravimo da se prilikom kompiliranja upravljačkog programa vežemo za kernel zaglavlja (engl. *kernel headers*) od trenutnog kernela za koji to radimo. Kernel zaglavlja sadrže vitalne informacije o programskim sučeljima, strukturama podataka i definicijama koje se koriste unutar konkretne inačice Linux kernela.

Dakle potrebno je imati identičnu inačicu kernel zaglavlja koja odgovaraju inačici kernela na kojem namjeravate koristiti konkretan upravljački program odnosno *kernel modul*.

Za početak instalirajmo potrebne kernel pakete (kernel zaglavlja, program za kompiliranje, …):

```
yum -y install kernel kernel-devel gcc make
```

U ovim uputama koristit ćemo upravljački program za 10Gbps mrežne kartice tvrtke *Intel*: upravljački program `ixgbe` inačice 5.19.9. koji ćemo kompilirati za naš kernel v.`5.14.0-362.18.1`.
Sada je potrebno izvorni programski kôd upravljačkog program kopirati u trenutni direktorij u kojem se nalazimo.
U našem slučaju, pošto smo korisnik ***root*** (*zbog potrebnih prava rada*), nalazimo se u direktoriju `/root` te izvršavamo:

```
wget http://tenet.dl.sourceforge.net/project/e1000/ixgbe%20stable/5.19.9/ixgbe-5.19.9.tar.gz
```

Odnosno s gornjom naredbom smo kopirali izvorni programski kôd upravljačkog programa `ixgbe`.

Sada dekomprimirajmo ovu arhivsku datoteku s izvornim programskim kodom (i dalje se nalazimo u direktoriju `/root/`):

```
tar -xf ixgbe-5.19.9.tar.gz
```

Potom uđimo u ovaj novi direktorij:

```
cd ixgbe-5.19.9/src
```

Iz ovog direktorija (`/root/ixgbe-5.19.9/src/`) pokrenimo kompiliranje upravljačkog programa (*kernel modula*):

```
make
```

Nakon nekog vremena dobit ćemo kompilirani upravljački program (*kernel modul*) imena `ixgbe.ko` u trenutnom direktoriju.

Pogledajmo što smo dobili:

```
modinfo ixgbe.ko
filename:       /root/ixgbe-5.19.9/src/ixgbe.ko
version:        5.19.9
license:        GPL
description:    Intel(R) 10GbE PCI Express Linux Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
rhelversion:    9.3
...
```

← vidimo da je to naš novi upravljački program (`version: 5.19.9`)
Odnosno možemo i instalirati ovaj *kernel modul* u strukturu trenutnog kernela:`/lib/modules/5.14.0-362.18.1.el9_3.0.1.x86_64/kernel/drivers/net/ethernet/intel/ixgbe/`. Stoga pokrenimo:

```
make install
```

Dodatno još možemo prekopirati *man* stranicu za ovaj *kernel modul* u područje s *man* stranicama:

```
cp /root/ixgbe-5.19.9/ixgbe.7 /usr/share/man/man7/
```

Sada, ako pokrenemo:

```
man ixgbe
```

← dobivamo detaljne *man* stranice za ovaj upravljački program, koje opisuju detalje upravljačkog programa i sve njegove dostupne parametre i opcije.

## Dynamic kernel module support (DKMS)

2. **Opcija dva: automatska izrada kernel modula pomoću DKMS mehanizna**
Sada ćemo se fokusirati na drugu i napredniju opciju upotrebom tzv. *dynamic kernel module support* (**DKMS**) mehanizma. Programsko sučelje *DKMS* omogućuje nam automatsku ponovnu izradu upravljačkih programa (*kernela modula*) u trenutno stablo kernela tijekom svake nadogradnje kernela. Dodatno, mnogi dobavljači hardvera često distribuiraju upravljačke programe za svoj hardver kao *DKMS* paket, tako da korisnici mogu automatski ažurirati instalirani upravljački program dok sustav nadograđuju na noviju inačicu kernela.

Kao krajnji korisnik možemo jednostavno konfigurirati *DKMS* za automatsko ažuriranje bilo kojeg prilagođenog upravljačkog programa koji se nalazi izvan osnovnog stabla kernela odnosno koji nije uključen u naš trenutni kernel.

Za to prvo moramo instalirati **EPEL** repozitorij:

```
yum -y  install epel-release
```

Potom trebamo instalirati *DKMS* komponente, koje obuhvaćaju: *header* datoteke za naš trenutni kernel, izvorni kôd kernela i druge elemente poput osnovnih paketa potrebnih za kompiliranje upravljačkih programa:

```
yum -y install dkms
```

Za svaki slučaj instalirajmo i potrebne *kernel* pakete:

```
yum -y install kernel kernel-devel
```

Nakon instalacije, još provjerimo našu inačicu kernela:

```
uname -r
5.14.0-362.18.1.el9_3.0.1.x86_64
```

Zatim ju usporedimo s novo instaliranim *kernel header* datotekama:

```
rpm -qa |grep kernel-headers
kernel-headers-5.14.0-362.18.1.el9_3.0.1.x86_64
```

← vidimo da se inačice podudaraju: `5.14.0-362.18.1` ←→ `kernel-headers-5.14.0-362.18.1`.

I u ovoj opciji kompiliranja pomoću DKMS sustava također ćemo koristiti upravljački program za 10Gbps mrežne kartice tvrtke *Intel* (upravljački program `ixgbe` inačice 5.19.9), na istoj inačici kernela. Sada je potrebno izvorni programski kôd kopirati u trenutni direktorij u kojem se nalazimo. Pošto smo korisnik ***root***, nalazimo se u direktoriju `/root` te izvršavamo:

```
wget http://tenet.dl.sourceforge.net/project/e1000/ixgbe%20stable/5.19.9/ixgbe-5.19.9.tar.gz
```

Sada dekomprimirajmo ovu arhivu s izvornim programskim kodom upravljačkog programa, u traženi odredišni direktorij:

```
tar -xf ixgbe-5.19.9.tar.gz
mv /root/ixgbe-5.19.9/src/ /usr/src/ixgbe-5.19.9
```

Dakle sadržaj arhive smo prekopirali o odredišni direktorij: **`/usr/src/ixgbe-5.19.9`**. I sada u tom odredišnom direktoriju moramo napraviti *DKMS* konfiguracijsku datoteku imena `dkms.conf`, što ćemo napraviti pomoću programa `vi`:

```
vi /usr/src/ixgbe-5.19.9/dkms.conf
```

U *DKMS* konfiguracijsku datoteku, za konkretni *kernel modul* moramo napraviti inicijalnu *DKMS* konfiguraciju.

Dakle u nju ubacujemo sljedeći kôd. U uređivaču teksta `vi` stisnite malo slovo **i** te ubacite sljedeći kôd:

```
PACKAGE_NAME="ixgbe"
PACKAGE_VERSION="5.19.9"
BUILT_MODULE_NAME[0]="ixgbe"
DEST_MODULE_LOCATION[0]="/kernel/drivers/net/ethernet/intel/ixgbe/"
AUTOINSTALL="yes"
```

Zatim snimite novi sadržaj u ovu datoteku, što postižemo tako da stisnemo dvotočku **:** a onda upisujemo **wq!** i *ENTER*)

Potom s naredbom `dkms` dodajemo ovaj budući *kernel modul* u DKMS okruženje:

```
dkms add -m ixgbe -v 5.19.9
```

Završavamo s kompiliranjem novog *kernel modula* iz izvornog programskog koda, također pomoću `dkms` naredbe:

```
dkms build -m ixgbe -v  5.19.9
```

Kada se proces kompiliranja završi, sve je spremno za instalaciju ovog novo kompiliranog *kernel modula*, pa nam ostaje još jedan korak, a to je instalacija i priprema sustava za novi *kernel modul*.

Instalaciju postižemo sa:

```
dkms install -m ixgbe -v 5.19.9

ixgbe.ko.xz
Running module version sanity check.
- Original module
- Found /lib/modules/5.14.0-362.18.1.el9_3.0.1.x86_64/kernel/drivers/net/ethernet/intel/ixgbe/ixgbe.ko.xz
  - Storing in /var/lib/dkms/ixgbe/original_module/5.14.0-362.18.1.el9_3.0.1.x86_64/x86_64/
  - Archiving for uninstallation purposes
- Installation
  - Installing to /lib/modules/5.14.0-362.18.1.el9_3.0.1.x86_64/extra/
Adding any weak-modules
depmod......
```

🡐 Vidimo da je novi *kernel modul* instaliran te da je na kraju pokrenuta `depmod` procedura, što znači da je sve uredno završilo.



Svaka naknadna nadogradnja kernela pokrenut će ***DKMS*** mehanizme koji će ponovno kompilirati i instalirati ovaj kernel modul u stablo novoinstaliranog kernela, i to bez naše intervencije!

**Izvori informacija (212),(213),(726),(1028),(1592),(1593),(1594), `man depmod`,`man kmod`,`man yum`, `man modinfo`,`man make`,`man uname`,`man rpm`,`man vi`,`man dkms`.**

### 11.1.1.4. Kompiliranje novog kernela

U radu s Linuxom, ponekad imamo i potrebu instalirati (naj)noviji kernel, zbog:

- Novih, bržih i/ili stabilnijih upravljačkih programa.
- Novih, poboljšanih ili optimiziranijih funkcionalnosti koje nam do tada nisu bile dostupne (u staroj inačici).
- Ili zbog poboljšanih, optimiziranih ili dodanih sigurnosnih parametara ili nadogradnji.

Stoga imamo dvije mogućnosti: ili instalirati zadnji dostupni službeni kernel od distribucije linuxa koju koristimo (pr. *CentOS* 7.x/8.x) ili ćemo sami kopirati zadnju stabilnu inačicu Linux kernela i sâmi ga kompilirati, što će biti slučaj u ovom poglavlju.

**Ovdje ćemo dati primjere za CentOS 7.x/8.x (64.bitnu inačicu), dok je za CentOS 6.x sve vrlo slično.**



Moramo biti sigurni da imamo barem minimalno 15 GB slobodnog prostora na disku.

Prvo napravimo nadogradnju sustava na zadnju inačicu, sa sljedećom naredbom:

```
yum update -y
```

Instalirajmo razvojne (*developerske*) alate i programske jezike, pomoću naredbe koja slijedi:

```
yum groupinstall 'Development Tools' -y
```

Instalirat ćemo i dodatne softverske pakete:

```
yum install -y ncurses-devel make bc elfutils-libelf-devel openssl-devel grub2 wget
```

Sada, kada je sve spremno, ponovno pokrenimo (*restartajmo*) računalo:

```
reboot
```

Pogledajmo koju inačicu kernela sada imamo, sa sljedećom naredbom:

```
uname -r
3.10.0-957.1.3.el7.x86_64
```

I provjerimo koju inačicu ***CentOS*** distribucije Linuxa imamo:

```
cat /etc/redhat-release
CentOS Linux release 7.6.1810 (Core)
```

Dakle inačica je 7.6 i pôd inačica 1810.

Mogli bismo instalirati i izvorni (*source*) kôd trenutnog kernela (za ***CentOS*** 7.6.1810) koji trenutno koristimo:

```
rpm -Fivh  http://vault.centos.org/7.6.1810/updates/Source/SPackages/kernel-3.10.0-957.1.3.el7.src.rpm
```

Međutim mi ćemo se odlučiti za upotrebu zadnje stabilne inačice 4.x linux kernela (konkretno v. 4.20.3), koju ćemo kopirati s internet adrese odnosno stranice https://cdn.kernel.org/ na sljedeći način:

```
cd /usr/src/
wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.20.3.tar.gz
```

Sada dekomprimirajmo izvorni (*source*) kôd novog kernela koji smo upravo kopirali. To ćemo napraviti ovako:

```
tar -xvf linux-4.20.3.tar.gz
```

Potom možemo obrisati komprimiranu datoteku kako nam ne bi zauzimala mjesto na disku:

```
rm -f linux-4.20.3.tar.gz
```

Zatim uđimo u novi raspakirani direktorij (mapu) koji sadrži izvorni (*source*) kôd novog kernela:

```
cd linux-4.20.3
```

Kopirajmo trenutnu konfiguraciju kernela kao inicijalnu za naš novi kernel; stoga kako bismo zadržali sve opcije i kernel module koje i sada koristimo u trenutnom kernelu inačice 3.10. Sustav će se pri tome pobrinuti da nove opcije i kernel moduli budu dodani u trenutku snimanja nove konfiguracije. Prvo krenimo s kopiranjem ove datoteke na željeno odredište (direktorij/mapu):

```
cp /boot/config-`uname -r` /usr/src/linux-4.20.3/.config
```

I sada pokrenimo rekonfiguraciju kernela, upotrebom naredbe `make` na sljedeći način:

```
make menuconfig
```



*Kod rekonfiguracije opcija kernela treba biti oprezan kako ne bismo uključili previše toga u sâm kernel, jer će postati prevelik i potencijalno usporavati sustav ili će eventualno biti nestabilan. Naime ovdje se radi o kompromisu ili balansu između uključenih funkcionalnosti i upravljačkih programa u sâm kernel odnosno s druge strane uključivanje istih kao kernel modula to jest vanjskih datoteka u odnosu na manji kernel s uključenim samo nužnim stvarima. Stoga je često dobro rješenje koristiti konfiguraciju koja dolazi s našom distribucijom linuxa, kako smo i napravili, te ju po potrebi sâmo malo promijeniti ili optimizirati.*

***U sljedećim koracima, dodat ćemo podršku za tzv. I/O(disk) scheduler mehanizam, naziva BFQ, u naš novi kernel.*** Nakon par trenutaka ćemo dobiti izbornik menija za konfiguraciju kernela, vidljivog na slici 83:

Slika 83. Izbornik za konfiguraciju kernela.

```
.config - Linux/x86 4.20.3 Kernel Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
 submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
 includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
 exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
        *** Compiler: gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36) ***
            General setup  --->
        [*] 64-bit kernel
            Processor type and features  --->
            Power management and ACPI options  --->
            Bus options (PCI etc.)  --->
            Binary Emulations  --->
            Firmware Drivers  --->
        [*] Virtualization  --->
            General architecture-dependent options  --->
       <Select>    < Exit >    < Help >    < Save >    < Load >
```

Ovdje se krećemo s kursorskim tipkama i s *ENTER* ulazimo u izbornike, a s odabirom **<Exit>** izlazimo u izbornik iznad.
S razmaknicom (tipka „*space*") ili sa odabirom **<Select>** biramo:

- Hoće li određena funkcionalnost biti ugrađena u kernel (oznaka *).
- Ili će biti dostupna kao kernel modul odnosno datoteka (oznaka **M**).
- Ili određena funkcionalnost NEĆE uopće biti dostupna ( ) prazno mjesto; bez oznake.

Mi želimo uključiti podršku za I/O(disk) *schedule*r **BFQ** (uz sve postojeće *I/O schedulere*) pa odaberimo: **IO Schedulers -->**" i zatim na dnu odabiremo: **BFQ I/O scheduler (NEW)** i označimo ga kao kernel modul **<M>**.
Potom uključimo podršku za **BFQ hierarchical scheduling support**, kao što je vidljivo na slici 84.

Slika 84. Pôdizbornik za konfiguraciju kernela, za BFQ dio.

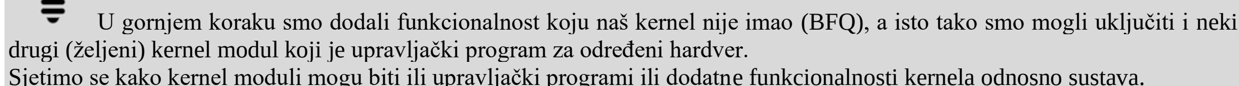
```
.config - Linux/x86 4.20.3 Kernel Configuration
> IO Schedulers
                          IO Schedulers
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
 submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
 includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
 exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
       <*> Deadline I/O scheduler
       <*> CFQ I/O scheduler
       [*]   CFQ Group Scheduling support
           Default I/O scheduler (Deadline)  --->
       <*> MQ deadline I/O scheduler
       <*> Kyber I/O scheduler
       <M> BFQ I/O scheduler
       [*]   BFQ hierarchical scheduling support
       <Select>    < Exit >    < Help >    < Save >    < Load >
```

Sada se vratimo na vršni izbornik **<Exit>**,**<Exit>** te ćemo dobiti pitanje želimo li snimiti novu konfiguraciju kernela:

```
 Do you wish to save your new kernel configuration?
 <ESC><ESC> to continue.
                     < Yes >        <  No  >
```

Slika 85. Izbornik za potvrdu snimanja nove inačice konfiguracije kernela

Odabiremo **<Yes>** jer želimo snimiti konfiguraciju za naš novi (budući) kernel.



U gornjem koraku smo dodali funkcionalnost koju naš kernel nije imao (BFQ), a isto tako smo mogli uključiti i neki drugi (željeni) kernel modul koji je upravljački program za određeni hardver.
Sjetimo se kako kernel moduli mogu biti ili upravljački programi ili dodatne funkcionalnosti kernela odnosno sustava.

Sada će nova konfiguracija kernela biti snimljena u datoteku `/usr/src/linux-4.20.3/.config`.
Dok će stara konfiguracija biti zapisana kao datoteka `/usr/src/linux-4.20.3/.config.old`.
Mogli smo i eventualno u slučaju, da smo sâmo trebali kernel module za već prije kompilirani kernel, kompilirati sâmo kernel module.
To bi postigli na sljedeći način:

```
make modules
```

I potom ih možemo instalirati na sustav:

```
make modules_install
```

Mi ćemo ipak, pošto tek instaliramo novi kernel, kompilirati prvo novi kernel, a tek potom sve njegove pripadajuće kernel module.
Dakle prvo kompilirajmo kernel sa sljedećom naredbom:

```
make bzImage
```

Ako imamo i želimo koristiti više od jedne CPU jezgre (primjerice 4) tada umjesto navedene naredbe možemo pokrenuti:

```
make -j4 bzImage
```

Ovaj korak će potrajati ovisno o brzini računala; minimalno nekoliko minuta do nekoliko sati.

I na kraju ćemo dobiti poruku poput:
```
Setup is 16844 bytes (padded to 16896 bytes).
System is 7701 kB
CRC ef56271a
Kernel: arch/x86/boot/bzImage is ready  (#1)
```
Rezultat kompiliranja je kernel (datoteka) imena `bzImage`, koja se nalazi u direktoriju (mapi):
`/usr/src/linux-4.20.3/arch/x86/boot/`

Potom trebamo kompilirati i sve kernel module odnosno upravljačke programe i posebne funkcionalnosti za naš novi kernel:
```
make modules
```

I ovo će potrajati, nekoliko desetaka minuta ili znatno duže. Međutim i ovdje, ako imamo i želimo koristiti više od jedne CPU jezgre (primjerice 4) za kompiliranje, tada umjesto navedene naredbe možemo pokrenuti sljedeću naredbu:
```
make -j4 modules
```
Sada kada imamo i novo kompilirani kernel i pripadajuće kernel module, možemo krenuti dalje.

U slučaju kada nam nešto nije dobro kompilirano ili kada smo imali greške koje trebamo ispraviti i ponovno sve kompilirati, potrebno je prvo očistiti sve privremeno generirane datoteke. Čišćenje privremeno kreiranih datoteka odrađujemo s naredbom:
```
make clean
```
S ovom naredbom se brišu sve privremeno kreirane datoteke, ali se ne diraju konfiguracijske datoteke, tako da je nakon ovog koraka moguće kompilirati kernel module ili sâm kernel. A tek tada možemo nastaviti dalje s kompiliranjem ili rekonfiguracijom pa ponovnim kompiliranjem kernela ili kernel modula.

Za novije kernele postoji i mogućnost u jednom koraku kompilirati i kernel i kernel module, što možemo napraviti sa sljedećom naredbom:
```
make
```

Međutim i ovdje, ako imamo i želimo koristiti više od jedne CPU jezgre (primjerice 4) za kompiliranje tada umjesto navedene naredbe možemo pokrenuti i naredbu:
```
make -j4
```

U svakom slučaju, sada kada imamo i kompilirani kernel i kernel module, potrebno je instalirati ih.
Prvo instalirajmo naše novo kompilirane kernel module:
```
make modules_install
```
U ovom koraku sustav je kreirao stablo direktorija: `/lib/modules/4.20.3/` u koji je stavio cijelu strukturu poddirektorija sa svim kernel modulima i pripadajućim konfiguracijskim datotekama. Pri tome su kernel moduli pozicionirani unutar pôddirektorija: `/lib/modules/4.20.3/kernel/drivers/`.

Potom krećemo na kernel i pripadajuće datoteke. U ovom koraku moramo pokrenuti sljedeću naredbu:
```
make install
```
Ova naredba će malo potrajati, a ona će pripremiti sustav za instalaciju novog kernela i pripadajućih datoteka te konfigurirati *boot loader* (u našem slučaju je to konkretno ***grub2***). Za ***boot loader*** pogledajte poglavlje: **11.1.1.5.**

Što se tiče novih datoteka u ovom koraku, sustav će kreirati sljedeće datoteke:

- `initramfs-4.20.2.img` → ovo je novo generirani inicijalni RAM disk (engl. *Initial RAM disk*) koji se učitava tijekom pokretanja sustava, a prije učitavanje sâmog kernela. Pogledajte poglavlje: **12.6.2. Upotreba RAM diska.**
- `vmlinuz-4.20.2` → ovo je naš novo kompilirani kernel v.4.20.3.
- `System.map-4.20.2` → ovo je datoteka s tablicom sistemskih simbola za naš novi kernel, a nužna za rad (novog) kernela.

Sve navedene datoteke bit će kopirane u `/boot` direktorij gdje se i trebaju nalaziti i gdje se nalaze datoteke nužne za pokretanje sustava (kao i svi stari *kerneli* i njihove pripadajuće datoteke). Pogledajmo što sada imamo u ovom direktoriju:
```
ls -al /boot/*4.20.3*

-rw-------. 1 root root 37727660 Jan 23 13:07 /boot/initramfs-4.20.3.img
-rw-r--r--. 1 root root  3695824 Jan 23 13:03 /boot/System.map-4.20.3
-rw-r--r--. 1 root root  7902592 Jan 23 13:03 /boot/vmlinuz-4.20.3
```
Dakle sve tri datoteke su tu. Dodatno, ovaj proces je promijenio i dva simbolička linka u `/boot` direktoriju:

- `System.map` → `/boot/System.map-4.20.3` - *System.map* pokazuje na naš novi *System.map* (4.20.3).
- `vmlinuz` → `/boot/vmlinuz-4.20.3` - *vmlinuz* (kernel) pokazuje na naš novi kernel (4.20.3).

Kako bi sve bilo po pravilima, kopirajmo i trenutnu konfiguraciju našeg novog kernela, s logičnim i deskriptivnim imenom, u `/boot/` direktorij, na sljedeći način:
```
cp /usr/src/linux-4.20.3/.config /boot/config-4.20.3.el7.x86_64
```



Nakon kompiliranja novog kernela, vrlo vjerojatno ćete morati rekreirati i ***initramfs***; stoga pogledajte poglavlje:
**10.7.1.2.1. initrd i/ili initramfs i inicijalizacija sustava.**

**Tablica svih podržanih simbola (Symvers)**
I još nam ostaje tablica (inačica) simbola za kernel i kernel module, koja se nalazi u datoteci: `Module.symvers`.
Nju ćemo sâmo komprimirati i snimiti u novu datoteku u `/boot/` direktoriju, isto s deskriptivnim (*opisnim*) nazivom:

```
gzip -c /usr/src/linux-4.20.3/Module.symvers > /boot/symvers-4.20.3.el7.x86_64.gz
```

Kako smo rekli, u zadnjem procesu instaliranja kernela, kreiran je i novi unos za *boot loader* (***grub2***), kako bi se tijekom pokretanja sustava/računala mogao odabrati odnosno učitati novi kernel (uz sačuvane starije dvije inačice kernela).
Dakle sustav je otvorio datoteku `/etc/grub2.cfg` i dodao novu sekciju koja započinje otprilike ovako (skraćeno):

```
menuentry 'CentOS Linux (4.20.3) 7 (Core)' --class centos --class gnu-linux -c
. . .
```

U ovoj konfiguracijskoj datoteci (`/etc/grub2.cfg`) su definirani svi unosi potrebni za učitavanje (svakog) novog kernela.
Dakle sada imamo konkretno četiri (4) kernela s kojima možemo pokrenuti sustav, što je vidljivo kao:

```
grep "menuentry '" /etc/grub2.cfg
0 → menuentry 'CentOS Linux (4.20.3) 7 (Core)' --class centos --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-3.10.0-957.el7.x86_64-advanced-300463cf-ec75-4079-a029-19e2aecd430a' {

1 → menuentry 'CentOS Linux (3.10.0-957.1.3.el7.x86_64) 7 (Core)' --class centos --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-3.10.0-957.el7.x86_64-advanced-300463cf-ec75-4079-a029-19e2aecd430a' {

2 → menuentry 'CentOS Linux (3.10.0-957.el7.x86_64) 7 (Core)' --class centos --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-3.10.0-957.el7.x86_64-advanced-300463cf-ec75-4079-a029-19e2aecd430a' {

3 → menuentry 'CentOS Linux (0-rescue-bf13e4561d7846329f8ddde1e8a195c1) 7 (Core)' --class centos --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-0-rescue-bf13e4561d7846329f8ddde1e8a195c1-advanced-300463cf-ec75-4079-a029-19e2aecd430a' {
```

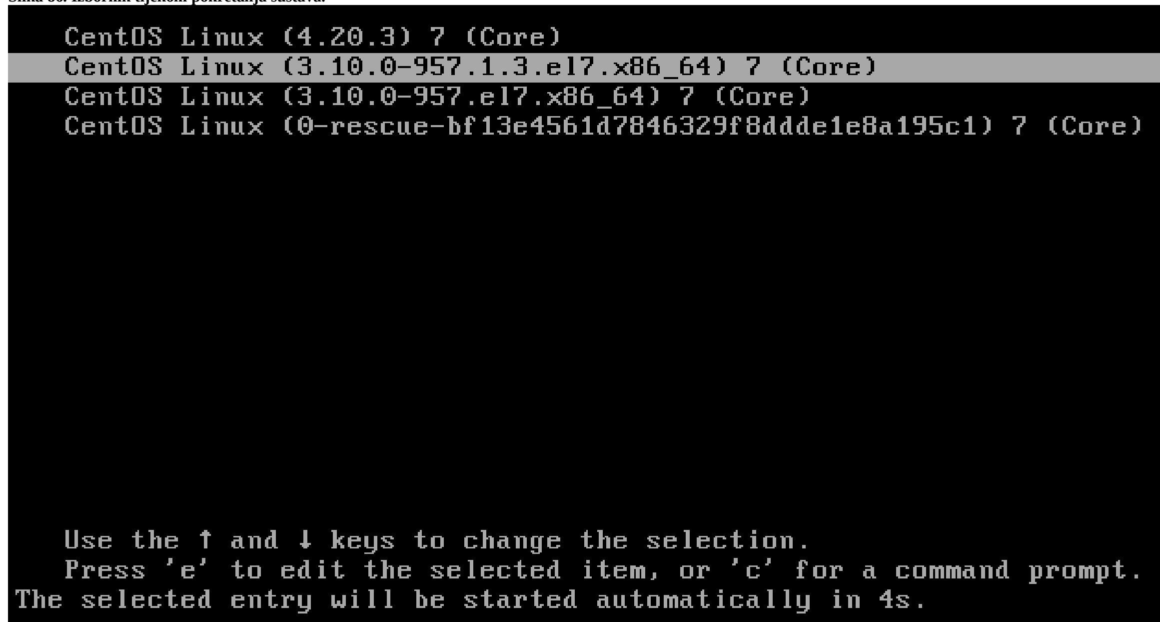
Ove sekcije u `/etc/grub2.cfg` datoteci imaju identifikatore od nula (**0**) pa na dalje.
Sličan ispis, ali s vidljivim identifikatorima, možemo vidjeti s naredbom `grubby` na sljedeći način:

```
grubby --info=ALL
```

Dakle naš unos za novi kernel ima identifikator nula (**0**). Međutim naš sustav će pokretati stari kernel odnosno unos s identifikatorom jedan (**1**), vidljiv kao: **`'CentOS Linux (3.10.0-957.1.3.el7.x86_64) 7 (Core)'`**.

Sada ćemo tijekom pokretanja računala imati sljedeći izbornik.
Dakle izbornik koji se pojavljuje u trenutku pokretanja sustava (računala) će izgledati ovako (slika 86):

**Slika 86. Izbornik tijekom pokretanja sustava.**




Vidimo kako ćemo i dalje imati automatski odabran i pokrenut stari kernel. Sada se logirajmo na sustav i promijenimo kernel koji se automatski pokreće, na naš novi, s identifikatorom nula (`0`).

To ćemo postići s naredbom `grub2-set-default`:

```
grub2-set-default 0
```

I sada ponovno pokrenimo sustav pa ćemo vidjeti kako je naš novi kernel odabran za automatsko pokretanje sustava:

**Slika 87. Izbornik tijekom pokretanja sustava.**

```
    CentOS Linux (4.20.3) 7 (Core)
    CentOS Linux (3.10.0-957.1.3.el7.x86_64) 7 (Core)
    CentOS Linux (3.10.0-957.el7.x86_64) 7 (Core)
    CentOS Linux (0-rescue-bf13e4561d7846329f8ddde1e8a195c1) 7 (Core)









    Use the ↑ and ↓ keys to change the selection.
    Press 'e' to edit the selected item, or 'c' for a command prompt.
 The selected entry will be started automatically in 2s.
```

Vidimo kako će se pokrenuti naš novi kernel inačice **4.20.3.** Sada se logirajmo na sustav i pokrenimo sljedeću naredbu:

```
uname -r

4.20.3
```

Dakle vidimo kako se stvarno koristi novi kernel `4.20.3`.

I na kraju provjerimo imamo li ***BFQ*** kernel modul, koji smo željeli imati, a koji smo sâmi kompilirali (skratili smo ispis):

```
modinfo bfq

filename:       /lib/modules/4.20.3/kernel/block/bfq.ko
description:    MQ Budget Fair Queueing I/O Scheduler
alias:          bfq-iosched
name:           bfq
```

Vidimo kako je kernel modul **BFQ** ovdje i kako se nalazi u direktoriju: `/lib/modules/4.20.3/kernel/block/bfq.ko`

### Što ako nakon svake nadogradnje sustava želimo uvijek imati zadnju dostupnu inačicu Linux kernela ?

U navedenom slučaju potrebno je napraviti nekoliko sljedećih koraka.

Prvo trebamo dodati novi repozitorij ***ELRepo*** na kojem se konstantno kompiliraju zadnje stabilne inačice Linux kernela:

```
yum -y install elrepo-release
```

Potom uredite datoteku: `/etc/yum.repos.d/elrepo.repo` te u sekciji `[elrepo-kernel]` promijenite opciju s `enabled=0` na `enabled=1`. S time smo omogućili upotrebu najnovijih inačica Linux kernela. Potom pokrenite naredbu:

```
yum -y install kernel-ml
```

Ovim korakom smo instalirali tzv. *Mainline Kernel* (stabilni). Naime nakon toga ćete dobiti zadnju stabilnu inačicu Linux kernela, te ćete i nakon svake nadogradnje sustava s `yum update`, uvijek imati zadnju stabilnu inačicu Linux kernela.
U slučaju kada se sustav nakon restarta automatski pokreće sa starim kernelom, to je moguće promijeniti. Sada se logirajmo na sustav i promijenimo kernel koji se automatski pokreće, na naš najnoviji, koji obično ima identifikator nula (`0`).
To ćemo postići s naredbom `grub2-set-default`:

```
grub2-set-default 0
```

***Ukoliko vam se sustav ne pokreće s novim kernelom ili imate dodatna pitanja, pogledajte sljedeće poglavlje.***



*Važno je znati da naša distribucija Linuxa standardno dolazi s dvije starije inačice kernela i jednom najnovijom. Odabir inačice kernela je automatski takav, da će se u pravilu sustav uvijek pokrenuti sa zadnjom (najnovijom) inačicom kernela.*
*I nakon nadogradnje sustava, uvijek ćemo imati automatski pokrenutu najnoviju inačicu, ali dostupne i prve dvije starije. Starije dvije inačice se čuvaju, kako bismo sustav ipak mogli ručno pokrenuti u slučaju kada nešto eventualno pođe po zlu.*

**Izvori informacija: (562),(563),(564),(565),(566),(K-12),** `man make`,`man modinfo`,`man grub2-set-default`.

#### 11.1.1.4.1. Dodatne radnje s kernelom

U određenim situacijama možda ćete imati potrebu brisanja nepotrebnih (starijih) inačica kernela na sustavu.
U normalnim okolnostima broj instaliranih i neiskorištenih kernela ne utječe na performanse računala ili poslužitelja.
Prema zadanim postavkama ***Red Hat*** (*Fedora*/*CentOS*/*Rocky*/*Oracle*) će zadržati posljednjih pet instaliranih kernela na vašem sustavu. Takvo ponašanje definirano je opcijom (retkom) `installonly_limit=5` unutar datoteke `/etc/yum.conf`.

Međutim, uklanjanje starih neiskorištenih kernela oslobodit će malo prostora na disku. Pogotovo, ako je vaš poslužitelj konfiguriran sa zasebnom particijom `/boot` i naiđete li na problem s malo preostalog prostora na disku ili toj particiji, a imate potrebu nadograditi kernel odnosno instalirati najnoviji. U tom slučaju uklanjanje starijih kernela jedino je rješenje.

Pogledajmo jednu takvu situaciju s premalom `/boot` particijom (skratili smo ispis):

```
df -h
Filesystem       Size  Used Avail Use% Mounted on
devtmpfs         4.0M     0  4.0M   0% /dev
tmpfs            890M     0  890M   0% /dev/shm
/dev/sda3        7.1G  1.5G  5.6G  21% /
/dev/sda2        376M  290M   87M  77% /boot
```

← Vidimo da je `/boot` particija ukupne veličine samo 376 MB. To često nije dovoljno. Posebno kod nadogradnje kernela.

## Za Red Hat 5, 6 i 7

Za to imamo dvije mogućnosti.
Međutim prvo instalirajmo jedan dodatni softverski paket (ako ga već nemamo):

```
yum install yum-utils
```

Unutar ovog softverskog paketa nalazi se jedan od uslužnih programa imena `package-cleanup` za čišćenje softverskih paketa koje možete koristiti i za brisanje starog kernela, kao što je prikazano u nastavku.
Pri tome koristimo opciju za brojanje, koja se koristi za određivanje maksimalnog broja kernela koje želite ostaviti na sustavu.

Primjerice, ako želimo da sustav u bilo kojem trenutku može imati samo tri kernela, to možemo napraviti sa:

```
package-cleanup --oldkernels --count=3
```

Nakon pokretanja gore navedene naredbe uklonit će se svi stari kerneli, a zadržati trenutni pokrenuti kernel te dva starija (novija) kernela kao sigurnosne kopije.

Druga i dodatna opcija (za ubuduće) je unutar datoteke `/etc/yum.conf` promijeniti vrijednost opcije to jest promijeniti redak: `installonly_limit`.
Primjerice moguće je promijeniti sustav na zadržavanje samo zadnja tri kernela, s promjenom ovog retka u:

```
installonly_limit=3
```

## Za Red Hat 8x i noviji

Pošto se od ***Red Hat 8.x*** koristi DNF, a ***yum*** je poveznica na ***DNF*** i ovdje su uvedene neke izmjene, pa ne možemo više koristiti gore navedene metode.
Primjerice, ako želimo zadržati samo trenutnu (zadnju) i jednu stariju inačicu kernela, to možemo napraviti sa:

```
yum remove --oldinstallonly
```

Dodatna opcija (za ubuduće) je unutar datoteke `/etc/dnf/dnf.conf` promijeniti vrijednost opcije to jest promijeniti redak: `installonly_limit`.
Primjerice moguće je promijeniti na zadržavanje samo zadnja tri kernela, s promjenom ovog retka u:

```
installonly_limit=3
```



Za više detalja o **DNF**-u, pogledajte poglavlje:
**7.2.3. Rad s DNF-om.**

**Izvori informacija: (1387),(1388),** `man package-cleanup`**,**`man yum`**,**`man dnf.conf`**,**`man df`**.**

### 11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava

U trenutku svakog pokretanja računala, ***BIOS*** *ugrađen* na matičnu ploču računala pokreće inicijalizaciju sustava tijekom koje se s diska koji je označen za pokretanje (*boot*) sustava, a s lokacije prvog sektora na disku koji se zove *Master Boot Record* (***MBR***) učitava prva komponenta prije učitavanja operativnog sustava. Ovaj komponenta to jest program odnosno zapis na prvom sektoru diska se naziva ***boot loader***. ***Red Hat***/*CentOS* Linux koristi takozvani ***GRUB boot loader***. Njegov primarni zadatak jest pronalazak sljedeće poziciju na disku (najčešće particije) s koje potom može učitati ostatak operativnog sustava.
Dalje u tekstu govorit ćemo o **BIOS** načinu pokretanja računala (ne **UEFI**) i *Master Boot Record* (**MBR**) shemi diska.



Za više informacija o shemi diska, ***MBR*** ili ***GPT*** zapisima, pročitajte poglavlje:
**13.4. Logička shema diska**
Detalji o procesu pokretanja računala vidljivi su u poglavlju (i njegovim podpoglavljima):
**12.1. Što se događa kod pokretanja računala (boot process).**
**12.1.2. Boot proces detaljnije.**

***GRUB boot loader*** u sljedećim koracima učitava ostatak operativnog sustava, počevši s učitavanjem kernela, kako je definirano u ***GRUB*** odnosno ***GRUB2*** konfiguracijskim datotekama. *Red Hat/CentOS* 7.x/8.x koriste ***GRUB2*** inačicu programa.
U konfiguracijskoj datoteci ***GRUB2*** *boot loadera*, moguće je za svaki definirani kernel dodati i razne opcije i argumente s kojim će se konkretno odabrani kernel učitati odnosno pokrenuti. Dakle važno je razumjeti kako je moguće definirati više kernela s kojim se sustav može pokrenuti; naravno od kojih se može učitati sâmo jedan od njih:

- Automatski – svaki puta primjerice kernel s indeksom broj jedan (**1**) to jest konkretno `index=1`.
- Ili ručnim odabirom – sve kako smo i vidjeli u prethodnom poglavlju.

Trenutna konfiguracija ***GRUB2*** se nalazi u datoteci: `/etc/grub2.cfg` koja je simbolička poveznica (*link*) na izvorišnu datoteku: `/boot/grub2/grub.cfg`. U ovoj datoteci se nalaze sekcije za svaki od kernela koji možemo odabrati za pokretanje sustava, s identifikatorima za svaki od njih, koji počinju od nule (**0**). Kako ne bi morali ručno mijenjati ovu tekstualnu datoteku mi ćemo koristiti program odnosno alat koji će to napraviti za nas. Ovaj program se zove `grubby`.



Konfiguracijska datoteka ***GRUB2*** jest: `/etc/grub2.cfg`, a generira se tijekom instalacije sustava ili pozivanjem programa `grub2-mkconfig` (ili `grub-mkconfig`), a `grubby` je također automatski ažurira svaki puta kada se instalira novi kernel. Kada se (re)generira ručno koristeći `grub2-mkconfig`, datoteka se generira u skladu s datotekama predložaka koji se nalaze u direktoriju `/etc/grub.d/` te s prilagođenim postavkama i opcijama u datoteci `/etc/default/grub`. Ručnim uređivanja `grub2.cfg` njen sadržaj će biti izgubljen svaki puta kada se koristi `grub2-mkconfig` za ponovno generiranje datoteke. Stoga morate paziti da se sve ručne promjene opcija i parametara zapisuju u: `/etc/default/grub`.

Pogledajmo koje sve sekcije vezane za kernele te njihove pripadajuće opcije i argumente imamo trenutno konfigurirane na našem sustavu. Kako bismo to vidjeli, pozovimo naredbu `grubby` na sljedeći način:

```
grubby --info=ALL

index=0
kernel=/boot/vmlinuz-4.20.3
args="ro crashkernel=auto rhgb quiet LANG=en_US.UTF-8"
root=UUID=300463cf-ec75-4079-a029-19e2aecd430a
initrd=/boot/initramfs-4.20.3.img
title=CentOS Linux (4.20.3) 7 (Core)

index=1
kernel=/boot/vmlinuz-3.10.0-957.5.1.el7.x86_64
args="ro crashkernel=auto rhgb quiet LANG=en_US.UTF-8"
root=UUID=300463cf-ec75-4079-a029-19e2aecd430a
initrd=/boot/initramfs-3.10.0-957.5.1.el7.x86_64.img
title=CentOS Linux (3.10.0-957.5.1.el7.x86_64) 7 (Core)
```

Kako bismo shvatili što je sve ovdje vidljivo, pogledat ćemo sekciju pod indeksom **0**, to jest: `index=0`:

- `kernel=/boot/vmlinuz-4.20.3` - označava gdje se nalazi sâm kernel (on je kako je vidljivo, datoteka u `/boot/` direktoriju) koji će se ovdje učitati i to, ako se upravo ôn odabere za pokretanje sustava.
- `args="ro crashkernel=auto rhgb quiet LANG=en_US.UTF-8"` - ovo su argumenti koji se šalju ovom kernelu, a počinju s ključnom riječi `args=` unutar navodnika `""` s time da se ključne riječi odvajaju razmakom. Pri tome:
  - `ro` - *root* datotečni sustav će se inicijalno montirati samo s pravima čitanja, ali ne i zapisivanja (*read only*).
  - `crashkernel=auto` - znači kako želimo imati i takozvani *crash kernel* koji će se automatski učitati prilikom podizanja sustava.



Za više detalja pogledajte poglavlje:
**16. Kernel Dump/Crashdump/core dump.**

- `rhgb` - za prikaz će se koristiti *RedHat Graphical Boot* sustav za prikaz poruka tijekom pokretanja sustava.
- `quiet` - poruke o greškama i druge poruke će se minimalno zapisivati/ispisivati (logirati).
- `LANG=en_US.UTF-8` - znači kako se postavke jezika stavljaju na *Američki Engleski*, s upotrebom ***UTF-8*** skupa znakovnika.


Pogledajte poglavlje:
**10.3.2. Kodiranje, dekodiranje i kodne stranice.**

- `root=UUID=300463cf-ec75-4079-a029-19e2aecd430a` - označava *UUID* identifikator particije na kojoj se nalazi kernel, ali i ostatak operativnog sustava. S naredbom `lsblk -f` možemo vidjeti koja je to particija.
- `initrd=/boot/initramfs-4.20.3.img` - označava inicijalni* RAM disk (*initial RAM disk*) sustav.


Pogledajte poglavlje:
**10.7.1.2.1. initrd i/ili initramfs i inicijalizacija sustava**.

- `title` – označava tekst koji će se prikazati za odabir ovog kernela u samom trenutku pokretanja sustava.
- `selinux=0` – ukoliko u potpunosti želimo onemogućiti ***SELinux*** sustav, postavimo ovu opciju.


Za detalje oko *SELinux* sustava pogledajte poglavlje:
**28.2. SELinux sustav.**

Pogledajmo još pokoju korisnu opciju kernela, a koju možemo koristiti u ***GRUB2:***

- `isolcpus=0-1` – označava izolirane CPU jezgre, ako ih baš želimo izolirati od *task/proces schedulera*.


Za detalje oko *izolacije CPU jezgri od task/proces schedulera* pogledajte poglavlje:
**10.7.2.4. Izolacija jezgri procesora.**

- `intel_iommu=on` i/ili (za ***SR-IOV*** mrežne kartice `iommu=pt`) označavaju upotrebu ***IOMMU*** komponente.


Za detalje oko ***IOMMU*** pogledajte poglavlje:
**10.7.1.4. Ulazno-izlazni memorijski kontroler (IOMMU).**

- `hugepages=2M hugepages=512` – za uključivanje i rezerviranje **HugePages** od 1GB (512 x 2MB=1GB).


Za detalje oko ***Huge Pages*** pogledajte poglavlje:
**12.3.4.2. Huge Pages.**

Lista kernel opcija i argumenata ovisi o točnoj inačici kernela, ali generalno se može naći na izvorima informacija: **(606),(607),(608).**


*Naziv **GRUB** je nastao od engleske kratice: **GRand Unified Bootloader**.*

**Odabir standardnog kernela (i pripadajućih opcija i parametara) tijekom svakog pokretanja sustava**
Primjerice postavimo da se tijekom pokretanja sustava, sustav standardno pokreće s kernelom definiranim u indeksu **1**:
```
grubby --set-default-index=1
```

**Micanje/brisanje pojedinih argumenata iz svih sekcija kernela**
U slučaju kada želimo obrisati sâmo argument `rhgb` iz svih sekcija kernela (odnosno svîh kernela), to možemo napraviti sa:
```
grubby --remove-args "rhgb" --update-kernel=ALL
```
Unutar zagrada smo mogli navesti i više argumenata, odvojenih razmakom, primjerice `rhgb quiet`.

**Dodavanje pojedinih argumenata u sve sekcija kernela**
U slučaju kada želimo dodati sâmo argument `rhgb` u sve sekcije kernela (svîh kernela), to možemo napraviti sa:
```
grubby --update-kernel=ALL --args="rhgb"
```
Unutar zagrada smo mogli navesti i više argumenata, odvojenih razmakom, primjerice `"rhgb quiet"`.

**Dodavanje ili brisanje pojedinih argumenata samo u željenu sekciju kernela (željeni kernel)**
Postoje i slučajevi kada želimo dodati primjerice sâmo argument `rhgb` u točno određenu sekciju kernela odnosno za točno određeni kernel. U slučaju kada želimo već navedeni argument dodati za trenutno pokrenuti kernel to možemo napraviti tako da prvo provjerimo koji je trenutno aktivni kernel:

```
uname -r

3.10.0-957.5.1.el7.x86_64
```

Vidimo da je to kernel: `3.10.0-957.5.1.el7.x86_64`.
Ovaj kernel nalazi se kao datoteka imena *vmlinuz-INAČICA*, konkretno u slijedećoj putanji:
`/boot/vmlinuz-3.10.0-957.5.1.el7.x86_64`

To znači da moramo napraviti sljedeće:

```
grubby --args "rhgb" --update-kernel /boot/vmlinuz-3.10.0-957.5.1.el7.x86_64
```

Ista stvar bi bila i za brisanje, s time da za brisanje koristimo opciju `--remove-args` kako smo već vidjeli.
I na kraju sve što smo mijenjali možemo vidjeti s naredbom:

```
grubby --info=ALL
```

Nakon ponovnog pokretanja (*restarta*) računala (ili u bilo kojem trenutku) možemo vidjeti s kojim se opcijama i argumentima pokrenuo naš trenutno aktivi kernel (što se zapisuje u datoteku `/proc/cmdline`):

```
cat /proc/cmdline

BOOT_IMAGE=/boot/vmlinuz-3.10.0-957.5.1.el7.x86_64 root=300463cf-ec75-4079-a029-19e2aecd430a ro crashkernel=auto rhgb quiet
```


Sve promjene koje smo ovdje radili biti će aktivirane tek nakon *restarta* računala/poslužitelja na kojem smo radili.

Program `grubby` dolazi u istoimenom ***rpm*** softverskom paketu imena `grubby`.
Dakle instalira se na sljedeći način (ako ga slučajno nemate na sustavu):

```
yum -y install grubby
```

***Grubby*** podržava mnoge *boot loadere*, poput:

- ***Grub*** - za 32 i 64 bitnu arhitekturu x86 procesora.
- ***Lilo*** - za 32 i 64 bitnu arhitekturu x86 procesora.
- ***Elilo*** - za ***ia64*** arhitekturu procesora.
- ***Yaboot*** - za ***powerpc*** arhitekturu procesora.
- ***Zipl*** - za ***s390*** arhitekturu procesora.


Za detalje o arhitekturama procesora, pogledajte poglavlje:
**10.7.1.1. Malo detaljnije o registrima, instrukcijama i arhitekturi procesora.**

## Ručni rad

Ako imamo takve potrebe i određene opcije ili parametre želimo ručno dodati za sve kernele na sustavu, za to je dovoljno ručno urediti datoteku: `/etc/default/grub` i to njen redak koji počinje sa: `GRUB_CMDLINE_LINUX=". . ."`
Primjerice, ako želimo dodati opciju `intel_iommu=on`, na kraj ovog retka, unutar navodnika ju dodajemo okvirno ovako:
`GRUB_CMDLINE_LINUX="...... intel_iommu=on"`

Dakle unutar navodnika, na kraj retka možemo dodati željene opcije, a potom je potrebno generirati *GRUB2* datoteke i zapise:

```
grub2-mkconfig
```

Zatim je potreban samo restart sustava.


Za detalje o radu **GRUB**(2) sustava te detalje o procesu pokretanja računala, pogledajte:
**11.1.1.5.1. Rad s GRUB-om prilikom pokretanja sustava (*rescue*).**
**12.1.2. Boot proces detaljnije.**

**Izvor informacija: (606),(607),(608),(609),(1056),(1117),(K-12),(K-14), `man grubby`, `man uname`, `man grub2-mkconfig`, `man 7 bootup`, `man 7 bootparam`, `man 7 kernel-command-line`.**

#### 11.1.1.5.1. Rad s GRUB-om prilikom pokretanja sustava (*rescue*)

Prilikom poretanja računala, nakon inicijalizacije i pokretanja BIOS-a, kada se *GRUB2* pokrene, pojavljuje se izbornik u kojem možemo birati između operativnih sustava (OS) pronađenih na sustavu odnosno definiranih u *GRUB2*.
*GRUB2* standardno automatski učitava zadani operativni sustav nakon definiranog vremenskog ograničenja.
Međutim, prije isteka tog vremena, a prije pokretanja operativnog sustava, moguće je odabrati dva načina rada:

- Pritiskom na slovo **e** – moguće je uređivati parametre kernela prije pokretanja operatinvnog sustava.
- Pritiskom na slovo **c** – ulazimo u posebni naredbeni redak GRUB-a. Naredbeni redak *GRUB2* nije obična Linux ljuska, poput **bash** ljuske, i prihvaća samo određene naredbe specifične za *GRUB2*



Za više informacija o shemi diska, ***MBR*** ili ***GPT*** zapisima, pročitajte poglavlje:
**13.4. Logička shema diska**

Detalji o procesu pokretanja računala vidljivi su u poglavlju (i njegovim podpoglavljima):
**12.1. Što se događa kod pokretanja računala (boot process).**
**12.1.2. Boot proces detaljnije.**

**Naredbeni redak GRUB2 odnosno takozvani GRUB2 terminal**
U ranoj fazi pokretanja sustava, u trenutku kada se pojavi *GRUB2* izbornik (meni), kada je izbor na odabranom kernelu, možete stisnuti tipku **c**. S time ćete ući u *GRUB2* naredbeni redak, koji izgleda ovako:

```
grub>
```

Ulaskom u *GRUB2* terminal dostupne su nam *GRUB2* naredbe koje su možda slične Linux naredbama, ali rade drugačije. Pogledajmo samo neke od njih:

- `set` - ispisuje postavke *GRUB2* poput: particije s koje se *GRUB2* pokreće (`boot=`), particiju s koje se učitava korijenski datotečni sustav operativnog sustava (`root=`), kernel opcija (`kernelopts=`) i slično.
- `ls` - ispisuje sve particije koje su dostupne *GRUB2* sustavu.
- `insmod` - učitava željeni *GRUB2* modul. Pogledajte poglavlje: **12.1.2. Boot proces detaljnije.**.
- `boot` - pokreće *boot loader* sa zadane particije (ili neke druge; primjerice s ***Windows*** particije).

Kod *GRUB2* naredbi `set`, `ls` i sličnih, oznake diskova su drugačije od Linuxa. Pogledajmo primjerice naše računalo s jednim diskom i dvije particije koje bi pod Linuxom bile vidljive kao (`/dev/sda1` i `/dev/sda2`):

```
grub>ls
   (hd0) (hd0,msdos2) (hd0,msdos1)
```

←U ispisu je `hd0` oznaka prvog diska (format ispisa je `hd`X, pri čemu je X redni broj diska). Na tom disku vidimo i dvije particije, koje su vidljive kao: `(hd0,msdos1)` i `(hd0,msdos2)`, a one se ispisuju od zadnje prema prvoj.

Vezano za *boot* proces, ako smo pokrenuli naredbu `set`, vidljivo je sljedeće (uz naš opis kao komentar):

```
grub>set
boot=hd0,msdos1     ← particija s koje se učitava GRUB2
root=hd0,msdos1     ← particija s koje se učitava korijenski datotečni sustav
```

Sve ove opcije su zapravo zapisane u datoteci `/etc/grub2/grub.cfg`, odnosno vidljive su kao unosi:

```
set boot='hd0,msdos1'
set root='hd0,msdos1'
```

Za informacije o ovoj particiji i njenom datotečnom sustavu, možemo pokrenuti naredbu, uz skraćenu oznaku particije (`1`):

```
grub>ls (hd0,1)/
/bin /boot /cdrom /dev /etc /home /lib /lib64 /opt /proc /root /run …
```

← Vidimo da *GRUB2* može dohvatiti direktorije u datotečnom sustavu, a posebice direktorij `/boot`, što je vrlo važno za nastavak.

Ako bi bilo potrebno primjerice promijeniti *boot* particiju u drugu po redu, to bi mogli napraviti sa:

```
set root=(hd0,msdos2)
```

Zatim trebamo učitati *GRUB2* modul (`normal`) koji je odgovoran za učitavanje kernela i initramfs-a operativnog sustava,a koji nudi dodatne opcije prilikom okretanja sustava:

```
grub>insmod normal
```

Sada možemo pokrenuti normalno pokretanje sustava (od *GRUB2* menija).

```
grub>normal
```

**Nakon ove naredbe sustav će se pokrenuti s prve particije diska!**

**Ručni rad - detaljan proces pokretanja Linuxa iz GRUB2 terminala**
U određenim slučajevima nepravilnosti u radu *GRUB2* ili nekih drugih problema, ako imamo potrebu za ručnim pokretanjem sustava to možemo učiniti sa sljedećim koracima. Pogledajmo primjer za pokretanje sustava s prve particije diska (`hd0,1`):

```
grub> set root=(hd0,1)
grub> linux /boot/vmlinuz-6.8.0-1.el9.elrepo.x86_64 root=/dev/sda1
grub> initrd /boot/initramfs-6.8.0-1.el9.elrepo.x86_64.img
grub> boot
```

← U prvom korak usmo definirali particiju diska, u drugom kernel (`linux`), a u trećem *initramfs* sustav (`initrd`), a na kraju, sa zadnjom naredbom `boot` inicijaliziramo pokretanje sustava.

**Za privremene promjene GRUB2**
Za promjenu parametara kernela samo tijekom pokretanja sustava, prilikom procesa pokretanja sustava (kao na slici **87.A.**) potrebno je pritisnuti slovo `e`. S kursorskim tipkama treba otiči u redak koji počinje s „`linux`”, ali na kraj retka u koji možemo upisati dodatne opcije kernela, koje su nam eventualno potrebne (neke od njih smo naveli u prethodnom poglavlju).

**Za trajne promjene možemo učiniti sljedeće**
Ukoliko želimo da promjene *GRUB2* budu trajne, imamo nekiliko opcija:
1. Ručno uređivati *GRUB2* konfiguracijske datoteke
2. Za promjene po pitanju odabira kernela i slično možemo koristiti `grubby` alat.



Za više informacija o **grubby** alatu, pogledajte prethodno poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

**Ručno uređivanje GRUB2 datoteka**
Ručno uređivanja *GRUB2* datoteka radimo, kada primjerice želimo proširiti *GRUB2* izbornik s novim unosima ili mijenjati već postojeće unose. Naime *GRUB2* omogućuje osnovnu prilagodbu izbornika za pokretanje sustava kako bi imali kontrolu nad onim što se pojavljuje na ekranu prilikom pokretanja sustava ili mienjali ponašanje sustava.

Primjerice kada se sustav ažurira ili se dogode promjene u konfiguraciji koja zahtijeva promjene u *GRUB2 boot loaderu* (npr. dodavanje novog operativnog sustava ili promjene kernela), *GRUB2* će automatski pretražiti `/etc/grub.d/` direktorij kako bi generirao novu konfiguracijsku datoteku `grub.cfg`. To osigurava da konfiguracija *GRUB2 boot loadera* bude dinamična i ažurirana s promjenama u sustavu, čime se omogućuje fleksibilnost u upravljanju pokretanjem sustava.

Moguće je i ručno pokrenuti naredbu za ažuriranje *GRUB2* konfiguracije, s naredbom `grub2-mkconfig` ovako:

```
grub2-mkconfig
```



Za više informacija i primjera o ovakvim promjena, pogledajte prethodno poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

## Reinstalacija GRUB2 sustava i *boot* komponenti u MBR

Proces (re)instalacije *GRUB2 boot loadera* odrađuje se pomoću naredbe `grub2-install` .
S ovom naredbom se u pozadini prvo pokreće proces kopiranja sadržaja diskovnih slika (prvo `boot.img` pa `diskboot.img` te `core.img` ) na početne dijelove diska (**MBR**, međuprostor i **VBR**).
Kada je sve potrebno (re)kreirati *GRUB2 boot loader* i zapise na disku?

- Ako ste mijenjali GRUB2 konfiguraciju ili ste nadogradili kernel.
- Ako ste mijenjali postavke pokretanja sustava (promjene particije s koje se treba pokretati sustav i slično).
- Ako je sustav oštećen i potrebno je rekreirati GRUB2 boot loader zapise.



Za više informacija u procesu pokretanja računala i *boot loaderu*, pogledajte poglavlje:
**12.1. Što se događa kod pokretanja računala (*boot process*).**

Ako ručno želite rekreirati *boot* zapise na particijama diska (pr. za disk `/dev/sda`) to možete napraviti sa:

```
grub2-install --target=/dev/sda
```

Ili jednostavno:

```
grub2-install /dev/sda
```

U slučaju kada se nešto dogodilo i s *GRUB2* instalacijskim programima, možete ih i reinstalirati:

```
yum reinstall grub2-tools
```

# Resetiranje lozinke za root korisnika (administratora)

Za resetiranje root lozinke prvo je potrebno restartati računalo.
Zatim u *GRUB2* izborniku treba odabrati "*RESCUE*" kernel, poput primjerice zadnje opcije na slici dolje:

**Slika 87.A. GRUB izbornik tijekom pokretanja sustava.**




U ovom trenutku ne smijete stisnuti *ENTER*, već tipku `e`.

Sada se treba pojaviti novi *GRUB2* izbornik za ovaj (*RESCUE*) kernel, s mogućnočću uređivanja parametara (ovog) kernela s kojim ćemo pokrenuti novi (privremeni) sustav.

**Slika 87.B. Dodatni GRUB izbornik tijekom pokretanja sustava.**




S kursorskim tipkama treba otiči u redak koji počinje s ključnom riječi `linux`, ali na kraj retka.
Na kraj tog retka potrebno je upisati opciju: `rd.break` (konkretno nakon ključne riječi `quiet`).

I zatim je potrebno stisnuti **`CTRL x`** da se sustav pokrene s "*RESCUE*" kernelom i to s našom opcijom `rd.break`.

Nakon nekoliko trenutaka dobit ćemo novi naredbeni redak poput:

```
switch_root:/#
```

Od ovog trenutka naša sistemska particija (pr. `/dev/sda1`), koja je u konkretnom slučaju formatirana s XFS datotečnim sustavom je montirana samo za čitanje (*readonly*) i to u direktorij `/sysroot`, što možemo provjeriti s naredbom:

```
mount | grep sysroot
/dev/sda1 on /sysroot type XFS (ro,relatime,attr2,inode ….)
```

← Oznaka „`ro`" (engl. *read only*) definira samo pravio čitanja, ali ne i zapisivanja!

S obzirom da promjene lozinke znače promjene u datoteci na sistemskoj particiji (u našem slučaju `/dev/sda1`) i to u datoteci `/etc/shadow`, kako bi to napravili naš datotečni sustav mora biti montiran s pravima čitanja i zapisivanja (`rw`).

Prvo remontirajmo našu particiju s pravima čitanja i zapisivanja (`rw`):

```
mount -o remount,rw /sysroot
```

Zatim provjerimo novo stanje odnosno je li naša particija montirana s parvima zapisivanja i čitanja (kao `rw`):

```
mount | grep sysroot

/dev/sda1 on /sysroot type XFS (rw,relatime,attr2,inode ….)
```

← Ovdje, kod montiranja, vidimo `rw`, što znači da imamo prava zapisivanja.

Sada je potrebno prebaciti `/sysroot` da postane novi korijenski (`/`) direktorij, kako bi promjene od tog trenutka bile rađene na njemu. To ćemo napraviti ovako:

```
chroot /sysroot
```



Za više informacija o *chroot*, pogledajte poglavlje:
**27.3. Chroot i chroot jail.**

I tek sada možemo promijeniti lozinku za *root* koisnika:

```
passwd root
```

← Slijedi standardna promjena lozinke

Međutim, ako koristimo *SELinux* (što je zadano ponašanje), potrebno je učiniti sljedeće.
Naime u ovom trenutku, prilikom pokretanja naredbe `passwd` za ponovno postavljanje lozinke, datoteka u kojoj se lozinka zapravo mijenja (`/etc/shadow`) je modificirana s netočnim *SELinux* kontekstom.

Naredba `touch /.autorelabel` kreirat će skrivenu datoteku pod nazivom `.autorelabel` u korijenskom direktoriju. Prilikom sljedećeg pokretanja, *SELinux* podsustav će otkriti ovu datoteku, a zatim će ponovno označiti sve datoteke na tom sustavu s ispravnim *SELinux kontekstima*.

Stoga pokrenimo *SELinux* relabeliranje datoteka:

```
touch /.autorelabel
```

A sada možemo izaći i iz *chroot* okruženja:

```
exit
```

I zatim možemo izaći iz cijelog sustava:

```
exit
```

Nakon ovoga *SELinux* će pokrenuto proces relabeliranja datoteka i promjena root lozinke je uspješno napravljena.



Za više informacija o *SELinuxu*, pogledajte poglavlje:
**28.2. SELinux sustav.**

**Sažetak: kako i na koji način se kreiraju GRUB2 konfiguracijske datoteke**
*GRUB2* konfiguracija se inicijalno generira automatski tijekom instalacije distribucije Linux, pomoću skripte za instalaciju poput ***Anaconda*** (za *Red Hat*) ili **Ubiquity** (za *Debian/Ubuntu*). Ona pokreće `grub2-mkconfig` koji čita informacije iz:

- `/etc/default/grub` – koji sadrži globalne postavke, opcije i parametre za *GRUB2*.
- `/etc/grub.d/` - direktorij koji sadrži individualne skripte za definiranje unosa u *GRUB2* meniju. Ove skripte obično generiraju sami softverski paketi ili administratori za prilagođavanje sustava. Važnije skripte u ovom direktoriju su:
  - `00_header` – sadrži važne globalne postavke i varijable.
  - `10_linux` – služi za dodavanje (novih) unosa za dostupne Linux kernele, kao i parametara tih kernela. Ona pretražuje direktorij `/boot/` u potraz za novim kernelima (datotekama imena `vmlinuz-XY`) i generira unose za pojedine kernele. Definira i dodatne parametre kernela za *GRUB* (`root` – particiju, `initrd` – *initframfs*) i sl.
  - `30_os-prober` – služi za prepoznavanju eventualno dodatno instaliranih operativnih sustava pa omogućuje pokretanje drugih operativnih sustava; primjerice *Windows* OS, druge ditribucije Linuxa i slično (tv. ***Dual Boot***).
  - `40_custom` – ona služi za korisničke prilagodbe (ako ih želimo raditi). Korisna je za specifične konfiguracije i automatizaciju zadataka prilikom pokretanja sustava.

Skripte u direktoriju `/etc/grub.d/` se izvršavaju prema rednim brojevima, počevši od najmanjeg prema višem.

Nakon instalacije, konfiguracija se automatski ažurira kada se primjerice instalira novi kernel.
U slučaju potrebe, moguće je i ručno pokrenuti `grub2-mkconfig` za regeneriranje nove *GRUB2* konfiguracije.

**Izvori informacija: (1615),(1616),(1617),(1618),(1619),(1620),(1621),(1622),(1624),(1625),(1626), `man grub2-mkimage`,`man grub2-install`,`man grub2-mkconfig`,`man chroot`,`man mount`, `man grubby`.**

### 11.1.1.6. Tainted kernel

Pojam ***tainted kernel*** (engl. *Tainted* = inficiran, zaražen, onečišćen) označava kernel koji je označen na način koji indicira da se s njim nešto dogodilo (ne nužno loše). Uzroka tome može biti više, a najčešći je korištenje vlasničkog/zatvorenog kernel modula (upravljačkog programa) ili onoga koji nije kompatibilan sa GPL licencom. Dakle obično je to rezultat učitavanja vlasničkih upravljačkih programa (zatvorenog kôda) za primjerice grafičke kartice tvrtki *NVIDIA* ili *AMD* ili nekih drugih poput specijaliziranih kartica ili hardvera za koje proizvođač ne nudi upravljački program otvorenog već zatvorenog programskog kôda (tzv. vlasnički odnosno *zatvoreni* softver). Naime kada se takvo stanje dogodi, od sustava ćemo dobiti brojčanu poruku u vîdu bît maske te oznaku poruke. Pogledajmo vrijednosti i značenja svih ovih 18 bitova bît maske:

| Oznaka bîta: | Zastavica | Broj | Opis |
|---|---|---|---|
| 0 | **G/P** | 1 | Korištenje vlasničkog/zatvorenog kernel modula (upravljačkog programa) ili onog koji nije kompatibilan sa ***GPL*** licencom. Ovu poruku možemo izbjeći (donekle) upotrebom kernela i pripadajućih upravljačkih programa koje dobivamo službenim kanalima (od sâme distribucije Linuxa). |
| 1 | **F** | 2 | Kernel modul je na silu učitan. Ovu poruku možemo izbjeći izbjegavajući „*force*“ opciju kod učitavanja/odčitavanja kernel modula. |
| 2 | **S** | 4 | *SMP* kernel se izvršava na procesoru koji nije *SMP* (više jezgren). Ovu poruku možemo izbjeći ne korištenjem *SMP* kernela na sustavima koji ne podržavaju SMP. |
| 3 | **R** | 8 | Kernel modul je na silu odčitan (a ne pripada trenutnom kernelu). Ovu poruku možemo izbjeći izbjegavajući „*force*“ opciju kod učitavanja/odčitavanja kernel modula. |
| 4 | **M** | 16 | Procesor je izvijestio o „*Machine Check Exception* (***MCE***)“ koja može indicirati grešku unutar sâmog procesora (hardverska greška u procesoru), sabirnice ili BIOSa ili temperature. Ovu poruku možemo izbjeći ispravno konfiguriranim hardverom koji je pravilno korišten. |
| 5 | **B** | 32 | Referenciranje na problematičnu stranicu memorije (*Memory page*) ili neka druga zastavica vezana za stranice memorije. Ovu poruku možemo izbjeći ispravno konfiguriranim/odabranim hardverom koji je pravilno korišten (napajanje, temperatura, hlađenje). Ovdje se to prvenstveno odnosi da memorijski sustav (CPU i RAM). |
| 6 | **U** | 64 | Nenormalan zahtjev od strane aplikacije na korisničkoj razini.<br>Provjeriti problematičnu aplikaciju. |
| 7 | **D** | 128 | Kernel je nedavno ubijen zbog neke greške (OOPS - *Kernel Panic*). Provjeriti što je uzrokovalo ovakvu grešku (možda **OOM** stanje ili neka veća greška u kernelu).<br>Po potrebi preći na noviju inačicu kernela. |
| 8 | **A** | 256 | Korisnik (*user*) je poništio (nadvladao) ACPI tablicu. |
| 9 | **W** | 512 | Kernel je izvijestio o grešci. Potrebna je vrlo detaljna analiza sustava. |
| 10 | **C** | 1024 | Upotreba upravljačkih programa koji jesu dio izvornog programskog kôda kernela, ali su iz kategorije nestabilnih odnosno netestiranih (engl. *Staging driver*). |
| 11 | **I** | 2048 | Rješavanje problema s pogreškama u firmvare-u primijenjeno od korisnika (*user level*). Ovu poruku možemo često izbjeći pronalaskom problematičnog firmware-a i njegovom nadogradnjom. |
| 12 | **O** | 4096 | Upotreba upravljačkog programa koji je izvan razvojnog stabla Linux kernela odnosno koji nije uključen u izvorni programski kôd Linux kernela. |
| 13 | **E** | 8192 | Učitan je kriptografski nepotpisani kernel modul (upravljački program).<br>Ovu poruku možemo izbjeći upotrebom certificiranih upravljačkih programa. |
| 14 | **L** | 16384 | Proces ili nît procesa nije prestala s izvršavanjem te nije oslobodila CPU (soft lockup). |
| 15 | **K** | 32768 | Kernel je na živo zakrpan (*Live patch*). |
| 16 | **X** | 65536 | Druge promijene definirane i u upotrebi zbog specifičnosti distribucije Linuxa. Ovo nije nužno ili uopće problem, a može se pripaziti da se koristi hardver kompatibilan s vašom distribucijom Linuxa odnosno ona direktno podržan u kernelu (ili pripadajućim kernel modulima). |
| 17 | **T** | 131072 | Kernel je kompiliran sa tzv. „*struct randomization*“ dodatkom. |

Pogledajmo jednu ovakvu poruku sustava vidljivu s naredbom `dmesg` ili vidljivoj u sistemskoj log datoteci: `/var/log/messages`:

```
Kernel is Tainted
nvidia: module license 'NVIDIA' taints kernel
Raw taint value as int/string: 01/' G/P '
```

Vidimo kako je kernel modul `nvidia` izazvao *tainted* kernel poruku broj jedan (`1`) vidljivu kao `G/P` jer je ovaj kernel modul (upravljački program) pisan kao vlasnički to jest sa zatvorenim (skrivenim) programskim kôdom.
U datoteku `/proc/sys/kernel/tainted` se upisuje *tainted* vrijednost. Sa sljedećim nîzom naredbi ispisat ćemo bît masku trenutnih vrijednosti postavljenih u ovoj datoteci odnosno *tainted* vrijednost našeg kernela.

```
for i in $(seq 18); do echo $(($i-1)) $(($(cat /proc/sys/kernel/tainted)>>($i-
1)&1));done
```

**Izvor informacija: (866),(867),`man bash`,`man seq`.**

### 11.1.1.7. Mehanizam prepoznavanja hardvera i učitavanje kernel modula

**Slijedi napredna cjelina.** U trenutku pokretanja operativnog sustava, kernel je tâj koji prepoznaje instalirani hardver: matičnu ploču, disk kontroler i diskove, grafičku karticu i sav ostali hardver. On se u tom trenutku brine i o tome da za prepoznati hardver učitaju i njegovi upravljački programi (*kernel moduli*), koji su ili već ugrađeni u sâm kernel ili da se dodatno učitaju upravljački programi iz zasebnih datoteka tj. *kernel modula* s diska. Ako se radi o *PCI* uređaju, svaki uređaj ima ugrađen jedinstveni **PCI ID** identifikator, bez obzira na koji utor na sabirnici ga spojili. Na osnovu tog **PCI ID** identifikatora kernel pronalazi odgovarajući kernel modul (upravljački program) za taj uređaj odnosno hardver, a slično je i s **USB** uređajima. Za taj mehanizam je konkretno zadužen servis koji se zove `udevd`, a koji je zadužen i za prepoznavanje hardverskih komponenti koje smo naknadno mogli spojiti na računalo, poput: USB diska, mrežne kartice i slično. Na novijim sustavima, ***udevd*** je zamijenjen sa `systemd-udevd` servisom. Za primjer pogledajmo našu *Intelovu* mrežnu karticu (**82545EM**), čiji upravljački program (*kernel modul*) je `e1000`.

```
lspci | grep Ethernet
00:08.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet Controller
```

Vidimo da je ona (`82545EM`) spojena na *PCI/PCIexpress* sabirnicu na poziciju `00:08.0`. Stoga saznajmo nešto više o njoj:

```
lspci -s 00:08.0 -n
00:08.0 0200: 8086:100f (rev 02)
```

Sada smo dobili informaciju da uređaj na *PCI/PCIexpress* sabirnici, u utoru: `00:08.0`, ima *PCI* oznake: `8086:100f`.
Međutim kako bismo shvatili što one znače, pogledajmo kako je sâm kernel prepoznao našu mrežnu karticu, spojenu na *PCI* sabirnicu, u utor: `00:08.0`. Kako bismo to saznali tražimo sadržaj `modalias` datoteke u `/sys/devices/pci/` strukturi:

```
cat /sys/devices/pci0000\:00/0000\:00\:08.0/modalias
  pci:v00008086d0000100Fsv000015ADsd00000750bc02sc00i00
```

Dobili smo sljedeće: pod ***vendor*** (`v`) vidimo oznaku `8086` [predstavlja proizvođača *Intel*], dok pod ***device*** (`d`) `100F`, što predstavlja sâm uređaj. Dok se `bc` (*Base class*) i `sc` (*Subsystem*) koriste da bi se prepoznala takozvana klasa (*Class*) *PCI* uređaja. U našem slučaju je to (spajaju se `bc` i `sc`) vrijednost: `0200`, koju smo i vidjeli s: `lspci -s 00:08.0 -n`. Ovu klasu `udevd` koristi kao identifikator klase uređaja (iz **PCI ID** baze), koju u malo čitkijem formatu, možete vidjeti i online na adresi: https://pci-ids.ucw.cz/read/PD/. U konkretnom slučaju imamo sljedeće očitanje: klasa `bc` `02` je „*Network controller*“, dok je `sc` `00` „*Ethernet controller*“. Dakle `udevd` tijekom inicijalizacije ove *Intelove* mrežne kartice (**82545EM**) zna da upravljački program (*kernel modul*) pripada u kategoriju: *Network controller + Ethernet controller*, tj, pripada klasi: `0200`.

**Kako su povezani ovi podaci s prepoznavanjem pravog kernel modula koji sustav treba učitati za ovu mrežnu karticu?**
Svaki kernel modul u sebi već ima popisane sve PCI identifikatore za uređaje za koje je i namijenjen.
Pogledajmo skraćenu listu *PCI* identifikatora kao *aliasa*, za naš upravljački program (*kernel modul*) imena: `e1000`.

```
modinfo e1000 | grep alias
alias:          pci:v00008086d00001010sv*sd*bc*sc*i*
alias:          pci:v00008086d0000100Fsv*sd*bc*sc*i*   ← naša mrežna kartica
alias:          pci:v00008086d0000100Esv*sd*bc*sc*i*
```

U trenutku nakon kompiliranja kernela i njegovih kernel modula, `kmod` sustav (ili noviji `systemd-modules-load`) kreira listu takozvanih *module aliasa* tako da indeksira i povezuje **PCI** identifikatore iz sâmih kernel modula i druge potrebne podatke, izvlačeći sve dodatno i iz datoteka: `modules.dep`, `modules.symbols` i `modules.builtin`.
Sve navedene datoteke, za trenutno aktivni kernel, možete vidjeti na sljedećoj lokaciji odnosno u direktoriju:

```
ls -al /lib/modules/`uname -r`/modules*
```

U navedenom direktoriju, na osnovu navedenih datoteka i svih dostupnih kernel modula za konkretni kernel, kreira se i posebna datoteka imena `modules.alias`. Navedena *kernel module alias* datoteka (`modules.alias`) sadrži kao *aliase* upravo *PCI* identifikatore uređaja, koji su navedeni za svaki kernel modul zasebno.
Pogledajmo datoteku `modules.alias` i to unos samo za uređaj s *PCI* identifikatorima **`8086:100f`**:

```
grep v00008086d0000100F /lib/modules/`uname -r`/modules.alias

alias pci:v00008086d0000100Fsv*sd*bc*sc*i* e1000
```

Ovdje vidimo da je *alias* za **PCI** identifikator `v00008086d0000100` kernel modul imena `e1000`.
Ponovno rekreiranje datoteke `modules.alias` se događa i u slučaju kada sami kreiramo novi kernel modul, odnosno nakon što pozovemo `depmod` naredbu, na sljedeći način:

```
depmod -a
```

U svakom slučaju kada `udevd` servis prepozna uređaj (hardver) tako da je prvo pročitao njegove *PCI* identifikatore, on prvo gleda u datoteku `modules.alias` te u njoj pronalazi pronađeni *PCI* identifikator uređaja (mrežne kartice u našem slučaju). On zatim u istom retku pronalazi i koji kernel modul treba učitati (u našem slučaju je to `e1000`) te ga potom i učitava.



Vezano za **PCI ID** bazu, pogledajte poglavlje:
**10.2.3. Nadogradnja baze podataka PCI identifikatora (PCI baze).**
Za detalje oko uređaja i njihovog prepoznavanja, pogledajte poglavlja:
**11.1.1.3. Parametri koje možemo poslati kernel modulima te automatsko učitavanje modula.**
**11.1.2.1. Uređaji (*devices*) detaljnije.**

**Izvori informacija: (1025),(1026),(1027),(1028),** **`man kmod`**, **`man depmod`**, **`man modinfo`**, **`man lspci`**, **`man 7 udev`**, **`man systemd-udevd`**, **`man systemd-modules-load`**.

### 11.1.2. Uređaji (*devices*) ukratko

U svakom *Unix* kao i *Linux* operativnom sustavu, gotovo sve stvari odrađuju se korištenjem neke (posebne) datoteke.
Čak se i pristup i rad s raznim uređajima odnosno hardverom, odrađuje preko posebnih datoteka, koje ga indirektno predstavljaju.
Kako smo već naučili, osnovne vrste datoteka u Linuxu, mogu biti:

- Takozvane *karakter* datoteke (engl. ***Character)*** koje se koriste za komunikaciju s uređajima s kojima se komunicira slanjem ili primanjem nîza karaktera/znakova odnosno podataka u nîzu.
- Takozvane *blok* datoteke (engl. ***Block)*** koje se koriste za komunikaciju s uređajima s kojima se komunicira slanjem ili primanjem većih blokova podataka.
- Posebne vrste datoteka, su redom:
  - ***Pipe*** i ***named Pipe*** - datoteke su koje se koriste za komunikaciju između procesa (programa).
  - ***Socket*** - datoteke su koje se koriste za komunikaciju između procesa (programa) i mrežnu komunikaciju.
  - ***Symbolic link*** - datoteke koje su simboličke poveznice na postojeće datoteke unutar datotečnog sustava.
- Dodatno, posebne datoteke koje predstavljaju uređaje i obično se nalaze u direktoriju `/dev/`

Pogledajmo i logičku shemu ovog sustava posebnih datoteka na primjeru upotrebe programa to jest naredbe za formatiranje prve particije drugog diska (`sdb`) koju predstavlja posebna datoteka `/dev/sdb1`. Naredba koju izvršavamo kada želimo formatirati prvu particiju drugog (*SATA*) diska je `mkfs.ext4`, a koju u našem slučaju pozivamo na sljedeći način:

```
mkfs.ext4 /dev/sdb1
```

S gore navedenom naredbom *formatirat* ćemo prvu particiju na drugom SATA (može biti i SCSI ili SAS vrsta diska) tvrdom disku, a koji je u Linuxu vidljiv kao posebna datoteka: `/dev/sdb`. Odnosno njegova prva particija je vidljiva kao posebna datoteka `/dev/sdb1`. Pogledajmo i kako logički izgleda pozivanje gore navedene naredbe (slika **88**):

**Slika 88. Logička veza tijekom pokretanja gore navedene naredbe.**




Dolje u tekstu govorit ćemo o ovoj posebnoj vrsti datoteka u `/dev/` direktoriju koje predstavljaju hardverske uređaje, te ćemo se ukratko upoznali i s *linux* uređajima (engl. *devices*). Pod linux uređajima (*linux devices*) podrazumijevamo posebne datoteke koje preko upravljačkih programa direktno pristupaju određenom hardveru odnosno fizičkim uređajima našeg računala.
Uređaje u Linuxu predstavljaju posebne datoteke koje se nalaze unutar `/dev/` direktorija.
Ove datoteke prema načinu rada, odnosno načinu pristupa samom hardveru, mogu raditi na dva načina:

- Kao ***karakter*** datoteke, koje od i prema uređajima šalju ili primaju podatke u nîzu znakova odnosno *karaktera*.
- Kao ***blok*** datoteke, koje od i prema uređajima šalju ili primaju podatke u blokovima podataka. Stoga se i uređaji koji ih koriste zovu *blok* uređaji.

## Upoznajmo se i s nekoliko osnovnih kategorija ovih posebnih uređaja pod Linuxom

**Diskovi**
Sljedeće posebne datoteke predstavljaju disk uređaje (oni su prema načinu rada *blok* uređaji):

- `/dev/hd*` - ATA (IDE) tvrdi disk. Tako `hda` označava primarni *master* disk, dok `hda1` označava prvu particiju na primarnom master tvrdom disku. Zatim `hdb` označava primarni *slave* disk, dok `hdb1` označava prvu particiju na primarnom *slave* disku i tako dalje.
- `/dev/sd*` - SATA, SCSI ili SAS tvrdi disk. Tako je `sda` prvi disk a `sdb` je drugi disk te `sdc` treći i tako dalje. Brojevi iza slova označavaju broj particije diska: isto kao kod *ATA* diskova.
- `/dev/nvme*n1` - NVMe *SSD* disk. Tako je `nvme0n1` prvi *NVMe* disk, `nvme1n1` drugi i tako dalje.

**Terminali**
Sljedeće posebne datoteke predstavljaju terminale (oni su prema načinu pristupa *karakter* uređaji):



Za opis načina rada *terminala* i njihove namjene, pogledajte poglavlje:
**3.1. Unix/Linux ljuska *(Shell)* i terminali.**

- `/dev/tty*` - terminali su koji se koriste za ispisivanje (engl. *Text output*) na monitor/ekran. Pri tome je `tty0` prva konzola odnosno fizički terminal, vidljiv kada smo direktno prijavljeni/logirani na računalo ispred nas.
- `/dev/tty` - predstavlja trenutni kontrolni terminal, trenutno pokrenutog procesa. Ako neki program čita ili piše na terminal, koristi se upravo ovaj terminal uređaj.
- `/dev/pts/*` - je pseudo odnosno virtualni terminal, a to su emulirani terminali koji se ponašaju poput stvarnih terminala, ali se koriste za ispis ili preusmjeravanje prema nekom procesu poput nove ljuske (*shella*) ili udaljenog pristupa (pr. *SSH*). Primjer upotrebe bi bili terminali poput: `/dev/pts/0` ili `/dev/pts/1` te drugi.

**Serijska sučelja (portovi)**
Sljedeće posebne datoteke predstavljaju serijske portove (oni su isto *karakter* uređaji):

- `/dev/ttyS*` - ovo su serijski portovi (*COM* portovi pod *Windows* OS-om). Pri tome je `ttyS1` prvi serijski port, a `ttyS2` je drugi serijski port itd. Brzine rada su obično 9600, 14400, 28800, 33600, … bps. Koristili su se za spajanje modema. Danas se uglavnom koriste za spajanje na "*konzole*" mrežnih uređaja (preklopnika, usmjerivača i sl.).

**Paralelna sučelja (portovi)**
Sljedeće posebne datoteke predstavljaju paralelne portove (oni su isto *karakter* uređaji):

- `/dev/lp*` - ovo su paralelni portovi. `lp0` je prvi paralelni port, `lp1` je drugi paralelni port.
  Koristili su se obično za spajanje pisača koji su imali paralelno sučelje za spajanje s računalom.

**Floppy disk** (meki disk)
Sljedeće posebne datoteke predstavljaju meki disk (*floppy disk)* koji je prema načinu rada *blok* uređaj:

- `/dev/fd*` - meki (*floppy*) diskovi. Pri tome je `fd0` prvi *floppy* disk, a `fd1` drugi itd.

**Audio uređaji**
Sljedeće posebne datoteke predstavljaju audio uređaje:

- `/dev/mixer` - dio je *OSS* (*Open Sound System*) upravljačkog programa za zvučnu karticu.
- `/dev/dsp` – predstavlja *Digital Signal Processor*. On predstavlja vezu između programa i zvučne kartice koja generira (*stvara*) zvuk.

**Posebni memorijski uređaji** (oni su *karakter* uređaji):

- `/dev/null` - uređaj u koji se mogu slati podaci kao u "crnu rupu", a iz nje se više ništa ne može vratiti. Korisno ako želimo pokrenuti neki program, ali nas ne zanima njegov izlaz, pa ga možemo preusmjeriti (napraviti *redirekciju*) u `/dev/null` uređaj i tako obrisati sve što izlazi iz njega.
- `/dev/zero` - ovo je generator nula. Za razliku od `/dev/null` on ne briše sve podatke koji ulaze u njega već on kreira odnosno generira nule (0).
- `/dev/random` - ovo je nedeterministički generator "slučajnih" (engl. *Random*) brojeva. Prilično je spor jer koristi entropiju (nepredvidivost) hardverskih komponenti, poput nepravilnosti u oscilacijama generatora takta. U slučaju kada dovoljna razina entropije nije dostupna on mora čekati kako bi se ona pojavila ili očekuje dodatno slučajno pomicanje miša ili slično, kako bi entropija bila što veća. Postoje i hardverski uređaji koji mogu generirati što nepredvidljiviji nîz generiranih podataka. Koristi se za kriptiranje podataka i slične namjene.
- `/dev/urandom` - on također radi slično kao `/dev/random`, ali kada se dosegne donji prag entropije, on ne čeka već počinje koristiti takozvani *pseudo random number generating* formulu kako bi simulirao što "slučajniji" i što nepredvidljiviji nîz podataka. Ovaj *uređaj* radi višestruko brže od `/dev/random`, ali je donekle je predvidiv pa se ne preporuča njegova upotreba za ekstremno sigurnosno zahtjevne primjene. Ipak postoji mogućnost uporabe posebnog hardverskog generatora slučajnih brojeva, koji je vidljiv kao posebni uređaj: `/dev/hwrng` (ako ga imate) te servisa `rngd` (dolazi u softverskom paketu: `rng-tools`). Ovaj servis može preusmjeriti hardverski generator slučajnih brojeva (zapravo `RDRAND` ili `RDSEED` funkcionalnost) u datoteku: `/dev/urandom` te s time drastično ubrzati kreiranje slučajnih brojeva i povećati entropiju (nepredvidljivost). Trenutnu entropiju možete vidjeti sa:
  ```
  cat /proc/sys/kernel/random/entropy_avail
  ```
  Sve vrijednosti veće od nekoliko tisuća (2000+) ukazuju na upotrebu hardverskog izvora entropije.

**Slijedi nekoliko primjera**
Sjetimo se kako je posebna datoteka: `/dev/zero` zapravo generator nula (`0`) a koju ćemo iskoristiti za generiranje nula unutar jedne datoteke imena: `sve.nule.txt`. Osim toga, želimo da konačna veličina datoteke bude 1024 *bajta*.
Za rad s ovom posebnom datotekom koristiti ćemo program `dd` (engl. *disk dump*) koji se koristi za konvertiranje i kopiranje datoteka, a pošto je u *Unix/Linux* svijetu sve što vidimo i s čime radimo, neka datoteka, to ćemo i iskoristiti.
Ali prvo pogledajmo koje sve prekidače naredba `dd` ima. Lista nekoliko njenih osnovnih prekidača je sljedeća:

- `if=` - označava ulaznu datoteku u ovaj program, iz koje se čitaju podaci (engl. *input file*).
- `of=` - označava izlaznu datoteku iz ovog programa, u koju se snimaju podaci (engl. *output file*).
- `bs=` - za koliko *bajta* omogućavamo čitanje/pisanje odjednom (ovo je logički gledano jedan blok podataka).
- `count=` - koliko ovakvih blokova podataka (iz `bs`) dozvoljavamo obraditi u nîzu. Ovo možemo nazvati i brojem ponavljanja vrijednosti `bs`.

Ograničit ćemo veličinu bloka (`bs=1024`) na 1024 bajta samo s jednim ponavljanjem (`count=1`) i to ovako:
```
dd if=/dev/zero of=/neki-direktorij/sve.nule.txt bs=1024 count=1

1+0 records in
1+0 records out
1024 bytes (1.0 kB) copied, 5.8944e-05 s, 170.4 MB/s
```
Vidimo kako smo kreirali datoteku željene veličine, punu nula (0), brzinom od 170 MB u sekundi(što ovisi o brzini računala).

Drugi primjer će biti kopiranje cijele diskete (/dev/fd0) u datoteku: /neki-direktorij/disketa.img

```
dd if=/dev/fd0 of=/neki-direktorij/disketa.img
```

U sljedećem primjeru ćemo koristiti posebnu datoteku koja generira slučajne brojeve vrlo brzo: /dev/urandom i pomoću nje generirati datoteku veličine 1MB, imena test.file i to u blokovima od 1024 bajta, po tisuću puta (1k).
Dakle konačna veličina datoteke će biti 1024 x 1000 = 1 024 000 dakle oko jedan milijun znakova odnosno 1 MB:

```
dd if=/dev/urandom of=/neki-direktorij/test.file bs=1024 count=1k

1024+0 records in
1024+0 records out
1048576 bytes (1.0 MB) copied, 0.130189 s, 20.1 MB/s
```



Važno je razumjeti da posebne datoteke u direktoriju /dev/ predstavljaju hardver u računalu.

**Izvori informacija: (207), (K-12), man mkfs, man dd, man 7 udev, man 4 urandom, man 4 zero.**

### 11.1.2.1. Uređaji (*devices*) detaljnije

***Slijedi napredno poglavlje!.*** U trenutku pokretanja operativnog sustava, kernel je tâj koji prepoznaje instalirani hardver: matičnu ploču, disk kontroler i diskove, grafičku karticu i sav ostali hardver. On se u tom trenutku brine i o tome da za prepoznati hardver učita i njegov upravljački program (*kernel modul*), koji je ili već ugrađen u sâm kernel ili da dodatno učita upravljački program iz zasebne datoteke tj. *kernel modul* s diska. Kako smo spomenuli u poglavlju: **10.2. (Re)konfiguracija sabirnice i uređaja na njoj,** svaki uređaj ima ugrađen **PCI ID** jedinstveni identifikator, bez obzira na koji utor na sabirnici ga spojili.

Taj identifikator možemo dobiti u ispisu unutar uglatih zagrada [X:Y] naredbe lspci -nn. Na osnovu tog **PCI ID** identifikatora kernel pronalazi odgovarajući kernel modul (upravljački program) za taj uređaj odnosno hardver. Slično je i s **USB** uređajima. Vezano za upravljačke program važno je znati kako linux kernel standardno dolazi s ugrađenim tisućama i tisućama upravljačkih programa za osnovni hardver, a dodatno se svi ostali upravljački programi nalaze kao posebne datoteke na datotečnom sustavu, u direktoriju: /lib/modules/***INAČICA-KERNELA***/kernel/drivers/ koje je moguće pokrenuti i naknadno. I na kraju, bilo da se radi o upravljačkom programu već ugrađenom u linux kernel ili onom učitanom iz posebne datoteke (kernel modula), u tom procesu kreira se i posebna struktura pôddirektorija /dev/ u kojoj se potom kreiraju i posebne datoteke koje predstavljaju te upravljačke programe i u konačnici konkretan hardver kojim upravlja upravljački program.

Naime u trenutku nakon pokretanja operativnog sustava Linux, poseban servis koji se zove udevd zadužen je za prepoznavanje dodatnih hardverskih komponenti sustava: USB disk, USB memorija, mrežna kartica i slično te kreiranja zasebnih unosa u posebnom direktoriju /dev/ koji se potom vežu za upravljački program (*driver*) odnosno kako se to pod Linuxom naziva *kernel modul*, za svaki pojedini uređaj. Poveznica između svake posebne datoteke u direktoriju /dev/ i njenog upravljačkog programa je takozvani ***Major*** broj (o njemu kasnije). Servis udevd koji smo spomenuli, a koji ih i povezuje, je prema svojem načinu rada takozvani „*Device Manager*" odnosno menadžer za hardver tj. za sve uređaje. On je zadužen za kreiranje posebnih datoteka u direktoriju /dev/ od kojih svaka predstavlja neki uređaj. U trenutku kada smo isključili neki od tih uređaja (primjerice USB disk), udevd briše i posebnu datoteku u /dev/ direktoriju, koja se referencira na taj disk preko upravljačkog programa.



Za detalje mehanizma prepoznavanja hardvera te rada ***udevd*** servisa, pogledajte poglavlje:
**11.1.1.7. Mehanizam prepoznavanja hardvera i učitavanje kernel modula.**



Na distribucijama Linuxa koje koriste systemd (***RedHat/Centos*** 7+) za ***udev*** funkcionalnost se koristi servis imena: **systemd-udevd**. Njegov status možete vidjeti s naredbom systemctl status systemd-udevd

Pošto se udevd program izvršava u korisničkom memorijskom prostoru (izvan *kernelovog* prostora), moguće su određene intervencije od strane korisnika, pošto udevd ovisi o posebnom virtualnom datotečnom sustavu sysfs.
Nadalje, moguće je mijenjati imena tih uređaja, korištenjem takozvanih ***udev rules*** konfiguracijskih datoteka koje se nalaze u direktoriju: /etc/udev/rules.d/. U priči o dodavanju novog ili isključivanju hardvera, zadatak je sâmog kernela da otkrije novi uređaj odnosno hardver. U svakom slučaju kernel za sâv hardver kreira i određene unose što su zapravo posebne datoteke koje su vidljive unutar sysfs datotečnog sustava, a koji se nalazi u direktoriju /sys/, a upravo koje i prati udevd servis.

U obje priče, bez obzira je li se neki hardver inicijalizirao u trenutku pokretanja sustava ili kasnije, prije trenutka učitavanja upravljačkog programa, kreira se posebna datoteka koja predstavlja taj upravljački program odnosno hardver koji stoji iza njega i to prema posebnim pravilima, o kojima ćemo sada govoriti.

Važno je znati i kako je moguće ručno kreirati posebnu datoteku unutar direktorija `/dev/` koja predstavlja određeni hardver, a koja se zove ***node***. Naravno da je naredba za to, upravo `mknod` (kao *Make NODE*). Svaki *device file* odnosno posebna datoteka unutar direktorija `/dev/` koja i predstavlja neki uređaj (hardver), ima dvije posebne oznake.
Za primjer pogledajmo datoteke koje predstavljaju prvi i drugi *SATA* disk odnosno diskove ***sda*** i ***sdb*** *(/dev/sda i /dev/sdb)*:

```
ls -al /dev/sda /dev/sdb

brw-rw---T 1 root disk 8,  0 Dec  2 18:13 /dev/sda
brw-rw---T 1 root disk 8, 16 Dec  2 18:13 /dev/sdb
```

Prvo slovo (`b`) u ovlastima datoteke govori nam kako se radi o blok uređajima. Napomena: druga moguća kategorija su *karakter* uređaji, koji bi imali prvo slovo (`c`). U petom stupcu za disk `/dev/sda` vidimo brojeve: `8,   0` a za disk *sdb* vidimo brojeve: `8,  16`. Analizirajmo ove brojeve:

1. Prvi broj (u ovom slučaju `8`) se naziva ***Major*** odnosno značajniji broj (u našem slučaju je to broj `8` za oba uređaja). Ovaj broj identificira o kojem upravljačkom programu se radi odnosno on identificira sâm upravljački program. U datoteku `/proc/devices` se upisuju svi upravljački programi sa pripadajućim ***Major*** brojevima koji su u upotrebi.
2. Drugi broj (u ovom slučaju `0`) se naziva ***Minor*** odnosno manje značajan (u našem slučaju disk *sda* ima broj `0` a disk *sdb* ima broj `16`). Ovaj broj identificira o kojem fizičkom ili logičkom uređaju (kojim upravlja upravljački program) se konkretno radi.

Radi se o tome, kako prvi (*Major*) broj identificira upravljački program koji se mora koristiti, dok drugi broj (*Minor*) identificira sâm uređaj za koji ga koristimo.

**Poveznica između posebne datoteke u `/dev/` direktoriju, koja predstavlja uređaj sa pripadajućim *Major* i *Minor* brojevima nalazi se u `/sys/dev/` direktoriju i to u jednom od dva poddirektorija:**

- `/block/` - ako se radi o *blok* uređaju.
- `/character/` - ako se radi o *karakter* uređaju.

U našem slučaju pogledajmo prvi disk `/dev/sda/` *i to* `8,0`. Koristit ćemo naredbu `readlink` zbog ljepšeg ispisa, ali mogli smo koristiti i naredbu `ls`. Pogledajmo sada naš uređaj s identifikatorima `8,0`:

```
readlink /sys/dev/block/8\:0/device/driver
../../../../../../bus/scsi/drivers/sd
```

Za blok uređaje ova informacija se nalazi i u direktoriju `/sys/block/`***`OZNAKA-DISKA`***`/device/driver`.
Stoga pokrenimo sljedeću naredbu:

```
readlink /sys/block/sda/device/driver
```

I dobit ćemo nešto poput: `../../../../../../bus/scsi/drivers/sd`
Dakle radi se o vršnom upravljačkom programu `sd`, koji je konkretno došao sa sâmim kernelom odnosno ugrađen je u njega.


Vezano za sabirnicu, pogledajte i poglavlje:
**10.2. (Re)konfiguracija sabirnice i uređaja na njoj**

Međutim u slučaju SATA/SCSI ili SAS diskova, ovo je malo složenije priča jer se radi o funkcioniranju i radu na dvije osnovne razine:

1. Vršna razina, kojom upravlja `sd` komponenta, a koja se oslanja na upravljački program za disk kontroler (dolje).
2. Sâm upravljački program za konkretan disk kontroler i u konačnici disk.

Saznali smo prvu razinu (`sd`), a sada nas zanima konkretan upravljački program za disk kontroler, na koji je spojen disk. Detaljne informacije možemo dobiti pomoću programa `udevadm`.

Pogledajmo i kako; mada ćemo ispis maksimalno skratiti samo na polja koja sadrže ključnu riječ „`DRIVERS`":

```
udevadm info -a -n /dev/sda |grep DRIVERS
```

I dobivamo:

```
    DRIVERS=="sd"
    DRIVERS=="ata_piix"
```

Ovdje smo došli do toga da se radi o upravljačkom programu imena: `ata_piix` što je upravljački program najniže razine za SATA disk kontroler (*Intel PIIX/ICH (S)ATA)* u našem slučaju.


Više informacija o diskovnom I/O podsustavu i njegovim dijelovima, pogledajte u poglavlju:
**14. Diskovni (I/O) podsustav.**

Pogledajmo i osnovnu skraćenu tablicu linux uređaja sa pripadajućim ***Major*** i ***Minor*** brojevima:

| Major broj | Tip (Blok/Karakter) | Vrsta uređaja | Minor broj | Uređaj | Opis |
|---|---|---|---|---|---|
| 1 | **Karakter** | Memorijski uređaji | 1 | `/dev/mem` | Predstavlja pristup memoriji. |
| | | | 3 | `/dev/null` | Predstavlja prazan uređaj, koji može primati podatke (u prazno). |
| | | | 5 | `/dev/zero` | Predstavlja generator nula. |
| | | | 8 | `/dev/random` | Predstavlja nedeterministički generator slučajnih brojeva. |
| | | | 9 | `/dev/urandom` | Predstavlja jednostavni (brzi) generator slučajnih brojeva. |
| 1 | **Blok** | RAM disk | 0 | `/dev/ram0` | Prvi RAM disk. |
| | | | 1 | `/dev/ram1` | Drugi RAM disk. |
| 3 | **Blok** | ATA (IDE) disk | 0 | `/dev/hda` | Master disk. |
| | | | 64 | `/dev/hdb` | Slave disk. |
| | | | 1 | `/dev/hd*1` | Prva particija bilo kojeg *hd* (ATA) diska. |
| 4 | **Karakter** | TTY uređaji (terminali/konzole) | 0 | `/dev/tty0` | Trenutna konzola/terminal za rad. |
| | | | 1 | `/dev/tty1` | Prva konzola/terminal. |
| | | | 2 | `/dev/tty2` | Druga konzola/terminal. |
| 5 | **Karakter** | Drugi TTY uređaji | 0 | `/dev/tty` | Trenutna konzola za rad. |
| | | | 1 | `/dev/console` | Sistemska konzola. |

| Major broj | Tip (Blok/Karakter) | Vrsta uređaja | Minor broj | Uređaj | Opis |
|---|---|---|---|---|---|
| 6 | **Karakter** | Paralelni portovi (sučelja) | 0 | `/dev/lp0` | Prvi paralelni port (sučelje). |
| | | | 1 | `/dev/lp1` | Drugi paralelni port (sučelje). |
| 7 | **Blok** | Loopback uređaji | 0 | `/dev/loop0` | Prvi *loop* uređaj. |
| | | | 1 | `/dev/loop1` | Drugi *loop* uređaj. |
| 8 | **Blok** | SCSI disk uređaji | 0 | `/dev/sda` | Prvi SCSI/SAS/SATA disk (cijeli disk). |
| | | | 1 | `/dev/sda1` | Prvi SCSI/SAS/SATA disk, prva particija |
| | | | 2 | `/dev/sda2` | Prvi SCSI/SAS/SATA disk, druga particija |
| | | | 16 | `/dev/sdb` | Drugi SCSI/SAS/SATA disk (cijeli disk). |
| | | | 32 | `/dev/sdc` | Treći SCSI/SAS/SATA disk (cijeli disk). |
| 9 | **Karakter** | SCSI tračni uređaji | 0 | `/dev/st0` | Prvi SCSI/SAS/SATA tračni uređaj, mode 0. |
| | | | 1 | `/dev/st1` | Drugi SCSI/SAS/SATA tračni uređaj, mode 0. |
| | | | 32 | `/dev/st0l` | Prvi SCSI/SAS/SATA tračni uređaj, mode 1. |
| | | | 33 | `/dev/st1l` | Drugi SCSI/SAS/SATA tračni uređaj, mode 1. |
| 9 | **Blok** | Metadiskovi (RAID) | 0 | `/dev/md0` | Prva metadisk grupa. |
| | | | 1 | `/dev/md1` | Druga metadisk grupa. |
| 10 | **Karakter** | Razni uređaji | 130 | `/dev/watchdog` | Watchdog brojač (timer). |
| | | | 183 | `/dev/hwrng` | Hardverski generator slučajnih brojeva. |
| | | | 200 | `/dev/net/tun` | **TUN** ili **TAP** virtualni mrežni uređaji. |
| | | | 229 | `/dev/fuse` | **FUSE** virtualni datotečni sustav. |
| | | | 232 | `/dev/kvm` | Linux KVM virtualizacija (hardware virtualization extensions) . |

| Major broj | Tip (Blok/Karakter) | Vrsta uređaja | Minor broj | Uređaj | Opis |
|---|---|---|---|---|---|
| 10 | Karakter | Razni uređaji (nasatvak) | … | … | … |
| | | | 238 | /dev/vhost-net | Host kernel akcelerirana **virtIO** mrežna kartica. |
| 14 | **Karakter** | Open Sound System (OSS) | 0 | /dev/mixer | Audio mikser. |
| | | | 2 | /dev/midi | Prvi audio *MIDI* port. |
| | | | 3 | /dev/dsp | Digitalni audio. |
| 22 | **Blok** | Drugi ATA kanal | 0 | /dev/hdc | Master disk. |
| | | | 64 | /dev/hdd | Slave disk. |
| 29 | **Karakter** | Universal frame buffer | 0 | /dev/fb0 | Prvi *frame buffer*. |
| | | | 1 | /dev/fb1 | Drugi *frame buffer*. |
| 81 | **Karakter** | Video for Linux | 0 | /dev/video0 | Prvi *video capture* uređaj. |
| | | | 1 | /dev/video1 | Drugi *video capture* uređaj. |
| 180 | **Karakter** | USB uređaji | 0 | /dev/usb/lp0 | Prvi USB printer. |
| | | | 48 | /dev/usb/scanner0 | Prvi USB scanner. |
| 188 | **Karakter** | USB serijski uređaji | 0 | /dev/ttyUSB0 | Prvi USB serijski port. |
| | | | 1 | /dev/ttyUSB1 | Drugi USB serijski port. |
| 203 | **Karakter** | CPU | 0 | /dev/cpu/0/cpuid | Prva jezgra CPUa – info. |
| | | | 1 | /dev/cpu/1/cpuid | Druga jezgra CPUa – info. |
| … | … | … | … | … | … |
| 248 | **Karakter** | RTC | 0 | /dev/rtc0 | *Realtime Clock (sat).* |
| … | … | … | … | … | … |

Vidjeli smo kako je inicijalno kernel zadužen za prepoznavanje i kreiranje posebnih datoteka u `/dev/` direktoriju (mapi).
Ove posebne datoteke predstavljaju uređaje (hardver) ili posebne funkcionalnosti (u tom slučaju ih se naziva pseudo uređajima) i njih u normalnim okolnostima nije potrebno ručno kreirati jer se o svemu već brine kernel, a kasnije i navedeni `udevd` servis.
Međutim u posebnim slučajevima, ipak možemo imati potrebu da sami kreiramo ovakvu posebnu datoteku, koja predstavlja uređaj ili neku posebnu funkcionalnost sustava.

Zamislimo vrlo rijedak primjer u kojem imamo posebnu karticu sa serijskim portom (***UART***) za koji smo učitali kernel modul (upravljački program), ali se zbog nekog razloga nije kreirala posebna datoteka koja predstavlja tu serijsku karticu odnosno `/dev/ttyS1`. Tada ćemo sâmi morati kreirati ovu posebnu datoteku, a sustav će ju povezati s njenim upravljačkim programom.

Njene *Major* i *Minor* brojeve smo pronašli u gore navedenoj tablici (*Major*: `4` i *Minor*: `65` a vrsta je *karakter* odnosno oznaka `c`). Posebnu *device* datoteku `/dev/ttyS1` ćemo napraviti na sljedeći način:

```
mknod /dev/ttyS1 c 4 65
```

I nakon toga imamo dostupan naš novi serijski port odnosno serijsko sučelje.
Dodatno, moguće je kreirati i posebne „*device*" datoteke i izvan strukture `/dev/` direktorija. Primjerice kod izolacije programa u takozvanom „***Chroot jail***" okruženju, kod kojega izoliramo određene programe unutar vršne strukture direktorija, možemo imati i neke dodatne potrebe. Jedna od njih je izolacija posebnih datoteka koje nam trebaju za rad ovih izoliranih programa, poput posebnih datoteka koje predstavljaju generator nula (inače je to: `/dev/null`) ili generator slučajnih brojeva (inače je to: `/dev/random`).
U slučaju kada je naš vršni direktorij za izolaciju: */var/chroot/* unutar njega ćemo kreirati i direktorij `/dev`.
Prvo kreirajmo ove direktorije odnosno cijelo malo stablo direktorija (mapa):

```
mkdir -p /var/chroot/dev
```

Sada ćemo kreirati posebne datoteke, pomoću naredbe `mknod` na sljedeći način:

```
mknod /var/chroot/dev/null c 1 3
mknod /var/chroot/dev/random c 1 8
```

Pri tome */dev/null* predstavlja: *Major*:`1` i *Minor*:`3` dok */dev/random* predstavlja: *Major*:`1` i *Minor*:`8`

Pogledajmo što smo sada dobili:

```
ls -al /var/chroot/dev/

crw-r--r--. 1 root root 1,  3 Sep  2 19:21 null
crw-r--r--. 1 root root 1,  8 Sep  2 19:22 random
```

Ne zaboravite promijeniti ovlasti za ove datoteke, za vlasnika i grupu te prava: čitanja, pisanja i izvršavanja.

**Izvori informacija: (214),(733), `man readlink`, `man mknod`, `man udevadm`, `man systemd-udevd`, `man 7 udev`, `man 4 random`, `man 4 ttyS`, `man 4 null`.**

#### 11.1.2.1.1. *Udev* pravila (*udev rules*)

***Slijedi napredno poglavlje!***
U cjelini prije, govorili smo kako je sistemski servis udevd zadužen za prepoznavanje svôg hardvera, te kako je on, nakon što je hardver prepoznat, zadužen za instaliranje njegovog pripadajućeg upravljačkog programa, odnosno prema linux terminologiji kernel modula. Dodatno, postoje i takozvana *udev* pravila odnosno „*udev rules*" za (razni) hardver, koja opisuju taj hardver detaljnije, te pomažu *udevd* servisu kako bi se pronađeni hardver bolje inicijalizirao i/ili konfigurirao. Cijeli nîz osnovnih pravila koja koristi udevd, ali i posebnih pravila za određene uređaje, nalazi se unutar direktorija: `/lib/udev/rules.d/` odnosno za *RedHat/CentOS* 7+ je to vršni direktorij: `/usr/lib/udev/rules.d/`. Naime, kada je ***udev*** *servis* pronašao sâv hardver, obično kreira nekoliko permanentnih *udev rules* datoteka u kojima opisuje taj hardver, te ga konfigurira. Ovi parametri se tijekom sljedećeg pokretanja operativnog sustava čitaju iz navedenih datoteka. Jedna od osnovnih datoteka za posebne uređaje je ona koja opisuje i definira mrežne kartice: `/etc/udev/rules.d/70-persistent-net.rules`*. Za ***RedHat/CentOS* 7+** je to: `/usr/lib/udev/rules.d/60-net.rules`. Nazivi i oznake ovih datoteka su eventualno podložne promjenama!

Pogledajmo jednu ovakvu *udev* datoteku* na računalu s dvije mrežne kartice:

- `eth0` je ***Broadcom*** mrežna kartica, koja koristi kernel modul (upravljački program/*driver*) `tg3`
- `eth1` je ***Intel*** mrežna kartica, koja koristi kernel modul (upravljački program/*driver*) `e1000e`

Sada pogledajmo sadržaj ove datoteke:

```
cat /etc/udev/rules.d/70-persistent-net.rules
# PCI device 0x14e4:/sys/devices/pci0000:00/0000:00:1c.2/0000:07:00.0 (tg3)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:30:05:ae:8d:d7",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth0"

# PCI device 0x8086:/sys/devices/pci0000:00/0000:00:1c.3/0000:09:00.0 (e1000e)
SUBSYSTEM=="net",  ACTION=="add",  DRIVERS=="?*",  ATTR{address}=="00:08:60:00:79:20",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth1"
```

Na ispisu je vidljiva lokacija uređaja na *PCI* sabirnici (*PCI device*) te *MAC* adresa pripadajuće mrežne kartice, primjerice za *`eth0`* : `ATTR{address}=="00:08:60:00:79:20"` te sâmo ime pripadajuće mrežne kartice, pod `NAME=`.

Osim toga i ovdje je moguće mijenjati određene parametre. Za ***RedHat/CentOS 6.x*** se u konkretnu datoteku trajno (*permanentno*) zapisuju sve pronađene mrežne kartice. Kod ***RedHat/CentOS 7+*** to nije slučaj je se ovdje nazivi i inicijalne postavke mrežne kartice (kao i nekih drugih uređaja) definiraju u nîzu ***udev*** pravila[(952)],[(953)]. Prema ovim pravilima se mrežna kartica odnosno njen upravljački program (*kernel modul*) povezuje s imenom mrežne kartice (primjerice s imenima poput: *`eth0`*, *`eth1`* i slično).
Dakle poveznica je: **PCI adresa *PCI* kartice - MAC adresa →** `eth`**xy** (odnosno ime mrežne kartice kako ćemo ju vidjeti).
Međutim, dodatno je moguće za svaki pronađeni hardver i ručno mijenjati određene konfiguracijske parametre.

Navest ćemo sâmo par primjera. Prvo pogledajmo koje sve parametre određeni uređaji nude:

- Za disk (primjerice `/dev/sdb`) to možemo saznati sa:

```
udevadm info -a -n /dev/sdb
```

- Za mrežnu karticu (primjerice `eth0`), to možemo saznati pomoću naredbe:

```
udevadm info -a -p /sys/class/net/eth0
```

Opcije i parametri koji su dostupni i koje je moguće mijenjati, navedeni su za svaki uređaj, u ispisu naredbe: udevadm koje smo naveli gore, i to u polju s atributima (`ATTRS`).
Dodatno, pomoću ovih naredbi možemo doći i do mnogih detalja o našem hardveru.
Primjerice za našu mrežnu karticu `eth0` pogledajmo koji kernel modul (upravljački program) ona koristi:

```
udevadm info -a -p /sys/class/net/eth0 | egrep "DRIVERS|KERNEL"
```

Kao odgovor dobivamo sljedeće:

```
    KERNEL=="eth0"
    KERNELS=="0000:07:00.0"
    DRIVERS=="tg3"
    KERNELS=="0000:00:1c.2"
    DRIVERS=="pcieport"
    KERNELS=="pci0000:00"
    DRIVERS==""
```

Vidimo da mrežna kartica `eth0` koristi kernel modul *`tg3`*. To možemo potvrditi i s naredbom `lspci -v`, ako pronađemo traženu mrežnu karticu, pod **Ethernet controller**. Međutim s naredbom `lspci -v` ne vidimo radi li se o *`eth0`*, *`eth1`* ili nekoj drugoj mrežnoj kartici. Iste informacije smo mogli dobiti i s naredbom ethtool na sljedeći način:

```
ethtool -i eth0
```

Pripazite na nekoliko osnovnih pravila, ako sâmo kreiramo novu *udev rules* datoteku (**OPREZ - pazite što radite**):

- Datoteka mora biti u direktoriju `/etc/udev/rules.d/` dok se na ***RedHat/CentOS 7+*** primarno koristi direktorij: `/usr/lib/udev/rules.d/`, uz prethodno navedeni, zbog kompatibilnosti unazad.
- Datoteka mora imati ekstenziju `.rules`, a ime datoteke mora početi s rednim brojem, jer se sve datoteke unutar ovog direktorija učitavaju prema redoslijedu, od manjeg prema većem broju.

U konačnici ime datoteke mora biti oblika: *BROJ-NEKO-IME.rules*. Primjer bi bio: `98-optimizacija-diska.rules`. Ako želite da se vaša datoteka zadnja izvrši, dodijelite joj broj u imenu veći od **90**.
Prvo ćemo napraviti *udev* pravilo *(rule)* za našu treću mrežnu karticu, na ***RedHat/CentOS 7+*** sustavu.
Ovu mrežnu karticu je sustav prepoznao kao `enp0s8`, a naša želja je da ju sustav tijekom inicijalizacije odmah preimenuje u `lan2` jer ju kao takvu želimo konfigurirati i koristiti. To možemo postići kreiranjem novog globalnog *udev* pravila.



Za novi način označavanja mrežnih kartica pod ***RedHat/CentOS 7+*** pogledajte poglavlje:
**25.6.4. Nazivi mrežnih kartica u Linuxu.**

Mi ćemo se odlučiti na novu *udev* datoteku naziva: `71-persistent-net.rules` odnosno pravilo koje ćemo kreirati u vršnom direktoriju: `/etc/udev/rules.d/`, koji se čitaju na kraju, nakon što se učitaju sva pravila definirana unutar direktorija: `/usr/lib/udev/rules.d/` (za ***RedHat/CentOS 7.x+***).
Ovdje imamo nekoliko varijanti odnosno pitanja: po kojem parametru želimo da se naša nova mrežna kartica identificira. Naime znamo da je naša mrežna kartica prepoznata kao `enp0s8` te da se radi o *Intelovoj* mrežnoj kartici *82545EM* za čiji rad Linux koristi upravljački program (kernel modul) imena: `e1000`. Osim toga ova mrežna kartica je spojena na **PCI** utor s identifikatorom: `00:08.0`.

Kako smo sve to saznali?
Prvo smo provjerili koji upravljački program ona koristi:

```
ethtool -i enp0s8 | grep driver
driver: e1000
```

Potom smo pronašli na kojem ***PCI*** identifikatoru (***PCI DOMENA:SABIRNICA:UREĐAJ***) se kartica nalazi:

```
lspci |  grep -i Ethernet | grep 82545EM
00:08.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet Controller
```

Uočite poveznicu između imena kartice: `enp0s8` i PCI identifikatora `00:08`.
Osim toga, znamo i **MAC** adresu ove mrežne kartice; što smo saznali sa:

```
ip link show enp0s8 | grep ether
    link/ether 08:00:27:87:b1:9d brd ff:ff:ff:ff:ff:ff
```

Ostaje nam odluka kako želimo ovu karticu identificirati, a u konačnici takvu konfiguraciju moramo zapisati u novu datoteku: `/etc/udev/rules.d/71-persistent-net.rules`. Ako smo se za identifikaciju odlučili prema nazivu mrežne kartice: `enp0s8` i njenog upravljačkog programa `e1000`, tada dodajmo sljedeći red odnosno konfiguraciju u ovu datoteku:

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="e1000", KERNEL=="enp0s8", NAME="lan2"
```

*Pravilo kod dodavanja mrežne kartice na sustav* | *Njen upravljački program* | *Kako je prepoznata* | *Njeno novo ime*



*Nakon što se sustav ponovno restarta naša nova mrežna kartica koja je prije bila vidljiva kao* `enp0s8`*, sada će biti vidljiva kao:* `lan2` *te ćemo ju moći normalno konfigurirati i koristiti kao takvu.*

Druga metoda može biti identifikacija pomoću ***MAC*** adrese i upravljačkog programa, pa bi tada umjesto gore navedene konfiguracije trebali dodati nešto poput sljedeće konfiguracije (sve u jednom retku):

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="e1000",
ATTR{address}=="08:00:27:87:b1:9d", KERNEL=="enp*", NAME="lan2"
```

Ovdje vidimo da smo se vezali za ***MAC*** adresu (`08:00:27:87:b1:9d`) te upravljački program (`e1000`) kao i inicijalno ime kartice (`enp*`) te sve navedene identifikatore mrežne kartice u konačnici vezali za novo ime kartice `lan2`.
Nakon što smo sve ovo napravili, možemo kreirati konfiguracijsku datoteku za konfiguriranje IP parametara mreže.

Pošto se sada ova mrežna kartica zove `lan2`, njena konfiguracijska datoteka će biti imena (trebamo ju kreirati i konfigurirati): `/etc/sysconfig/network-scripts/ifcfg-lan2`.



Za detalje oko konfiguracije IP parametara mrežne kartice, pogledajte poglavlje:
**25.6.5. Konfiguracija mrežnih kartica.**

U sljedećem primjeru, napravit ćemo *udev rule* za naš *SSD* disk `/dev/sda`, u kojem ćemo promijeniti disk *scheduler* na **`Deadline`**.



Za detalje optimizacije rada *SSD* diska, pogledajte poglavlje:
**14.1.4 Optimizacija Generic Block Layera i I/O Schedulera**.
Dodatno pogledajte i primjere za optimizaciju mreže:
**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*)**
Konkretno, pogledajte pravilo: **80-net-txqueuelen.rules.**

Dodatno, prvo provjerimo jesmo li stvarno odabrali *SSD* disk. Ovdje tražimo onaj s vrijednosti `0` (dakle koji nije rotacijski/mehanički disk). Pokrenimo slijedeći nîz naredbi (u jednom retku):

```
for f in /sys/block/sd?/queue/rotational; do printf "$f is "; cat $f; done
```

I dobit ćemo slijedeći ispis:

```
/sys/block/sda/queue/rotational is 0
/sys/block/sdb/queue/rotational is 1
/sys/block/sdc/queue/rotational is 1
```

Vidimo da se ovdje definitivno radi o disku `/dev/sda`. Sada ćemo kreirati datoteku imena: `80-ssd-disk-optimizacija.rules` sljedećeg sadržaja:

```
# Postavimo DEADLINE scheduler jer je bolji za SSD diskove
ACTION=="add|change", KERNEL=="sda", ATTR{queue/rotational}=="0",
ATTR{queue/scheduler}="deadline"
```

Restartajmo sustav te provjerimo *disk schedulere* ponovno, sa sljedećom naredbom odnosno nîzom naredbi:

```
for f in /sys/block/sd?/queue/scheduler; do printf "$f is "; cat $f; done

/sys/block/sda/queue/scheduler is noop anticipatory [deadline] cfq
/sys/block/sdb/queue/scheduler is noop anticipatory deadline [cfq]
/sys/block/sdc/queue/scheduler is noop anticipatory deadline [cfq]
```

Vidimo kako je naš *SSD* disk (*/dev/sda*) sada odabrao `[deadline]` *disk scheduler*, a druga dva mehanička/rotacijska diska i dalje imaju odabran `[cfq]` *disk scheduler*.

**Izvor informacija:(952),(953), `man udev`, `man ethtool`, `man lspci`, `man udevadm`, `man ip`, `man 5 udev.conf`.**

---

### 11.1.2.1.2. Nadogradnja *firmwarea* uređaja

***Slijedi napredno poglavlje!***
Uz sâmu instalaciju upravljačkog programa (*kernel modula*) neki uređaji odnosno hardver za rad trebaju i takozvani *firmware* ili *mikrokôd*, kako bi uređaj bio uopće funkcionalan ili kako bi imao dodatne funkcionalnosti. Ovo je najčešće slučaj s raznim mrežnim karticama, ali i drugim hardverom, poput grafičkih kartica, disk i RAID kontrolera, USB uređaja i slično. Kod nekih uređaja, bez učitavanja *firmwarea* na njima rade sâmo osnovne funkcionalnosti, dok one napredne ne rade, a kod drugih ne rade niti osnovne funkcionalnosti uređaja. Stariji, a i neki posebni uređaji poput RAID kontrolera, imali su ili imaju takav dizajn, u kojem se njihov *firmware* nalazio već programiran odnosno upisan u EEPROM/FLASH memoriju na uređaju, što daje i osnovne, ali i sve napredne funkcionalnosti rada uređaja. U takvom dizajnu, mogli smo i sâmi napraviti nadogradnju njihovog *firmwarea*. U današnje vrijeme, sve više uređaja ili uopće nema takav *firmware* ili on nudi samo najosnovnije funkcije, pa je stoga potreban i dodatan *firmware*, sa svim ostalim funkcijama. Stoga je za ovu zadaću potrebna funkcija operativnog sustava, koja tijekom svakog pokretanja sustava, učitava i *firmware* za razne uređaje.



Za *CPU mikrokôd*, koji također možemo nazvati i *firmware*, pogledajte poglavlje:
**10.7.1.2 CPU Mikrokod**.

Važno je razumjeti, kako uz standardnu instalaciju linuxa, ne smiju doći sve inačice *firmwarea*, za razne uređaje. To je stoga što neki od proizvođača uređaja i njihovog pripadajućeg *firmwarea*, objavljuju svoj *firmware* pod licencama koje zabranjuju njihovu distribuciju unutar instalacije Linuxa. Za takav hardver je stoga potrebno:

1. Ili instalirati softverske pakete, unutar kojih se nalazi potreban *firmware*.
2. Ili ručno kopirati *firmware* uređaja, sa web stranica proizvođača.

Za osnovni slučaj u kojemu *firmware* dolazi s linuxom, on obično dolazi uz instalaciju kernela u paketu: *`kernel-INAČICA-KERNELA`*, koji raspakirava *firmware* datoteke u direktorij: `/lib/modules/`***`INAČICA-KERNELA`***`/kernel/drivers/firmware` te dodatno u pratećem paketu imena: `kernel-firmware-INAČICA-KERNELA`, koji sadrži veću količinu *firmwarea*, a koji će se raspakirati u direktorij: `/lib/firmware/`.
Dodatno je moguće da *firmware* dolazi i u paketu s alatima za nadogradnju *firmwarea* odnosno *mikrokôd*, poput paketa: `microcode_ctl`.

Za slučaj (**1**) u kojem morate sami instalirati softverski paket u kojem se nalazi *firmware*, neke od njih možete promaći na dostupnim repozitorijima vašeg Linuxa. Pogledajmo listu onih koji su dostupni na ovaj način, pomoću naredbe:

```
yum search firmware
```

**Koja je procedura učitavanja *firmwarea***

Naime za automatsko učitavanje *firmwarea* je također zadužen `udevd` servis/*daemon*, a njegova procedura rada je sljedeća:

1. Servis `udevd` učitava upravljački program (*driver*) za hardver.
2. Upravljački program (*driver*), ako mu je potreban *firmware*, to zatraži od sustava.
3. Kernel kreira događaj (*event*) prema `udev` servisu, u kojem se traži *firmware* za konkretan uređaj.
4. Servis `udevd` pokreće skriptu (obično: `/lib/udev/firmware.sh`) koja iz *firmware* datoteke izvlači podatke u formatu koji mu je prihvatljiv i zapisuje ih u posebnu privremenu datoteku.
5. Tu privremenu datoteku čita kernel i prosljeđuje ju upravljačkom programu konkretnog uređaja, za koji je namijenjen ovaj *firmware*.
6. Upravljački program uređaja sada učitava *firmware* za uređaj.

Kod ručnog učitavanja upravljačkog programa (*drivera/kernel modula*) *firmware* će također biti automatski učitan, ako postoji. Za potrebe *debuggiranja*, ako ručno učitavamo neki novi kernel modul, moguće je prije nego smo učitali kernel modul uključiti logiranje `udevd` servisa.

Tako ćemo moći vidjeti je li sve prošlo kako treba te je li i učitan potrebni *firmware*:

```
udevadm monitor --property
```

Sada u drugom *shellu/prozoru (terminialu)* možemo učitati kernel modul te ćemo u realnom vremenu, u prvom terminalu u kojem smo pokrenuli naredbu `udevadm monitor -property` i vidjeti što se događa.

Kada budemo pratili učitavanje *firmwarea*, sâma *firmware* datoteka će biti vidljiva u redu `FIRMWARE` poput ovoga dolje:

```
FIRMWARE=e100/d101m_ucode.bin
```

Podrazumijeva se kako se radi o vršnom direktoriju: `/lib/firmware/` unutar kojega se trebaju nalaziti ili *firmware* datoteke ili pôddirektoriji u kojima se nalaze navedene datoteke.

*Firmware* datoteke su obično sljedećih ekstenzija:

- `.dat`
- `.ucode`
- `.fw`
- `.bin`

Pokrenutu naredbu `udevadm monitor --property` prekidamo sa stisnutim tipkama: `CTRL` i `C`.

Pogledajmo listu sâmo nekih od mrežnih kartica (kernel modula) koje uglavnom traže učitavanje dodatnog *firmwarea*:

- ***Broadcom***: kernel moduli: `tg3`, `bnx2` i `bnx2x` (što uključuje desetke mrežnih kartica).
- ***QLogic***: deseci mrežnih kartica.
- ***Realtec***: deseci mrežnih kartica (8192E/SU, 8105/6e, 8168/d/e/f/g/h, …).
- ***Chelsio***: kernel moduli: `cxgb3`, `cxgb4` (nîz mrežnih kartica poput **T3** i **T4**).
- ***Myricom***: kernel modul: `myri10ge` (nîz 10Gbps mrežnih kartica).
- Većina bežičnih (engl. *wireless*) mrežnih kartica.

Vezano za mrežne kartice, inačicu *firmwarea* koji je u upotrebi za mrežnu karticu `eth0` možemo vidjeti sa (skratili smo ispis):

```
ethtool -i eth0

driver: bnx2x
version: 1.712.30-0
firmware-version: bc 7.10.72
```

Vidimo kako je trenutno aktivna inačica *firmwarea*: `bc 7.10.72`

U slučaju kada mi imamo noviji *firmware* od onoga koji je pohranjen na samoj kartici (ako je na njoj cijeli *firmware*) ili ako se radi o dodatnom *firmwareu*, tada će linux učitati najnoviju inačicu. To će tada izgledati otprilike ovako:

```
dmesg | grep firmware
bnx2x: QLogic 5771x/578xx 10/20-Gigabit Ethernet Driver bnx2x 1.712.30-0 (2014/02/10)
bnx2x:04:00.0: irq 82 for MSI/MSI-X
bnx2x 0000:04:00.0: firmware: requesting bnx2x/bnx2x-e1-7.10.72.0.fw
```

Dakle vidimo da je učitana nova inačica: `bnx2x-e1-7.10.72.0.fw` (***7.10.72***).

**Izvori informacija: (147),(148),(149), `man udevadm`,`man ethtool`,`man dmesg`.**

# 11.2. Sistemske biblioteke (*Libraries*)

**Što su sistemske i dijeljene biblioteke (engl. Libraries)?**
Svaki program, pisan u bilo kojem programskom jeziku (*C, C++, Java, Scala, Rust* …) sastoji se od većeg broja logičkih cjelina. Vrlo često svi programi koriste veliki broj logičkih cjelina koje su im svima zajedničke. Primjerice svi matematički kalkulatori moraju imati napisane funkcionalnosti za matematičke operacije zbrajanja, oduzimanja, množenja, dijeljenja i mnoge druge. Raznih kalkulatora postoji na stotine, kao i drugih programa koji imaju potrebu raditi navedene matematičke operacije. Nadalje, i mnogi drugi programi koriste na stotine ili tisuće funkcionalnosti koje su im zajedničke. Logično bi prema tome bilo napraviti neki program koji bi na najbolji i najbrži način mogao odrađivati te funkcionalnosti, a da ga mi iz svog "glavnog" programa sâmo možemo pozivati i koristiti.

**Dijeljene biblioteke**
Upravo tome i služe dijeljene biblioteke. Svaka od njih ima određeni broj funkcionalnosti koje možemo koristiti u našem (ili bilo kojem) programu. U praksi, postoje softverski paketi dijeljenih biblioteka sa stotinama i tisućama implementiranih funkcionalnosti; od matematičkih operacija, operacija za baratanje s mrežnim paketima, do svih drugih funkcionalnosti.
Prednost upotrebe dijeljenih biblioteka je u tome što naš program postaje manji i jednostavniji. Osim toga, s obzirom da veliki broj programa koristi iste dijeljene biblioteke, više nema potrebe za kopiranjem istih biblioteka unutar svakog programa i zauzimanja mjesta na disku, već više programa može koristiti istu (dijeljenu) biblioteku. Odatle i naziv dijeljene biblioteke.



Dijeljene biblioteke imaju ekstenziju `.so` (engl. *Shared Object*), koja označava „dijeljeni objekt".

## 11.2.1. Statičke biblioteke (*Static Libraries*)

Statičke biblioteke su potpuna suprotnost dijeljenim bibliotekama. One mogu biti biblioteke koje su vidljive i dijeljene s drugima, ali su u procesu kompiliranja programa statički povezane s njim (engl. *Linked*). Odnosno nakon procesa kompajliranja programa one postaju njegov sastavni dio, što znači kako se nalaze ugrađene unutar njega. To znači i kako je njihova funkcionalnost tada dostupna sâmo programu unutar kojeg se nalaze statički povezane.



Statičke biblioteke obično imaju ekstenziju `.a`

## 11.2.2. Prednosti i mane dijeljenih i statičkih biblioteka

**Statičke** biblioteke
Statičke odnosno *statički povezane* biblioteke se nalaze unutar sâmog programa pa stoga, ako imamo deset programa od kojih svaki ima iste statičke biblioteke unutar sebe, praktično imamo deset kopija tih istih biblioteka.
Samim time raste i veličina programa koji koristi statičke biblioteke (engl. *Static Libraries*).
Njihova prednost je u tome što se programi koji imaju ugrađene statičke biblioteke mogu lako kopirati na drugi sustav i pokrenuti bez problema, odnosno bez ovisnosti o vanjskim bibliotekama koje možda na odredišnom sustavu nemamo.

**Dijeljene** biblioteke
Programi koji koriste dijeljene biblioteke samo se referenciraju na njih, te svi programi koji trebaju određenu dijeljenu biblioteku imaju samo referencu na tu (pojedinu) dijeljenu biblioteku, koju ne moraju sadržavati unutar sebe.
Stoga je veličina ovakvih programa manja jer s drugim programima dijele zajedničke dijeljenje biblioteke.

Mana ovog modela je u tome što prebacivanjem odnosno kopiranjem programa koji koristi dijeljene biblioteke na drugo računalo, na drugom računalu program neće raditi, ako na njemu nisu dostupne sve dijeljene biblioteke koje su mu potrebne.

#### 11.2.2.1. Kako to radi

***Slijedi napredno poglavlje!***
Tijekom pokretanja svakog programa koji koristi dijeljene biblioteke (u pravilu je to većina programa) prvo se provjerava može li program doći do dijeljenih biblioteka koje su mu potrebne. Slično kao što se za izvršne datoteke provjerava sistemska varijabla `PATH` i provjeravaju sve putanje direktorija navedene u njoj.
U tom slučaju se traži dostupnost izvršnih programa u bilo kojoj navedenoj putanji, tako se i za sve dijeljene biblioteke pretražuju točno definirane putanje do direktorija (mapa) koji ih eventualno sadrže.
Standardno se za većinu distribucija Linuxa provjerava dostupnost dijeljenih biblioteka unutar sljedećih direktorija (mapa):

- `/lib/`
- `/lib64/`
- `/usr/lib/`
- `/usr/lib64/`

Dakle svaki program pri pokretanju prvo provjerava može li pronaći sve potrebne dijeljene biblioteke u gore navedenim direktorijima te ako ne može, izbacuje grešku s informacijom koja mu točno biblioteka nedostaje.
Zapravo se o dostupnosti i pokretanju dijeljenih biblioteka za svaki program brine proces koji se zove "***Dynamic Linker***".

S naredbom `ldconfig` možemo vidjeti koji direktoriji (mape) su dodani kao izvori dijeljenih biblioteka te koje su to biblioteke u njima. Pogledajmo skraćeni ispis ove naredbe:

```
ldconfig -v

/lib:
        libc.so.6 -> libc-2.12.so
        libutil.so.1 -> libutil-2.12.so
        libselinux.so.1 -> libselinux.so.1
        libz.so.1 -> libz.so.1.2.3

... ... ...

/lib64:
        libplds4.so -> libplds4.so
        libsemanage.so.1 -> libsemanage.so.1
        libdbus-1.so.3 -> libdbus-1.so.3.4.0
        libsepol.so.1 -> libsepol.so.1
        libc.so.6 -> libc-2.12.so
        libutil.so.1 -> libutil-2.12.so
        libselinux.so.1 -> libselinux.so.1
... ... ...

/usr/lib:
        libgamin-1.so.0 -> libgamin-1.so.0.1.10
        libformw.so.5 -> libformw.so.5.7
        libform.so.5 -> libform.so.5.7
        libtic.so.5 -> libtic.so.5.7
        libmenu.so.5 -> libmenu.so.5.7
        libpanelw.so.5 -> libpanelw.so.5.7
        libxml2.so.2 -> libxml2.so.2.7.6
... ... ...
```

U normalnom radu, tijekom instalacije novog programa koji prema potrebi instalira i nove dijeljene biblioteke, sustav ih u pravilu kopira upravo u neki od gore navedenih direktorija (ili pôddirektorija).



Dijeljene biblioteke u **Linuxu** imaju ekstenziju `.so` (engl. *Shared Object*), koja označava „dijeljeni objekt“.
U operativnom sustavu **Windows**, one imaju ekstenziju `.dll` (engl. *Dynamic link library*) te istu funkcionalnost.

**Izvori informacija:** **`ldconfig --help`**, **`man ldd`**, **`man ldconfig`**

### 11.2.3. Rad s dijeljenim bibliotekama

***Slijedi napredno poglavlje (11.2.3.x)!***

***Dynamic linker*** je dio operativnog sustava koji tijekom pokretanja svakog programa koji koristi dijeljene biblioteke, prije sâmog pokretanja programa, pokušava pronaći sve dijeljene biblioteke koje su mu potrebne za rad.
On potom sve tražene dijeljene biblioteke učitava s diska u RAM memoriju i kreira sve potrebno kako bi se program mogao uspješno izvršiti te pristupiti tim bibliotekama u RAM memoriji. Drugim riječima on dinamički povezuje dijeljene biblioteke s našim (pokrenutim) programom. Ovdje postoji još jedan pojam koji bi bilo dobro razumjeti.
**Linking** ili povezivanje je proces koji povezuje sve biblioteke u trenutku kompiliranja/prevođenja (engl. *Compile*) programa, kako bi se program uopće mogao prevesti u izvršni kôd, koji mi kao korisnici vidimo kao izvršnu odnosno binarnu datoteku.



U Linux sustavu se izvršne binarne datoteke zovu "***ELF***" datoteke (engl. *Executable and Linkable Format*).
Za više detalja otvorite poveznicu: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format.

Pogledajmo primjerice informacije o našoj *BASH* ljusci, koja je također kompilirana i dolazi nam kao binarna izvršna datoteka. Za ovu potrebu ćemo koristiti naredbu `file` koja nam daje informacije o vrsti (tipu) datoteke koju provjeravamo.

Dakle naša ***bash*** ljuska je zapravo izvršna datoteka: `/bin/bash`. Stoga pokrenimo sljedeću naredbu:

```
file /bin/bash

/bin/bash: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.18, stripped
```

Vidimo kako je vrsta ove izvršne datoteke: **ELF** te da je dinamički povezana (*`dynamically linked`*) što znači kako koristi dijeljene biblioteke na koje se oslanja odnosno koje su joj potrebne za rad.

#### 11.2.3.1. Dodavanje novih direktorija za dijeljene biblioteke

Vratimo se na naredbu `ldconfig`. Naime naredba `ldconfig` kod prvog pokretanja (inicijalizacije) zapravo prosljeđuje putanje do svih dostupnih biblioteka *Dynamic Linkeru* koji ih kasnije može koristiti za potrebe svih programa.
To znači kako se ova naredba ne treba pokretati, ako nismo dodali novi direktorij za dijeljene biblioteke. Kod instalacije novih programa koji tijekom sâme instalacije instaliraju i nove dijeljene biblioteke u standardne direktorije, nije potrebno napraviti ništa dodatno. Čak i u slučaju kada se instaliraju nove biblioteke u neki novi direktorij, u većini slučajeva instalacijska procedura odrađuje ostatak posla odnosno dodaje novi direktorij u *putanju* za dijeljene biblioteke.
Primjer ovakvog rada su primjerice *mySQL* ili *MariaDB* relacijske baza podataka, odnosno njihova instalacija ili instalacija klijenata za ove baze, koja instalira nekoliko dijeljenih biblioteka u novi direktorij, koji do tog trenutka nije postojao.
Stoga instalacijska procedura kreira novi unos koji govori *Dynamic Linkeru* kako se pojavio dodatni direktorij s dijeljenim datotekama.

Pogledajmo kako izgleda lista direktorija za dijeljene biblioteke nakon instalacije ***mySQL*** baze podataka (izbacili smo veći ispisa kako bismo se fokusirali na ono što nam je ovdje važno):

```
ldconfig -v

/usr/lib:
        libgamin-1.so.0 -> libgamin-1.so.0.1.10
        libformw.so.5 -> libformw.so.5.7
/usr/lib64:
        libcups.so.2 -> libcups.so.2
        libLLVM-3.4-mesa.so -> libLLVM-3.4-mesa.so
        libFLAC++.so.6 -> libFLAC++.so.6.2.0
/usr/lib64/mysql:
        libmysqlclient_r.so.16 -> libmysqlclient_r.so.16.0.0
        libmysqlclient.so.16 -> libmysqlclient.so.16.0.0
```

Vidimo kako se pojavio direktorij: `/usr/lib64/mysql` s dijeljenim bibliotekama unutar njega.

**Ali kako to radi?**
Unutar direktorija: `/etc/ld.so.conf.d/` se nalaze datoteke; po jedna za svaki program koji dodaje svoje biblioteke u neki zaseban (novi) direktorij, a koji nije neki od standardnih poput: `/lib/`, `/lib64/`, `/usr/lib/` ili `/usr/lib64/`.
U našem slučaju ***mySQL*** je ovdje kreirao svoju datoteku imena: `mysql-x86_64.conf`.
Pogledajmo sadržaj ove datoteke:

```
cat /etc/ld.so.conf.d/mysql-x86_64.conf
/usr/lib64/mysql
```

Vidimo kako ona sâmo ima upisanu novu putanju direktorija (mape) unutar kojeg se nalaze njene dijeljene biblioteke.

Nakon sâmog kreiranja nove datoteke, ništa se nije promijenilo te je potrebno pokrenuti naredbu `ldconfig` bez parametara, kako bi ona mogla dodati novi direktorij *Dynamic Linkeru*. To bi trebalo napraviti ovako:

```
ldconfig
```

U slučaju instalacije ***mySQL***-a on je to već napravio na kraju instalacije, za nas.

Međutim, ako imamo potrebu sâmi dodavati novi direktorij s nekim našim bibliotekama, procedura je ista i svodi se na:

1. Kreiranje nove datoteke unutar direktorija: `/etc/ld.so.conf.d/`
2. Pokretanje naredbe `ldconfig` kako bi se osvježila lista direktorija za dijeljene biblioteke.

**Izvor informacija: `ldconfig --help`,`man ldd`,`man ldconfig`**

## 11.2.3.2. Što još treba znati

Nakon što se pokrene naredba `ldconfig` ona inicijalno radi sljedeće:

- Prvo provjerava konfiguracijsku datoteku: `/etc/ld.so.conf` u kojoj se nalazi definicija gdje je sve potrebno tražiti nove konfiguracijske datoteke za nove putanje (direktorije). Ovdje se standardno nalazi samo definiran direktorij `/etc/ld.so.conf.d/` sa svim datotekama unutar njega, koje imaju ekstenziju `.conf`
- Potom se kreira lista svih dijeljenih biblioteka koje su pronađene iz svih navedenih putanja. Lista se generira u datoteku `/etc/ld.so.cache`. Ova datoteka je u posebnom formatu te ju nije moguće (jednostavno) pročitati.

***Dynamic linker*** se zapravo nalazi u datoteci:

- `/lib/ld-linux.so.*` za 32.bitne Linuxe.
- `/lib64/ld-linux-x86-64.so.*` za 64.bitne Linuxe.

Sadržaj gore navedene *cache* datoteke `ld.so.cache` je moguće i izlistati, što znači izlistanje svih dijeljenih biblioteka koje se nalaze u njemu. U ovoj datoteci se nalaze i sve datoteke koje su trenutno učitane u RAM memoriju. Naredba `ldconfig -p` nam daje listu svih dijeljenih biblioteka koje se trenutno nalaze učitane u RAM memoriju. Sada to i pogledajmo:

```
ldconfig -p

605 libs found in cache `/etc/ld.so.cache'
        libz.so.1 (libc6,x86-64) => /lib64/libz.so.1
        libz.so.1 (libc6) => /lib/libz.so.1
        libz.so (libc6,x86-64) => /usr/lib64/libz.so
        libyajl.so.1 (libc6,x86-64) => /usr/lib64/libyajl.so.1
        libxul.so (libc6,x86-64) => /usr/lib64/xulrunner/libxul.so
        libxtables.so.4 (libc6,x86-64) => /lib64/libxtables.so.4
        libxslt.so.1 (libc6,x86-64) => /usr/lib64/libxslt.so.1
        libxpcom.so (libc6,x86-64) => /usr/lib64/xulrunner/libxpcom.so
        libxml2.so.2 (libc6,x86-64) => /usr/lib64/libxml2.so.2
        libxml2.so.2 (libc6) => /usr/lib/libxml2.so.2
        libxcb.so.1 (libc6,x86-64) => /usr/lib64/libxcb.so.1
        libxcb.so (libc6,x86-64) => /usr/lib64/libxcb.so
        . . .
```

Ovdje ćemo vidjeti listu svih učitanih (u RAM memoriju) dijeljenih biblioteka.

## 11.2.3.3. Koje dijeljene biblioteke su nam potrebne za koji program

U slučajevima kada želimo za određenu izvršnu binarnu datoteku (***ELF***) provjeriti koje dijeljene biblioteke su joj potrebne, za tu namjenu možemo koristiti naredbu `ldd` (engl. *List Dynamic Dependencies*). Pogledajmo recimo *BASH* ljusku (njena izvršna datoteka je `/bin/bash`).

Naredbi `ldd` je potrebno proslijediti punu putanju do željene datoteke, i to ovako:

```
ldd /bin/bash
        linux-vdso.so.1 =>  (0x00007ffc6eeef000)
        libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007feff03f8000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007feff01f4000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fefefe5f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007feff0626000)
```

Vidimo kako je za *BASH* ljusku, tj. izvršnoj datoteci koja pokreće ljusku `/bin/bash` potrebno nekoliko dijeljenih biblioteka za rad. Osim toga točno za svaku od njih vidimo gdje je locirana na disku (`=> /lib/*` …) te na kojoj memorijskoj lokaciji unutar sustava virtualne memorije se i nalazi (`0xYYYYYYY`).

Jedna od važnijih posebnih biblioteka sustava koje ovdje vidimo je `linux-vdso.so.1`.
***VDSO*** odnosno engl. „*virtual dynamic shared object*" je mala dijeljena biblioteka koju koristi kernel, tako da se ona preslikava u adresni prostor dostupan svim programima (aplikacijama). Same aplikacije u svom radu se ne moraju brinuti o *VDSO* komponenti jer se ona ionako poziva unutar programskih biblioteka na nižoj razini odnosno na razini biblioteka za **C** programski jezik. To znači da se aplikacije programiraju na normalan način koristeći standardne funkcije, a u pozadini će se **C** biblioteke pobrinuti za korištenje bilo koje funkcionalnosti koja je dostupna putem vDSO-a. Naime u radu aplikacija koriste se određeni sistemski pozivi koje kernel treba osigurati aplikacijama, unutar njihovog adresnog prostora (*user space*), do kojih se dolazi preko *VDSO* komponente. Bez *VDSO* komponente, korištenje tih sistemskih poziva bilo bi vrlo sporo i neučinkovito.
O *VDSO* malo detaljnije u sljedećoj cjelini.
Druge biblioteke odnosno njihova upotreba ovisi o samoj aplikaciji i njenim potrebama.

Osim naredbe `ldd` za provjeru ovisnosti o dijeljenim bibliotekama moguće je istu funkcionalnost dobiti direktno i s *Dynamic Linkerom*. Pogledajmo iste primjere za *BASH* ljusku i to prvo za 32.bitni linux:

```
/lib/ld-linux.so.2 --list /bin/bash

        linux-vdso.so.1 =>  (0x00007ffd38ba4000)
        libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007fddff909000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007fddff705000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fddff370000)
        /lib/ld-linux.so.2 (0x00007fddffb37000)
```

A potom za 64.bitni linux:

```
/lib64/ld-linux-x86-64.so.2 --list /bin/bash

        linux-vdso.so.1 =>  (0x00007ffc7adaf000)
        libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007feec1754000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007feec1550000)
        libc.so.6 => /lib64/libc.so.6 (0x00007feec11bb000)
        /lib64/ld-linux-x86-64.so.2 (0x00007feec1982000)
```

U slučaju kada već imamo pokrenut neki program (proces), moguće je i od njega saznati koje sve biblioteke koristi za rad.

Za tu namjenu se koristi naredba `pldd`, kojoj kao parametar moramo dati ***PID*** broj pokrenutog programa (procesa), primjerice:

```
pldd 814

814:    /usr/sbin/rsyslogd
linux-vdso.so.1
/lib64/libz.so.1
/lib64/libpthread.so.0
/lib64/libdl.so.2
/lib64/librt.so.1
/lib64/libestr.so.0
/lib64/libfastjson.so.4
/lib64/libsystemd.so.0
/lib64/libuuid.so.1
/lib64/libgcc_s.so.1
/lib64/libc.so.6
/lib64/ld-linux-x86-64.so.2
/lib64/liblzma.so.5
/lib64/liblz4.so.1
/lib64/libcap.so.2
/lib64/libmount.so.1
/lib64/libgcrypt.so.20
/lib64/libblkid.so.1
/lib64/libselinux.so.1
/lib64/libgpg-error.so.0
/lib64/libpcre2-8.so.0
/usr/lib64/rsyslog/lmnet.so
/usr/lib64/rsyslog/imuxsock.so
/usr/lib64/rsyslog/imjournal.so
/usr/lib64/rsyslog/imudp.so
. . .    . . .
```

Dakle u konkretnom slučaju vidimo da je proces s ***PID*** brojem **814**, pokrenuti ***syslog*** servis te za njega vidimo koje sve programske biblioteke koristi (ispis smo skratili).

**Izvori informacija:** **`ldconfig --help`**,**`man ldd`**,**`man pldd`**,**`man 7 vdso`**.

### 11.2.3.4. Napredno o dijeljenim bibliotekama

Osim lokacije (putanje na disku) dijeljenih biblioteka, vidljive su još dvije stvari:

- `linux-vdso.so.1` biblioteka nema svoju lokaciju na disku. Odakle se onda učitala?
- **Memorijske lokacije** za iste dijeljene biblioteke su tijekom svakog pozivanja drugačije. Zbog čega?

Već spomenut: `linux-vdso.so.1` ili `vdso` (Virtual Dynamic Shared Object) predstavlja virtualnu dijeljenu biblioteku, koju nam dodjeljuje linux kernel i ubacuje je u memorijski adresni prostor našeg programa odnosno aplikacije. Ova virtualna dijeljena biblioteka je nekada vidljiva i kao `linux-gate.so.1`.

Osnovna funkcija ove virtualne biblioteke je omogućavanje pristupa rutinama koje su implementirane na razini kernela i izvorno se nalaze u izoliranom memorijskom prostoru kernela (engl. *Kernel Space*), ali ih kernel preslikava u adresni prostor programa/aplikacije koja se izvršava.
On nam praktično daje mogućnost pozivanja sistemskih poziva koji su nužni za rad programa odnosno aplikacije. Pristup ovim rutinama (pozivima) na ovakav način je važan jer se za neke rutine (funkcionalnosti) tako preskaču mnogi slojevi linux komponenti, koje u slučaju ovih funkcionalnosti nisu potrebne. Pošto se sve događa preko *VDSO* modela, sve je i dalje sigurno i zaštićeno jer nigdje nema direktnog pristupa izoliranom memorijskom prostoru linux kernela, što je druga svrha ovakvog rada.

**Memorijske lokacije** na kojim se nalaze dijeljene biblioteke se mijenjaju jer tijekom inicijalizacije programa, *Dynamic Linker* dodjeljuje svaki puta novi pokazivač na novu memorijsku lokaciju. Naime svaki put kada ponovno pokrenemo program odnosno aplikaciju ona se učitava na neku drugu memorijsku lokaciju.
Ovo ne znači da će se i programu koji je već pokrenut promijeniti memorijske adrese biblioteka u radu. To znači samo to kako će se tijekom svakog novog pokretanja bilo kojeg programa, memorijska lokacija na koju će se učitati dijeljene biblioteke ili dijelovi programa, učitati u drugi dio virtualne memorije, a sve iz sigurnosnih razloga (ovo odrađuje ***ASLR*** mehanizam zaštite). Dakle ovo učitavanje programa se događa sâmo tijekom prvog pokretanja programa ili nakon njegovog zaustavljanja, pa ponovnog pokretanja.
Pogledajmo našu BASH ljusku, ali ovaj put onu u kojoj radimo. Prvo pronađimo njen ***PID*** broj na sljedeći način:

```
pidof bash

19849
```

Sada ćemo korištenjem **/proc** sustava ispisati sve memorijske lokacije koje koristi naš trenutni BASH proces sa ***PID***-om `19849`.
Dakle za svaki Proces ID (***PID***), unutar `/proc/` direktorija, nalazi se datoteka koja sadrži tablicu preslikavanja (mapiranja) memorije, imena `map`. U njoj su zapisana preslikavanja memorije na sljedeći način:

***Memorijska adresa (lokacija) u virtualnoj memoriji ←→ biblioteka ili proces/naredba/datoteka***

Pogledajmo sadržaj `map` datoteke za naš proces (***PID***=`19849`), s naredbom `cat`:

```
cat /proc/19849/maps

00400000-004d4000         r-xp 00000000 fd:02 15207210          /bin/bash
006d4000-006dd000         rw-p 000d4000 fd:02 15207210          /bin/bash
006dd000-006e3000         rw-p 00000000 00:00 0
0196d000-019cf000         rw-p 00000000 00:00 0                 [heap]
7fc7f0673000-7fc7f067f000 r-xp 00000000 fd:02 15206978  /lib64/libnss_files-2.12.so
7fc7f067f000-7fc7f087f000 ---p 0000c000 fd:02 15206978  /lib64/libnss_files-2.12.so
7fc7f087f000-7fc7f0880000 r--p 0000c000 fd:02 15206978  /lib64/libnss_files-2.12.so
7fc7f0880000-7fc7f0881000 rw-p 0000d000 fd:02 15206978  /lib64/libnss_files-2.12.so
7fc7f0881000-7fc7f6712000 r--p     00000000  fd:02 15470584  /usr/lib/locale/locale-archive-rpm
7fc7f6712000-7fc7f689c000 r-xp 00000000 fd:02 15206962  /lib64/libc-2.12.so
7fc7f6a9c000-7fc7f6aa0000 r--p 0018a000 fd:02 15206962  /lib64/libc-2.12.so
7fc7f6aa0000-7fc7f6aa1000 rw-p 0018e000 fd:02 15206962  /lib64/libc-2.12.so
7fc7f6aa6000-7fc7f6aa8000 r-xp 00000000 fd:02 15206968  /lib64/libdl-2.12.so
7fc7f6ca9000-7fc7f6caa000 rw-p 00003000 fd:02 15206968  /lib64/libdl-2.12.so
7fc7f6caa000-7fc7f6cc7000 r-xp 00000000 fd:02 15210225  /lib64/libtinfo.so.5.7
7fc7f6cc7000-7fc7f6ec6000 ---p 0001d000 fd:02 15210225  /lib64/libtinfo.so.5.7
7fc7f6ecb000-7fc7f6eeb000 r-xp 00000000 fd:02 15204365  /lib64/ld-2.12.so
7fc7f70e2000-7fc7f70e9000     r—s  00000000  fd:02 16124866  /usr/lib64/gconv/gconv-modules.cache
7fc7f70e9000-7fc7f70ea000 rw-p 00000000 00:00 0
7fc7f70ea000-7fc7f70eb000 r--p 0001f000 fd:02 15204365  /lib64/ld-2.12.so
7fc7f70eb000-7fc7f70ec000 rw-p 00020000 fd:02 15204365  /lib64/ld-2.12.so
7fffc700a000-7fffc701f000 rw-p 00000000 00:00 0         [stack]
7fffc7147000-7fffc7149000 r-xp 00000000 00:00 0         [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0    [vsyscall]
```

Za više detalja o preslikavanju memorije i sustavu virtualne memorije pročitajte sljedeće poglavlje.

**Izvori informacija: (246),(630),** **`ldconfig –help`**,**`man ldd`**,**`man pidof`**,**`man 5 procfs`**.

# 12. Sustav virtualne memorije i memorijski menadžment

Sustav virtualne memorije je vrlo važna komponenta zadužena za rad operativnog sustava računala.
No kako bi ovaj sustav uopće funkcionirao, potrebna mu je RAM odnosno *radna* memorija (engl. *Random Access Memory*).
RAM memorija je hardverska komponenta računala koja se koristi za zapisivanje i čitanje podataka: od pokretanja programa, odnosno zapisivanja programa u nju, zapisivanja međurezultata rada tih programa, te na kraju njihovog čitanja iz RAM memorije.
Jednostavno rečeno, sve što radimo na računalu prvo se učitava u RAM memoriju i radi u RAM memoriji: od operativnog sustava i svih upravljačkih programa do svih naših programa (aplikacija) koje koristimo.

**Pohrana podataka u RAM memoriju**
Na najnižoj hardverskoj razini podaci se u RAM memoriju pohranjuju ovisno o arhitekturi procesora. Primjerice kod **x86, ARM i RISC-V** arhitektura procesora se koristi takozvani ***little-endian*** način pohrane. To znači da kada se niz podataka (bajtova) pohranjuje u memoriju, bajt najmanje važnosti dolazi prvi. Što znači da je redoslijed bajtova podataka efektivno obrnut kada se pohranjuje u memoriji. Ako se primjerice treba pohraniti heksadecimalni niz: **`0x AB CD EF 11`**, on će tada u memoriji biti pohranjen kao: **`0x 11 EF CD AB`**. Dok se kod ***big-endian*** načina pohrane podaci pohranjuju bez obrtanja.

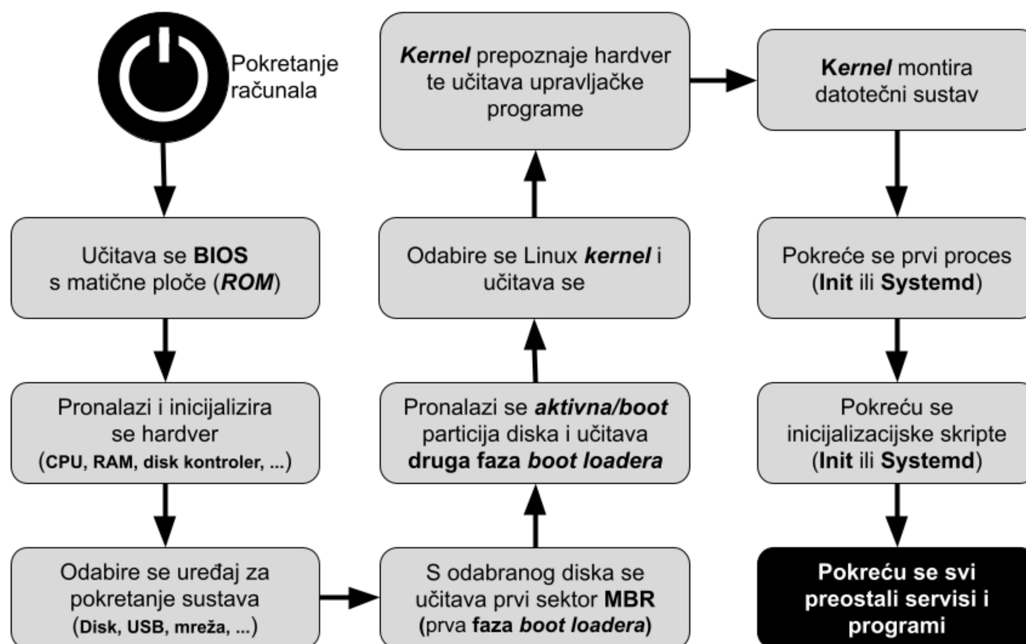
RAM memorija specifična je i po tome što je brzina čitanja i pisanja jednaka, neovisno o poziciji na koju se zapisuje.
Za razliku od drugih vrsta memorija poput tvrdog diska, CD/DVD i sličnih medija, koji imaju ova ograničenja zbog vrtnje ploča diska ili pomjeranja glave za čitanje/pisanje. Osim toga RAM memorija je vrlo brza, točnije najbrža memorija koju imamo u računalu. Vrijeme pristupa RAM memoriji se mjeri u nano sekundama, dok se primjerice vrijeme pristupa tvrdog diska podacima, mjeri u milisekundama, što je oko 1.000.000.000 puta sporije. Osim toga i brzina prijenosa podataka u i iz RAM memorije je vrlo brza te također najbrža u našem računalu. Tako današnje DDR4 RAM memorije imaju propusnost od preko 50 GB/s (giga bajta u sekundi). Važno je znati kako se sadržaj RAM memorije gubi kod gašenja računala jer se RAM memorija mora stalno osvježavati odnosno napajati.

---

## 12.1. Što se događa kod pokretanja računala (*boot process*)

Nakon uključivanja odnosno pokretanja računala inicijalizira se ***BIOS*** (*Basic Input Output System*) koji je ugrađen u matičnu ploču računala. ***BIOS*** pronalazi tvrdi disk s kojeg pokreće učitavanje prve faze tzv. *boot loadera*, s prvog sektora diska, koji se naziva *Master Boot Record* (**MBR**). Potom se učitava slijedeća faza *boot loadera* obično s takozvane ***aktivne*** (*boot*) particije sistemskog diska, odnosno one koja je označena kao takva. Međutim ponekad postoji i još jedna međufaza (1.5.) koja učitava drugu fazu *boot loadera*, što je slučaj kod primjerice ***GRUB2*** *boot loadera* koji koriste ***Red Hat*** kompatibilne distribucije Linuxa. Na kraju druga faza *boot loadera* (koja je obično i zadnja) odabire i učitava Linux kernel. Linux kernel prepoznaje sâv hardver i učitava sve potrebne upravljačke programe te montira datotečni sustav u vršni direktorij (*/*). Datotečni sustav je od tog trenutka dostupan za korištenje, sa svim direktorijima (mapama) i datotekama na njemu. Potom se pokreće prvi program (***PID*** broja 1) koji se zove ***Init*** ili ***Systemd*** (ovisno o Linuxu). Linux sustav zatim učitava i druge pomoćne programe i biblioteke. Učitavaju se i dodatni servisi, a na kraju se učitavaju i svi ostali potrebni programi za normalno funkcioniranje operativnog sustava. Cijeli proces pokretanja sustava vidljiv je na slici **88.1**.

**Slika 88.1. Proces pokretanja računala (tzv. *boot* proces)**



Važno je razumjeti kako se sve navedene programske komponente učitavaju u RAM memoriju te se ona intenzivno koristi, kako od početka rada računala i inicijalizacije BIOS-a, tako i u daljem radu sustava: od učitavanja i rada posebnih sistemskih servisa sve do korisničkih programa.

## Što se detaljnije događa tijekom pokretanja računala?

*Basic Input/Output System* (**BIOS**) osnovni je ulazno/izlazni sustav, a također je poznat kao, **ROM BIOS**, **BIOS ROM** ili **PC BIOS**. **BIOS** (*firmware)* dolazi unaprijed instaliran na matičnoj ploči računala i on je praktično prvi softver koji se pokreće kada se računalo uključi. ***BIOS*** je zapravo mali program koji je ugrađen u ROM memoriju na matičnoj ploči računala, koji prvo služi za inicijalizaciju hardvera na najnižoj razini; od inicijalizacije tipkovnice, miša, disk kontrolera i tvrdog diska, CD-DVD-ROM-a, do CPUa i memorije, ali i sve druge periferije.

Naime u inicijalnom trenutku nakon što se računalo uključi, centralni procesor računala (CPU) nema nikakav softver u **RAM** memoriji, tako da neki mehanizam mora učitati osnovni softver odnosno osnovni inicijalni (*mikro*) operativni sustav u memoriju, a prije nego što se išta drugo može izvršiti. Stoga CPU učitava ono što bi mogli nazvati i *mikro* operativnim sustavom odnosno *BIOS-om*, koji je pohranjen u posebnoj vrsti takozvane ROM (engl. *Read Only Memory*) memorije na matičnoj ploči računala (u *EPROM* čipu).
Dakle CPU prvo počinje učitavati i izvršavati programski kôd s određene adrese unutar ROM memorije BIOS-a, koji nakon pronalaska i inicijalizacije komponenti računala te inicijalizacije RAM memorije kreće s pronalaženjem diska s kojeg može pokrenuti operativni sustav. Naime namjena *BIOSa* ovdje je testiranje i inicijalizacija svih hardverskih komponenti računala te pokretanje prve faze sekvence za učitavanje ostatka sustava s diska. Prvi korak je traženje uređaja koji ispunjavaju uvjete za sudjelovanje u pokretanju sustava (*boot process*) te učitavanje malog programa iz posebnog sektora na disku.
Ovdje se zapravo radi o prvom sektoru na disku koji se naziva *Master Boot Record* (**MBR**), koji je veličine:

- **512** bajta za diskove čija veličina bloka je 512 bajta (obično su to diskovi do 2TB).
- **2048** bajta za CD-ROM i DVD-ROM uređaje
- I na novijim diskovima, iako su im svi podatkovni sektori 4096 bajta (4kB), oni i dalje zbog kompatibilnosti koriste početni sektor od 512 bajta.

Navedeni mali program koji pokreće ovu sekvencu pokretanja sustava poznat je kao ***bootstrap loader***, ***bootstrap*** ili ***boot loader***. Dakle namjena prvog sektora na disku je za smještanje prve faze ***boot loadera*** (primjerice kod upotrebe ***GRUB*** *boot loadera*).
Međutim u praksi se često koriste više stupanjski ***boot loaderi***, tijekom kojih se nekoliko programa, sve veće složenosti, učitava jedan za drugim u procesu lančanog učitavanja. To konkretno znači da je prva faza **boot loadera**, pohranjena u prvi sektor na disku, u prvih 512 bajta. Ako kao ***boot loader*** koristimo *GNU GRand Unified Bootloader* (***GRUB),*** odnosno njegovu noviju inačicu naziva ***GRUB2***, što je u današnje vrijeme čest slučaj za većinu distribucija Linuxa, to znači da je njegova prva komponenta instalirana upravo u ovaj prvi sektor na disku, koji se naziva *Master Boot Record* (**MBR**) odnosno glavni zapis za pokretanje sustava.

***GRUB(2)*** zapravo koristi trostupanjski način rada, što znači da *Master Boot Record* (**MBR**) sadrži prvi stupanj ***GRUB*** *boot loadera* (engl. *Stage 1 boot loader*). Pošto je prostor u ***MBR*** zapisu vrlo ograničen (512 bajta), ***GRUB*** ovdje ima samo mehanizam za učitavanje sljedećeg stupnja *boot loadera* te osnovne podatke koji se obavezno i moraju ovdje nalaziti:

- Popis particija, s definicijom na kojem sektoru koja od njih počinje, a gdje završava.
- Popis vrsta particija, za svaku particiju zasebno, te definiciju koja particija je ***aktiva*** (tzv. ***boot*** particija), s koje se može nastaviti proces pokretanja sustava, a na kojoj se nalazi druga faza *boot loadera* i operativni sustav (Linux).



Vezano za ***boot loader*** (konkretno ***GRUB2***), pogledajte poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

Za detalje o **MBR** zapis na disku, pogledajte poglavlje:
**13.4.1. Napredno: MBR Detaljnije.**

Što se tiče ***boot loadera***, pogledajte i poglavlje:
**13.4. Logička shema diska.**

***Boot loaderi*** drugog stupnja, kao što je ***GRUB***, ali i ***rEFInd***, ***BOOTMGR***, ***Syslinux***, ***NTLDR*** ili ***iBoot***, sami po sebi nisu operativni sustavi, niti im je to namjena. Njihov zadatak je učitati operativni sustav.
***Boot loader*** drugog stupnja pri tome za pokretanja ne treba upravljačke programe za pristup disku s kojeg se učitava, već umjesto toga može koristiti generičke (pojednostavljene) metode pristupa diskovnom sustavu, odnosno samo pojedinim direktorijima i datotekema nužnim za nastavak njegovo učitavanja.

***Boot loaderi*** drugog stupnja često se pohranjuje na onu particiju diska koja je označena kao ***aktivna*** (***boot*** particija), kako je već definirano u **MBR** zapisu. Međutim to nije slučaj kod *GRUB2* koji ima još jedan međustupanj.

Konkretno, ***boot loaderi*** drugog stupnja se često upisuju u prvi sektor aktivne particije, dakle u prvi sektor unutar particije, koji se naziva *volume boot record* (**VBR**), *volume boot sector, partition boot record* ili *partition boot sector*.

Kada se ***boot loader*** drugog stupnja pokrenuo, on kreće s učitavanjem kernela operativnog sustava, na istoj particiji diska, na kojoj se i sâm nalazi. Kernel potom prepoznaje hardver, učitava upravljačke programe te montira (*mount*) datotečni sustav s cijelim Linuxom odnosno stablom direktorija i datoteka. Potom se pokreće prvi program pod Linuxom (***init*** i ***systemd***), koji učitava sve potrebne servise i programe. I konačno se pokreću preostali korisnički programi. Pri tome je važno razumjeti da se sve navedene komponente (BIOS, *boot loaderi*), kernel te programi i servisi Linuxa, učitavaju i rade u RAM memoriji.

Dakle RAM memorija se cijelo vrijeme intenzivno koristi. Sve što je od podataka potrebno sačuvati, zbog restarta ili gašenja računala, snima se na tvrdi disk. Međutim Linux zbog brzine rada pokušava sve što je moguće više stvari odnosno programa, servisa i pripadajućih podataka, držati u RAM memoriji, a minimalno koristiti tvrdi disk.



Zbog gore navedene optimizacije u radu Linuxa, tijekom urednog gašenja Linuxa, on sve što je potrebno iz RAM memorije uredno snima na tvrdi disk, a te tek potom gasi računalo. Problem nastaje kada se Linux nasilno ugasi: može se dogoditi da su svi vaši podaci ostali u RAM memoriji te da nisu snimljeni na tvrdi disk.

***Sjetimo se da se sadržaj RAM memorije gubi tijekom gašenja računala!.***

**Izvori informacija: (907),(908),(1000),(1085),(1086),(1087),(1088),(1089), `man 7 bootup`, `man 7 bootparam`.**

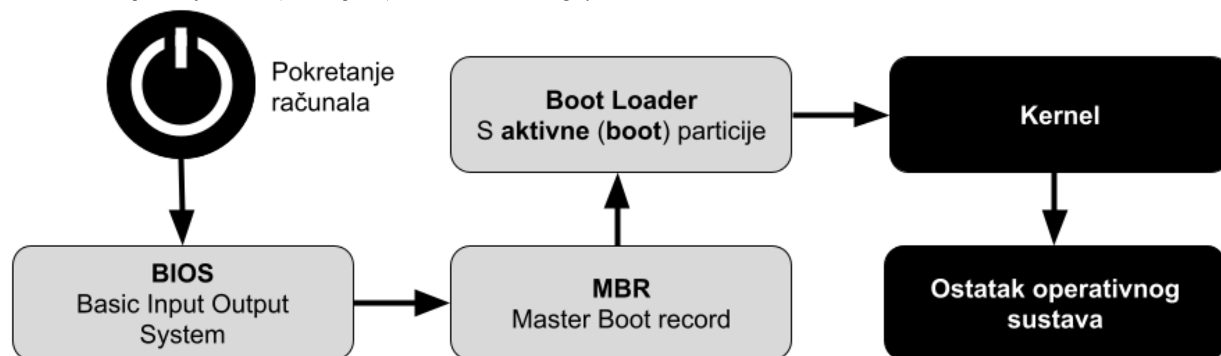
## 12.1.1. BIOS vs UEFI

**Slijedi napredna cjelina!**

Naučili smo da je *Basic Input/Output System* (**BIOS**) osnovni ulazno/izlazni sustav, koji je također poznat kao, **ROM BIOS**, **BIOS ROM** ili **PC BIOS**. **BIOS** (*firmware)* dolazi unaprijed instaliran na matičnoj ploči osobnog računala i on je praktično prvi softver koji se pokreće kada se računalo uključi. **BIOS** prvo pronalazi, inicijalizira i testira (***POST*** metoda) hardver: matičnu ploču i čipset, RAM, CPU te ostale komponente računala. On nakon inicijalizacije hardvera i pronalaska diskovnog uređaja koji je namijenjen za pokretanje sustava, učitava prvu fazu ***boot loadera*** koji se nalazi na prvom sektoru **(MBR)** diska.

Pošto je u ***BIOS***-u moguće definirati redoslijed uređaja s kojih se može pokušati pokrenuti sustav (*boot proces*), **BIOS** provjerava svaki od ovih uređaja redom, kako bi vidio može li se s njih pokrenuti sustav, pokušavajući s prvog uređaja učitati prvi sektor na disku odnosno ***master boot sektor*** (**MBR**).
Ako se s ***boot sektora*** prvog diska ne može ništa učitati, **BIOS** prelazi na sljedeći uređaj.
Procedura pokretanja računala u **BIOS** načinu rada (tzv. *Legacy boot*) je pojednostavljeno vidljiva na slici **88.2.**

**Slika 88.2. Proces pokretanja računala (tzv. *boot* proces), u BIOS odnosno tzv. Legacy Boot načinu rada.**




## U čemu je problem?

Navedeni **BIOS**, s obzorom na svoju inicijalnu povijest razvoja iz 1975. godine, nosi i određena ograničenja.
Ograničenja BIOS-a su sljedeća:

- On radi u takozvanom **16-bitnom** „*realnom*“ načinu rada (baziranom na 16-bitnom *Intel* 8088 procesoru), koji se također naziva ***real address mode***, a to je način rada svih procesora kompatibilnih s x86 arhitekturom. Ovaj način rada dobio je ime po činjenici da adrese u ovom načinu rada uvijek odgovaraju stvarnim adresama u RAM memoriji. *Realni* način rada karakterizira 20-bitni segmentirani memorijski adresni prostor, koji nam dozvoljava adresiranje do maksimalno 1 MB memorije te osigurava neograničen izravan softverski pristup svoj adresabilnoj memoriji, I/O adresama i perifernom hardveru. *Realni* način rada nema mehanizme zaštite memorije, višezadaćnosti (*multitasking*) ili razine privilegija izvršavanja programskog koda.

- Pri implementaciji je potrebno programiranje BIOS-a u asemblerskom jeziku jer se BIOS posebno dizajnira za rad s točno određenim modelom računala to jest matične ploče i pripadajućeg sistemskog *čipseta*.
- Postoji ograničenje na veličinu diskova do maksimalne veličine od 2.2 TB (što je moguće zaobići od strane operativnog sustava).

Rješavanje ovih problema započelo je 1998. godine i u početku se novi sustav zvao ***Intel Boot Initiative***.
Kasnije je preimenovan u ***Extensible Firmware Interface*** (**EFI**), koji je zatim preimenovan u ***Unified EFI Forum***, koji je razvio specifikaciju danas poznatu pod nazivom: ***Unified Extensible Firmware Interface*** (**UEFI**).
Sučelje definirano specifikacijom EFI-ja uključuje tablice podataka koje sadrže informacije o platformi te usluge pokretanja i izvođenja (Engl. *boot and runtime services*) koje su dostupne *boot loaderu* operativnog sustava i sâmom operativnom sustavu.

**UEFI** *firmware* pruža nekoliko tehničkih prednosti u odnosu na tradicionalni **BIOS** sustav:

- Ima sposobnost direktnog pokretanja sustava s diskova većih od 2TB, ako koriste *GUID Partition Table* (**GPT**), iako se i dalje može koristiti **MBR** shema particioniranja diska.
- Nudi 32-bitno ili 64-bitno okruženje (za razliku od 16-bitnog kod **BIOS**-a), prije inicijalizacije operativnog sustava. Primjerice implementacija 64-bitnog **UEFI** *firmvera* podržavaju tzv. dugački način rada (Engl. *Long mode*), koji aplikacijama u okruženju prije pokretanja operativnog sustava omogućuje korištenje 64-bitnog adresiranja memorije, kako bismo dobili izravan pristup cijeloj RAM memoriji računala. Jedini zahtjev pri tome je da su ***UEFI firmware (loader)*** i ***boot loader*** operativnog sustava (ili kernel) isto bîtni (pr. oba moraju biti 32-bitna ili 64-bitna), mada je i to riješeno od Linux kernela 3.15, pa su moguće i kombinacije (32-bitno i 64-bitno).
- Modularni dizajn i kompatibilnost unatrag te programiranje **UEFI**-ja u ***C*** jeziku, a ne ***asemblerskom*** jeziku kao prije.

***UEFI*** sustavi mogu pristupiti **GPT** diskovima i pokretati se izravno s njih, što Linuxu omogućuje korištenje **UEFI** metoda pokretanja. Međutim za učitavanje Linuxa s *GPT* diskova na *UEFI* sustavima, potrebna je posebna **EFI** sistemska particija (tzv. **ESP**), koja sadrži **UEFI** aplikacije kao što su: *bootloader*, kernel operativnog sustava i pomoćni softver.
Takva se postavka obično naziva **UEFI-GPT**, dok se za **ESP** particiju na disku preporučuje da bude veličine najmanje 512 MB i formatirana s **FAT32** datotečnim sustavom radi maksimalne kompatibilnosti.



Zbog kompatibilnosti unatrag, određeni broj **UEFI** implementacija također podržava pokretanje sustava s diskova particioniranim s **MBR** to jest na stari način: instaliranjem ***boot loadeara*** u ***MBR***, te druge faze ***boot loadera*** u **BR** particije.



To je konkretno implementirano putem modula za podršku kompatibilnosti (Tzv. **CSM**) koji osigurava kompatibilnost sa starim **BIOS**-om. Ova metoda rada se naziva „**legacy BIOS**", a moguća je samo, ako ju podržava matična ploča (***UEFI***) računala.

U tom slučaju, pokretanje Linuxa na ***UEFI*** sustavima je isto kao i na starim sustavima temeljenim na ***BIOS***-u, ali je takav način rada potrebno odabrati ulaskom u ***UEFI*** izbornik, odmah nakon uključivanja računala.
To se obično postiže s tipkama: ***F1*** ili ***F2 ili F10*** ili ***Delete***.

# UEFI način rada

Za razliku od starog ***BIOS***-a, ***UEFI*** se ne oslanja direktno na sektore na disku za pokretanje sustava, umjesto toga on definira ***boot menadžer*** kao dio ***UEFI*** specifikacije. Kada se računalo uključi (pokrene), ***boot menadžer*** (***EFI boot loader***) provjerava konfiguraciju za pokretanje sustava i na temelju njenih postavki izvršava ***boot loader*** operativnog sustava ili direktno učitava kernel operativnog sustava, koji nastavlja s pokretanjem sustava.

Konfiguracija pokretanja je definirana u varijablama pohranjenim u **NVRAM** memoriji, uključujući varijable koje označavaju putanje datotečnog sustava do ***boot loader-a*** operativnog sustava ili kernela operativnog sustava.

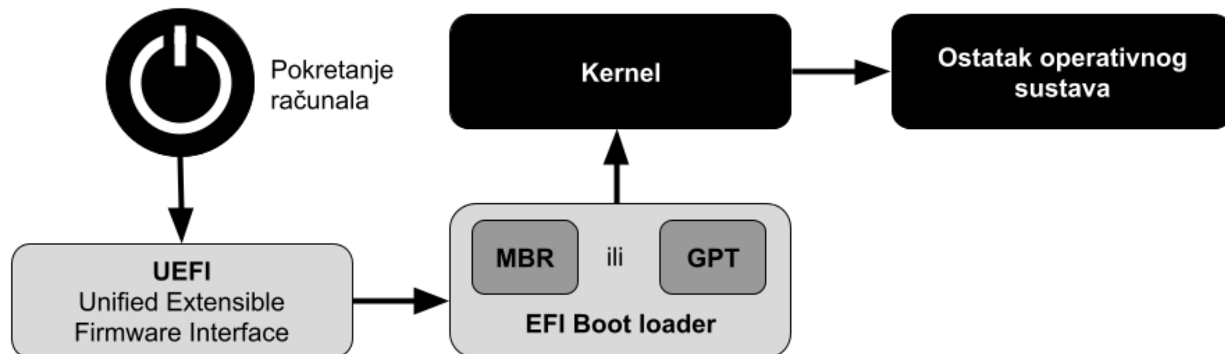
***UEFI*** može i automatski otkriti ***boot loader*** operativnog sustava, što omogućuje jednostavno pokretanje sustava s izmjenjivih uređaja kao što su ***USB*** *flash* pogoni. Ovo automatizirano otkrivanje oslanja se na standardizirane putanje do datoteka potrebnih za pokretanje sustava, pri čemu se putanje razlikuju ovisno o arhitekturi računala.

Format putanje datoteke definiran je kao:

**<EFI_particija>\/EFI\BOOT\BOOT<MACHINE_TYPE_SHORT_NAME>.EFI.**

To konkretno za x86 arhitekturu računala znači putanju (s datotekom), poput:
`/efi/boot/bootx64.efi`.

**Slika 88.3. Proces pokretanja računala (tzv. *boot* proces), u UEFI načinu rada.**




**Detalji o EFI (ESP) particiji**

**EFI** particija, često se naziva i **ESP**, particija je uređaja za pohranu podataka koja se koristi u računalima koja se pridržavaju **UEFI** specifikacije. Njoj pristupa *UEFI firmver* kada je računalo uključeno, a na nju se pohranjuju *UEFI* aplikacije i datoteke koje te aplikacije trebaju pokrenuti, uključujući *boot loader* operativnog sustava. Podržane sheme particioniranja uključuju i **MBR** i **GPT**, kao i takozvani *El Torito* (*ISO9660*) standard na optičkim diskovima. *UEFI* specifikacija definira da se **ESP** (*EFI*) particija može formatirati s: ***FAT32***, ***FAT16*** ili ***FAT12*** datotečnim sustavima.

Na **ESP** particiji je također moguće imati ***boot*** sektor za pokretanje sustava, kao što se koristilo u *BIOS* načinu rada, sve zbog kompatibilnosti unatrag. ***ESP*** particija sadrži ***boot loader*** ili kernel(e) za sve instalirane operativne sustave (koji se eventualno nalaze na drugim particijama), datoteke upravljačkih programa za hardverske uređaje koji su prisutni u računalu, a koje *firmware* koristi u vrijeme pokretanja te sistemske i uslužne programe koji su namijenjeni za pokretanje, prije pokretanja operativnog sustava.

Pokretanje **UEFI** sustava s diskova particioniranih s **GPT** obično se naziva **UEFI-GPT** pokretanje. Unatoč činjenici da **UEFI** specifikacija zahtijeva da koriste **GPT** particijske tablice odnosno particije, neke implementacije *UEFI firmware-a* odmah se prebacuju na pokretanje **CSM**-a temeljenog na **BIOS**-u ovisno o vrsti particijske tablice diska za pokretanje sustava (*boot particije*), čime se učinkovito zaobilazi potreba za pokretanjem ***UEFI***-ja s **EFI** particija sustava, na diskovima particioniranim s **MBR** vrstom particija. Takva shema pokretanja sustava obično se naziva **UEFI-MBR**.



Ukoliko vam računalo podržava samo i isključivo ***UEFI*** (ne i ***Legacy BIOS***) načina rada za instalaciju operativnog sustava, morate imati instalacijski DVD-ROM ili *bootabilni* USB disk s podrškom za *UEFI* način instalacije sustava!.

***UEFI*** pruža i svoju radnu ljusku (*Shell*), koja se može koristiti za pokretanje drugih *UEFI* aplikacija, uključujući *UEFI boot loadere*, moguće je iz nje i particionirati disk ili dobiti razne druge informacija o sustavu ili *firmware*-u.

Za podršku za **UEFI** u Linux kernelu, kernel mora biti kompiliran s nekoliko opcija od kojih su važne navedene dvije:

```
grep -i EFI /boot/config-`uname -r`

CONFIG_EFI_PARTITION=y
CONFIG_EFI=y
```

**Izvori informacija: (1000),(1091),(1092),(1093),(1094),(1095),(1096),(1097), `man 7 bootup`, `man 7 bootparam`.**

## 12.1.2. Boot proces detaljnije

Do sada smo naučili da pokretanje Linux sustava uključuje različite komponente i odrađuje se u nekoliko koraka.
Nakon uključivanja računala, **BIOS** ili **UEFI** inicijalizira zaslon i tipkovnicu te testira glavnu (RAM) memoriju. Do ove faze računalo još uvijek ne pristupa nijednom mediju za pohranu odnosno: tvrdom disku (ATA/SATA/NVMe), USB disku, DVD i slično. Naknadno se iz CMOS ROM memorije u BIOS-u učitavaju informacije o trenutnom datumu, vremenu i najvažnijim periferijama. Kada se prepozna medij za pokretanje i njegova geometrija, kontrola sustava prelazi iz BIOS-a/UEFI-ja u fazu učitavanja takozvanog *boot loadera* sa za to označenog diska.

**Boot loader**
Vezano za proceduru pokretanja računala, kod računala s tradicionalnim BIOS-om moguće je učitati kôd odnosno posebni program koji se nalazi zapisan samo i isključivo u prvom fizičkom sektoru diska, koji je standardno veličine 512 bajta.
Ovaj prvi sektor diska se naziva *Master Boot Record* (**MBR**). Kod sustava koji kao *boot loader* odnosno navedeni kôd koji se pohranjuje u prvi dio prvog sektora diska, koriste tzv. *GRUB 2*, događa se sljedeće.
Naime *GRUB2* se sastoji od nekoliko stupnjeva, a samo prvi stupanj *GRUB2 boot loadera* se može pohraniti na prvi sektor diska, zbog svoje veličine i zbog ograničene veličine sektora na 512 bajta. Njegova jedina namjena kao prve faze *boot lodera* jest definicija sljedeće točke s koje se može nastaviti učitavanje sljedećeg *GRUB2* faze odnosno stupnja.



(Ova) prva faza *GRUB2 boot loadera* pohranjena na datotečnom sustavu vidljiva je kao datoteka imena: `boot.img`, a može se pronaći i u direktoriju: `/boot/grub2/i386-pc/`. Ova datoteka je zapravo diskovna slika prve faze *GRUB2 boot loadera*, koja se standardno instalira u prvi dio **MBR** zapisa.

Prva faza *GRUB2 boot loadera* (najčešće) učitava takozvanu fazu 1.5. *GRUB2 boot loadera*, iako je moguće i direktno učitavanje druge faze *GRUB2 boot loadera*.

**GRUB *boot loader* faza 1.5.**
**Vezano za fazu 1.5. Postoje razlike ovisno koristi li se MBR ili GPT shema particioniranja diska!**

**Za MBR shemu particioniranja diska vrijedi sljedeće**
U nulti sektor **MBR**-a se pohranjuje sadržaj datoteke (diskovne slike) `boot.img`, koji mora sadržavati i particijku tablicu.
Potom između **MBR** i **VBR** postoji još jedan mali prostor na disku na koji se instaliraju dvije diskovne slike.
One se prema zadanim postavkama upisuje u sektore između **MBR**-a i prve particije diska odnosno **VBR** zapisa prve particije. Naime zbog povijesnih (i kompatibilnih) razloga, prva particija tvrdog diska ne počinje u sektoru 1 jer brojanje počinje s **0**, već u sektoru 63, ostavljajući 63 sektora praznog prostora (63 x 512bajta) koji nisu dio bilo koje particije ili datotečnog sustava. Ovaj razmak od nešto manje od 32kB između **MBR**-a i prvog sektora datotečnog sustava (odnosno **VBR**-a) naziva se područje DOS kompatibilnosti ili MBR razmak, jer je DOS (operativni sustav iz 1983.g.) zahtijevao da particije započinju na granicama cilindra. Pošto svaki cilindar ima 64 sektora, to znači 64 sektora x 512 bajta pa stoga dolazimo do 32kB prostora.
Da budemo malo preciziniji, u sektore između **MBR** i **VBR** (sektori od 1 do 63), tijekom instalacije *GRUB2* sustava instaliraju se dvije diskovne slike. U sektor **1** (512 bajta) se ovisno o vrsti diska instalira sadržaj (samo jedne od) sljedećih diskovnih slika:

- `diskboot.img` - ako je disk tvrdi disk (mehanički) ili SSD/NVMe ili sl.
- `cdboot.img` - ako je medij CD-ROM.
- `lnxboot.img` - ako želimo (što nije zadana opcija) da se GRUB2 ponaša kao LILO boot loader. …

Nakon toga se počevši od drugog sektora do kraja ovog prostora (sektori 2 do 63) instalira sadržaj sljedeće diskovne slike:

- `core.img` - on je jezgra *GRUB2* sustava. Generira se tijekom pokretanja naredbe `grub2-mkimage` analizirajući konfiguracijske datoteke *GRUB*-a, posebno one u direktoriju: `/etc/grub.d/`, da bi se odredilo koji minimum *GRUB2* modula je potrebno uključiti u ovu diskovnu sliku. U konačnici `core.img` mora imati sve potrebne GRUB module kako bi mogao pristupiti direktoriju `/boot/grub2/` na formatiranoj particiji disk, koja se koristi u **fazi 2**. Ovakav modularni dizajn omogućuje da *GRUB2* diskovna slika bude dovoljno mala, budući da su područja diska gdje se mora instalirati maksimalne veličine 31.5KB. `Core.img` je obično u formatu *FAT* datotečnog sustava.

U konačnici `core.img` diskovna slika koja se instalira na disk sadrži jezgru *GRUB2* i osnovne module, uključujući `fat.mod`, `ext2.mod` ili `xfs.mod` i slične *GRUB2* module koji su u stanju dohvatiti sadržaj aktivne particije na kojoj se nalazi direktorij `/boot/`. Osim navedenih modula koji preposznaju format datotečnih sustava, potrebno je još nekoliko modula, koji mogu biti ili integralni dio `core.img` ili se mogu dohvatiti s `/boot` direktorija, ako je tako definirano u `grub.cfg` datoteci. Pogledajmo neke od češće korištenih *GRUB2* modula koji mogu biti integralni dio `core.img` ili se učitavaju s particije diska iz direktorija `/boot/grub2/i386-pc/`:

- `part_msdos.mod` - *GRUB* modul nužan za interpretiranje **MBR** zapisa i pristup particijama diska koje koriste **MBR** (*DOS*) shemu particioniranja. On priprema okruženje potrebno za pristup datotečnom sustavu pomoću modula poput `fat.mod`, `ext2.mod` i sličnih (ovisno o tome s kojim datotečnim sustavom je particija formatirana).
- `fat.mod`, `ext2.mod`, `xfs.mod` - *GRUB* moduli nužni za prepoznavanje datotečnog sustava na kojem treba doći do direktorija nužnog za nastavak učitavanja GRUB2 i kernela odnosno direktorija `/boot/`.



Navedene datoteke odnosno diskovne slike i *GRUB* moduli, nalaze se u direktoriju: `/usr/lib/grub/i386-pc/` ili u: `/boot/grub2/i386-pc/`. Sve ove datoteke dolaze u programskom paketu imena: `grub2-pc-modules`.

**Za GPT (GUID Partition Table) shemu particioniranja diska vrijedi sljedeće**
Prema ***GPT*** shemi diska između **MBR** i **VBR** ne postoji razmak. Ovdje je nakon **MBR** dijela diska dio s **GPT** zaglavljem, nakon kojega dolazi *GPT* dio s particijskim unosima i podacima o postojećim particijama na disku. Tek nakon njega dolazi prva particija, koja kod *GPT* sustava mora biti namjenjena za *boot loader*, a ne smije biti formatirana te ne smije biti veća od **1 MB**. Osim toga ova particija mora imati naziv "`BIOS_grub`"
U tu neformatiranu particiju se smješta (instalira) diskovna slika odnosno konkretno sadržaj dvaju datoteka redom: `diskboot.img` + `core.img`. Dakle na isti način kao kod **MBR** sheme particioniranja diska.

**Za MBR i GPT fazu 1.5. vrijedi sljedeće**
Ova faza 1.5 *GRUB2* ne montira `/boot` particiju u tradicionalnom smislu montiranja. Umjesto toga ona koristi posebne module za ograničeni pristup datotekama na toj particiji. Pri tome *GRUB* moduli poput `fat.mod` ili `ext2.mod` zaobilaze potrebu za punim montiranjem paricije jer poznaju format datotečnog sustava. Direktorij `/boot` nalazi se na datotečnom sustavu formatiranom s datotečnim sustavom koji on pomoću svojih modula zna koristiti, odnosno ovi *GRUB* moduli mogu direktno locirati i pročitati pojedine datoteke na particiji bez potrebe za punim razumijevanjem strukture datotečnog sustava.
Pogledajmo disk s jednom particijom, prema ***MBR*** shemi particioniranja diska, u tablici:

| **MBR shema diska s jednom particijom** | | | |
|---|---|---|---|
| **MBR (512 bajta)** | **Međuprostor (32kB)** *[zapravo 31.5kB]* | **Boot record prve particije (Volume Boot record) [VBR]** | **Prva particija (pr. s EXT4 datotečnim sustavom)** |
| Sadržaj datoteke: `boot.img`<br>***Booot kôd*** *+ opcionalno identifikator (446 bajta)*<br>+<br>***Particijska tablica*** *(64 bajta) [32 bita po particiji]*<br>***MBR potpis*** | Sadržaj datoteka:<br>`diskboot.img`<br>***Boot kôd***<br>+<br>`core.img`<br>*Zaglavlje*<br>*GRUB2 jezgra*<br>*GRUB2 moduli; pr.:* `fat.mod`,`ext2.mod`<br>*Dodatne datoteke* | *Boot kôd* ***(ne koristi ga GRUB2)***<br>*Naziv particije*<br>*Vrsta medija*<br>*Vrsta datotečnog sustava*<br>... | ***EXT 4 datotečni sustav***<br>*Stablo direktorija:*<br>`/`<br>`/boot/grub2/`<br>`...`<br>... |

Pogledajmo važnije elemente MBR particijske sheme diska:

**MBR zapis**
*Master Boot Record* (**MBR**) zapis je maksimalne veličine 512 bajta i sastoji se od sljedećih važnijih cjelina:

- ***Master boot code*** (446 bajta) – ovaj odjeljak sadrži izvršni kod (*bootstrapping*) odgovoran za učitavanje sljedeće faze *boot loadera*, a potencijalno i druga polja poput identifikatora diska. Ovaj identifikator diska je broj koji se nasumično generira kada se prvi puta kreira i pohrani **MBR** zapis na disk.
- ***Odjeljak s particijskom tablicom diska*** (64 bajta) – sljedeća 64 bajta rezervirana su za particijsku tablicu, koja može sadržavati opis do četiri primarne particije ili tri primarne particije i jednu proširenu particiju. Zapis za svaku particiju je maksimalne veličine 16 bajta i uključuje: status particije [*1 bajt*] (**0x80** ako je aktivna ili **0x00** ako nije naznačena kao aktivna ), vrstu particije [1 bajt] (primarna ili proširena), početni [*4 bajta*] i završni sektor [*4 bajta*] particije. *Zbog ovog ograničenja particija na 4 bajta (32 bita), maksimalna veličina MBR particija je 2TB.*
- ***MBR potpis*** (2 bajta) – posljednja dva bajta sadrže potpis, koji označava da je ovo ispravan MBR zapis.

**Međuprostor između MBR** i **VBR**:

- Veličine je 63 sektora i ovdje se uobičajeno pohranjuje sadržaj datoteka koje su diskovne slike, prvo iz datoteke `diskboot.img` te potom iz `core.img`. U ovaj međuprostor stane do maksimalno **31.5 kB** (***63x512bajta***) podataka.

**Volume Boot record (VBR) odnosno takozvani Boot record unutar particije**
*Volume Boot record* (**VBR**) odnosno *Boot record* (**BR**) je zapis koji se nalazi na početku svake particije.
On se kreira prilikom formatiranja particije, a nalazi se pohranjen u prvom sektoru particije (na prvom *cilindru* diska).
*Volume Boot record* se sastoji od sljedećih logičkih cjelina:

- ***Boot code*** - ovaj odjeljak sadrži izvršni kod (*bootstrapping*) odgovoran za učitavanje operativnog sustava. *GRUB2* ne koristi ovaj dio za zapis *boot loadera*, ali ga mogu koristiti drugi *boot loaderi*.
- ***Naziv particije*** - ovdje je zapisan naziv particije.
- ***Vrsta datotečnog sustava*** s kojim je particija formatirana (pr. *NTFS, XFS, EXT4*, …).
- ***Veličina sektora*** - ovdje je definirana veličina sektora.
- ***Broj sektora po klasteru***.

- ***Medij*** - definicija medija za pohranu. Primjerice:
  - `0xF8` - tvrdi disk (engl. Hard Disk Drive, HDD): ova vrijednost označava da se particija nalazi na tvrdom disku, što je najčešći slučaj za većinu particija na osobnim računalima.
  - `0xF0` - floppy disk.
  - `0xFE` - CD-ROM.
  - `0xFF` - flash memorija: Ova vrijednost se često koristi za particije koje se nalaze na USB uređajima, memorijskim karticama, SSD-ovima i drugim vrstama flash memorije.
  - `0x80` - mrežni disk.
  - `0x81` - RAM disk.
  - `0x82` - CD-RW.
  - `0x83` – DVD, …
- ***Broj rezerviranih sektora***- Ovaj broj označava koliko sektora na početku particije je rezervirano za posebne svrhe, obično za metapodatke datotečnog sustava. Na primjer, u **NTFS** datotečnom sustavu, prvi sektor obično je rezerviran za **MFT** (*Master File Table*), dok FAT32 sustav datoteka rezervira nekoliko sektora za boot sektor i **FAT** tablice.
- ***Maksimalan broj sektora na particiji***.
- ***Broj tablica datoteka*** (tzv. *File Allocation Table*) - Ovaj broj označava koliko tablica datoteka je prisutno na particiji. Na primjer, u *File Allocation Table* (**FAT**) sustavu datoteka, particija može imati jednu ili dvije tablice, ovisno o verziji FAT-a i veličini particije. U NTFS sustavima datoteka, obično postoji samo jedna tablica datoteka. Ova tablica sadrži metapodatke o svim datotekama i direktorijima na particiji. Međutim za primjerice **XFS** i **EXT4** datotečne susstave, koncept "*broja tablica*" nije relevantan na isti način kao što je to za **FAT** ili **NTFS** sustave datoteka. Naime *EXT4* koristi indekse kako bi ubrzao pristup datotekama i direktorijima, a ne tradicionalne tablice datoteka (engl. *File Allocation Table*) kao što to rade neki drugi datotečni sustavi poput FAT-a ili NTFS-a. Dok **XFS** koristi B+ stablo (B+ tree) za organizaciju metapodataka, poput *inode* tablica, i također ne koristi klasične tablice datoteka kao što ih koriste **FAT** ili **NTFS**.

## Vratimo se na drugu fazu GRUB2

**GRUB2 faza 2**
Pošto je faza **1.5** *GRUB2* pomoću potrebnih *GRUB* modula dohvatila sadržaj direktorija `/boot` s formatiranog datotečnog sustava, s te lokacije, unutar direktorija: `/boot/grub2/i386-pc/` učitavaju se dodatni *GRUB2* moduli poput:

- `normal.mod` – ovaj *GRUB2* modul pruža osnovne funkcije za korisničko sučelje GRUB-a. Omogućuje prikaz izbornika operativnih sustava, kao i izvršavanje naredbi unutar *GRUB* ljuske. `Normal.mod` je važan za korisničko sučelje i omogućuje korisnicima da odaberu koji operativni sustav žele pokrenuti ili da izvrše određene akcije unutar *GRUB-a*.
- `linux.mod` – ovaj *GRUB2* modul omogućuje *GRUB*-u interakciju s Linux kernelom i Linux datotečnim sustavima. Osim toga on omogućuje učitavanje Linux kernela iz datotečnog sustava, postavljanje parametara kernela i inicijalizaciju pokretanja ostatka Linuxa. Ovaj modul je stoga ključan za pokretanje Linux operativnih sustava s *GRUB*-om.

Nakon što je *GRUB2* u ovoj drugoj fazi s diska učitao *GRUB2* modul `linux.mod`, on pomoću njega traži linux kernel (`vmlinuz-<inačica>` datoteku u direktoriju `/boot/`). Linux kernel se dekomprimira i učitava u memoriju.
Kernel se inicijalizira i postavlja virtualni memorijski prostor. Kernel zatim preuzima kontrolu nad hardverom od *GRUB-a*.
Nakon učitavanja i inicijalizacije kernela, samo ako je ***initramfs*** naveden u `grub.cfg` (što je uobičajeno), `linux.mod` tada pronalazi datoteku ***initramfs*** (`initramfs-<inačica>.img`). Navedena datoteka je diskovna slika *initramfs* sustava koja se učitava u privremeni RAM disk. Kernel montira *initramfs* kao korijenski datotečni sustav. Potom **Init** proces unutar *initramfs* pokreće učitavanja dodatnih kernel modula i priprema razne uređaje (hardver) potrebne za ostatak procesa pokretanja operativnog sustava.



Vezano za ***boot loader*** (konkretno ***GRUB2***), pogledajte i poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**
**11.1.1.5.1. Rad s GRUB-om prilikom pokretanja sustava (*rescue*).**

Za detalje o **MBR** zapis na disku, pogledajte poglavlje:
**13.4.1. Napredno: MBR Detaljnije.**

Što se tiče ***boot loadera*** i sheme diska, pogledajte i poglavlje:
**13.4. Logička shema diska.**

**Izvori informacija: (1615),(1616),(1617),(1618),(1619),(1620),(1621),(1622),(1623), `man grub2-mkimage`**

## 12.2. Koje vrste RAM memorija postoje i u čemu su razlike

### 12.2.1. SDRAM vrsta memorije

**SDRAM** (*Synchronous Dynamic Random Access Memory*) je **DRAM** (*Dynamic Random Access Memory*) vrsta memorije koja je sinkronizirana sa sistemskom sabirnicom. To znači da ova vrsta memorije, a to su praktično sve današnje vrste memorija koje možemo susresti kao radnu memoriju računala, čeka na signal takta (engl. *clock signal*) prije nego prihvati bilo koju kontrolnu naredbu. Svaka naredba prema memoriji se odrađuje unutar određenog takta, kroz takozvani cjevovod (engl. *Pipeline*) koji omogućava prihvaćanje većeg broja naredbi istovremeno. Upotreba ovog cjevovoda omogućava prihvaćanje novih naredbi, prije nego li su prethodne izvršene tj. čekaju na izvršavanje. Obično postoje dva cjevovoda: jedan za čitanje, a drugi za zapisivanje. Kod zapisivanja podataka, naredbe za zapisivanje se ubacuju u cjevovod za zapisivanje, nakon prve naredbe može slijediti i druga u nîzu, bez čekanja na dovršetak prve. Isto se događa i kod čitanja. Na konačan rezultat čitanja ili zapisivanja, tj. na dohvat podataka iz memorije se zbog toga čeka određeni broj taktova; dok memorija stvarno za izvrši traženu operaciju, a ovo kašnjenje se zove SDRAM latencija (engl. *Latency*) i ona je važan parametar performansi memorije.
Memorija je podijeljena u određeni broj takozvanih *banki* (engl. *Banks*) ili područja za zapisivanje ili čitanje, što omogućava čitanje ili pisanje u i iz više njih u isto vrijeme. Memorija je također podijeljena na stupce i polja za čitanje i pisanje.
Prve "modernije" vrste ovakve memorije su se jednostavno zvale SDR (*Single Data Rate*) SDRAM memorije, a njeni nasljednici se zovu DDR (*Double Data Rate*) memorije. Važno je znati i to da **(S)DRAM** vrste memorija zahtijevaju stalno osvježavanje memorije, kako bi se sačuvali podaci pohranjeni u njoj.

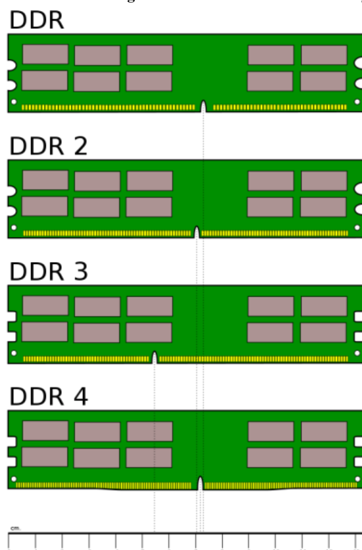
**Parametri rada RAM memorije**
Takozvano *Read cycle time* ili vrijeme odnosno ciklus čitanja je vrijeme između naredbe za čitanje određenog polja u memoriji i trenutka kada su ti podaci dohvaćeni. Kod SDRAM memorije koja radi na 100MHz to vrijeme je bilo 10ns, a kod DDR-400 memorije je to vrijeme smanjeno na 5ns, a toliko je ostalo i na svim novijim DDR memorijama. Drugi parametar je *Column Access Strobe* (***CAS***) *latency* koji označava vrijeme između slanja podatka o adresi polja i primanja stvarnih podataka. Brzina za gotovo sve vrste memorija je između 10.ns. i 15.ns. sve od starih SDR SDRAM do svih novih DDR (SDRAM) memorija. Jedino što se ovdje mijenjalo je broj ciklusa koji je potreban kako bi se odradila zadana operacija. S obzirom na to kako su vremena pristupa između 10.ns. i 15.ns., broj ciklusa na nižim taktovima je manji, dok je na većim taktovima veći: 10.ns. - 15.ns. na DDR-400 (radni takt 200MHz) je CL2-3 (2-3 takta) dok je na DDR2-800 to CL4-6 tj. na DDR3-1600 je to CL8-12 taktova. Prve SDRAM memorije su se zvale ***SDR*** (*Single Data Rate*) SDRAM memorije što znači da su omogućavale slanje jedne naredbe tj. čitanja ili pisanja samo jednog podatka u jednom ciklusu (engl. *clock rate*). Memorijski integrirani sklopovi (*čipovi*) su radili uobičajeno na 100MHz ili 133MHz i imali su širinu sabirnice 4, 8 ili 16 bita. Pošto se na svakom memorijskom modulu nalazi veći broj memorijskih čipova, memorijski moduli su bili 64 bitni (za ***non-ECC***) ili 72 bitni (za ***ECC*** podvrstu memorije). Dakle omogućavali su čitanje ili pisanje 64 odnosno 72 bita po pojedinom taktu.

**DDR RAM**
Nasljednik **SDR** je **DDR** RAM (engl. *Double Data Rate*) vrsta **SDRAM** memorije: tj. DDR1 je naslijedila DDR2, potom su slijedile DDR3, DDR4 i DDR5 memorije. Iako imaju isto ime ***DDR***, one nisu kompatibilne jedna s drugom i to već od fizičkih dimenzija. One su u odnosu na starije ***SDR*** memorije omogućile veće frekvencije rada, kao i prijenos dvostruko više podataka u svakom taktu. Odatle i naziv "*Double Data Rate*". Pogledajmo koji je izračun brzine prijenosa podataka, ovih memorija:

$$\boldsymbol{Propusnost\ memorije\ MB/s} = \frac{\boldsymbol{Radni\ takt\ memorije\ (MHz)\ x\ 64\ (broj\ bitova\ koji\ se\ prenose)}}{\mathbf{8}}$$

Za SDR RAM = 100 MHz x 64 bita / 8 = 800 MB/s, a za DDR RAM istog takta = 100 MHz x 64 bita **x2** (jer po taktu prenosi dvostruko više podataka) = 1600 MB/s. Pogledajmo i fizički izgled raznih vrsta DDR (SDRAM) memorija (slika 89):

**Slika 89. Fizički izgled raznih vrsta SDRAM memorija.**




Osim **(S)DRAM** vrste memorije, postoji i RAM memorija za posebnu namjenu, a to je takozvana **statička** memorija sa slučajnim pristupom, takozvana **SRAM** memorija. Pojam statički razlikuje **SRAM** od **DRAM**-a po tome što se **SRAM** ne mora stalno osvježavati, kako bi joj se sačuvao sadržaj. **SRAM** je brži i skuplji od **DRAM**-a: obično se koristi za predmemoriju (pr. **L1/L2/L3** CPU *cache*) i interne registre CPU-a, dok se **DRAM** memorija (**SDRAM**) koristi za glavnu memoriju računala.

Naime kod **(S)DRAM** memorija, osvježavanje memorije postupak je povremenog, ali konstantnog čitanja podataka iz memorije i trenutnog prepisivanja pročitanih podataka na isto područje memorije, bez izmjena, u svrhu očuvanja podataka.
Ovo osvježavanje memorije je postupak održavanja podataka unutar memorije, koji se konstantno odvija u pozadini, a potreban je tijekom rada, za sve vrste dinamičke memorije sa slučajnim pristupom (**DRAM**) i zapravo je jedna od važnih karakteristika ove kategorije odnosno vrste memorije.

Slika je preuzeta s Wikipedije (public domain licenca)

**Izvori informacija: (991),(992) ,(K-4), (K-9)**

### 12.2.1.1. RAM memorije koje se danas najčešće koriste

Pogledajmo tablicu s karakteristikama rada i nazivima **SDR** i **DDR** RAM memorija.

| **Vrsta RAM Memorije** *[uz naziv modula]* | **Takt (frekvencija) rada memorijske sabirnice** | **Brzina prijenosa podataka** | **Brzina prijenosa u MT/s (Mega Transfera u sek.)** |
|---|---|---|---|
| **DDR 5 – 5000** *[PC5-40.000]* | **2.500 MHz (2,5 GHz)** | **40 GB/s** | **5.000** |
| **DDR 4 – 3200** *[PC4-25.600]* | **1.600 MHz (1,6 GHz)** | **25,6 GB/s** | **3.200** |
| **DDR 4 – 2400** *[PC4-19.200]* | **1.200 MHz (1,2 GHz)** | **19,2 GB/s** | **2.400** |
| **DDR 4 - 2133** *[PC4-17.000]* | **1.067 MHz (1 GHz)** | **17 GB/s** | **2.133** |
| **DDR 3 - 1866** *[PC3-14.900]* | **933 MHz** | **14,9 GB/s** | **1.866** |
| **DDR 2 – 533** *[PC-4.200]* | **266 MHz** | **4,2 GB/s** | **533** |
| **DDR (1) – 400** *[PC-3.200]* | **200 MHz** | **3,2 GB/s** | **400** |
| **SDR** *[PC-100]* | **100 MHz** | **800 MB/s** | **100** |



Kod **DDR** memorija, primjerice oznaka **DDR 4 - 2400** ne označava stvarnu frekvenciju od 2.400 MHz, već je njen takt (frekv. memorijske sabirnice) na 1.200 MHz. Pošto je ovo „*Double Data rate*" vrsta memorije koja ima dvostruko veći prijenos podataka od **SDR** memorije, ona se označava kao da radi na dvostruko većem taktu, to jest na 2.400 MHz!

**U novije vrijeme, zbog po mnogima zbunjujućih oznaka RAM memorije, sve više se koristi jedinstvena oznaka za mjerenje efektivne brzine prijenosa podataka DDR2 memorija, kao MT/s odnosno Mega Transfera u sekundi!**

**Izvor informacija: (1125),(1678),(1679)**

### 12.2.1.2. Dual in-line Memory Module (DIMM)

*Dual in-line Memory Module* (**DIMM**) vrsta memorija zapravo označava fizički dizajn memorije, koja je naslijedila stariji oblik **SIMM** (*Single In-line Memory Module*) vrste memorija. Zapravo se radi o tome kako su stare SIMM memorije imale kontakte s obje strane memorijskog modula (štampane pločice), ali su kontakti s druge strane bili zapravo redundantni (zalihost) onima s prve strane. Osim toga SIMM memorije su bile 32 bitne. Kod DIMM memorija to nije slučaj, sve su 64 bitne te su kontakti s jedne strane pločice zaduženi za spajanje memorijskih čipova s jedne strane, a drugi za spajanje memorijskih čipova s druge strane. Fizičke vrste **DIMM** memorija su vidljive u tablici:

| **Broj pinova** (*nožica*) | **Namjena SDRAM memorije** |
|---|---|
| 72 | **SO-DIMM** (manji format) - za FPM DRAM i EDO DRAM |
| 100 | Za pisače (*printere*) i plotere. |
| 144 | **SO-DIMM** SDR SDRAM (manji format - za laptope i sl.) |
| 168 | **SDR SDRAM** |
| 172 | **Micro DIMM** - DDR SDRAM |
| 184 | **DDR SDRAM** |
| 200 | **SO-DIMM** (manji format - za laptope i sl.) - DDR SDRAM i DDR2 SDRAM |
| 204 | **SO-DIMM** (manji format - za laptope i sl.) - DDR3 SDRAM |
| 214 | **SO-DIMM** (manji format - za laptope i sl.) - DDR2 SDRAM |
| 240 | **Micro DIMM** - DDR2 SDRAM, DDR3 SDRAM, i FB-DIMM DRAM |
| 244 | **Mini DIMM** - DDR2 SDRAM |
| 260 | **Mini DIMM** [SO-DIMM (manji format - za laptope i sl.)] - DDR4 SDRAM |
| 262 | **DDR5** SDRAM **SO-DIMM** |
| 288 | **DDR4** SDRAM |

## 12.2.2. Podvrste Memorija

***Slijedi napredno poglavlje (12.2.2.x)!.*** Postoji i nekoliko podvrsta memorija, koje se koriste uglavnom za poslužitelje ili posebne namjene. Pogledajmo koje su to podvrste memorija.

### 12.2.2.1. ECC Memorija

**ECC** (engl. *Error Correcting Code*) vrsta memorije može detektirati i ispraviti greške u radu memorijskih čipova unutar memorijskog modula. Detekcija i ispravljanje grešaka se obično odnosi na pogrešku unutar jednog bîta podataka unutar jednog memorijskog modula i to na bilo kojem memorijskom čipu. Detekcija i korekcija jednog bîta se odnosi na mogućnost otkrivanja i ispravljanja greške na jednom bîtu na 64-bitnoj "riječi" (jedinica prijenosa podataka kroz memorijsku sabirnicu). Uglavnom novije ECC memorije i memorijski kontroleri mogu otkriti (ali ne i ispraviti) i greške na dva bîta po 64-bitnoj riječi. Najnovije inačice ECC memorija i memorijskih kontrolera koji ih podržavaju, razvijaju se s mogućnosti detekcije i korekcije dva pogrešna bîta na 64-bîtnu riječ. Za funkcionalnost ECC memorije, potrebna je i podrška od strane matične ploče odnosno memorijskog kontrolera, koji mora podržavati ECC memoriju. Dakle ECC memorije se ne mogu koristiti na matičnim pločama koje ih ne podržavaju (iako ih je fizički moguće ugraditi). To je zbog toga što funkcionalnost detekcije i korekcije greške odrađuje takozvani EDAC čip. Njegova funkcionalnost je danas integrirana na svim novijim procesorima (engl. CPU) koji podržavaju ECC memoriju. Fizički izgled ECC memorije pogledajte na slici **90.**
Postoji i još naprednija vrsta korekcije greške, patentirana od tvrtke *IBM* pod nazivom "Chipkill ECC", koja omogućava korekciju više bitova grešaka ili čak gubitak (kvar) cijelog memorijskog čipa na memorijskom modulu.
ECC vrsta memorije se ugrađuje i u "*Cache*" memorije unutar procesora. Ovakve cache memorije procesora, odnosno *Level* 1, 2 ili 3 memorije unutar CPU-a su prema vrsti, upravo ECC vrste memorije.

**Slika 90. Izgled SDR SDRAM PC-100 ECC memorije.**

Obratite pažnju na broj memorijskih čipova (**9**), za razliku od "obične" RAM memorije koji ih obično imaju **8** (ili neki drugi manji ali parni broj):




**U čemu je problem**
Električne ili magnetske smetnje unutar računala mogu uzrokovati greške unutar memorijskih čipova, tako da dolazi do pogrešaka unutar jednog bîta; točnije rečeno moguće je da jedan bît pređe u drugo stanje: 0 u 1 ili 1 u 0. Osim toga, moguće su i greške u sâmim čipovima, zbog pregrijavanja ili drugih faktora. Drugi najčešći faktor je pozadinsko zračenje (engl. *Background Radiation*) odnosno ionizirajuće zračenje kojem smo i sâmi izloženi, a koje dolazi iz raznih umjetnih ili prirodnih izvora. Postoji bojazan da će s obzirom na činjenicu da sve komponente pa i memorijski čipovi postaju sve manji, te rade na sve manjim naponima, učinak pozadinskog zračenja postati sve veći. Ovu teoriju podupire činjenica kako će neželjene čestice manjih energija moći neželjeno mijenjati stanje memorijskih "stanica" (engl. *Memory Cell*) u koje se pohranjuju podaci.
Studija koju je provodio **Google** od 2007 do 2009 u koju je uključio vrlo veliki broj poslužitelja, a čiji su rezultati prikazani na konferenciji "SIGMETRICS/Performance" 2009. godine pokazala je da su ove greške mnogo češće nego su na to ukazivali laboratorijski rezultati ili teorija.

Skraćeno, rezultat je: **Više od 8% DIMM memorijskih modula na godinu, bilo je pogođen ovim greškama.**

Posljedice ovakvih grešaka (uglavnom grešaka na jednom bîtu memorije), na sustavima koji nemaju ECC memorije mogu dovesti do rušenja aplikacije ili operativnog sustava ili korupcije podataka.
Na velikim sustavima ovakve greške su jedan od češćih uzroka rušenja sustava.

**Linux**
Iako možda ne zvuči nužno ili se o tome baš i ne razmišlja, ali osim toga što nam ECC memorija u pozadini detektira i rješava pogreške, često želimo biti i svjesni toga. Dakle možda neka od ECC memorija ugrađenih u naš poslužitelj konstantno ima nekih problema koji se stalno i detektiraju, ali i ispravljaju, a da toga nismo svjesni. Naravno takvu memoriju bi trebalo što prije otkriti i zamijeniti. Zbog toga je razvijena funkcionalnost koja omogućava Linuxu pristup memorijskom kontroleru, za koji smo rekli kako je u novije vrijeme ugrađen unutar CPU-a i to njegovom **EDAC** dijelu, koji je zadužen za detekcije i korekcije grešaka na ECC memoriji. Ova funkcionalnost je potpuno dostupna na svim kernelima 2.6.32 na dalje, te može biti već kompilirana u kernel ili dostupna kao kernel modul. Ona se sastoji od dvije komponente:

- **EDAC** jezgre (*Core*) koju čini `dac_core` kernel modul komponenta.
- Sâmog **EDAC** upravljačkog programa (*drivera*), koji je specifičan za vaš memorijski kontroler/CPU tj. ***EDAC*** (*chip*). Dakle ovaj upravljački program je specifičan za vaš CPU, pa za to mogu biti zaduženi: `sb_edac`, `e752x_edac`, `i7core_edac`, `i7300_edac` ili neki drugi kernel moduli.

Za dodatne informacije potražite pojam EDAC na poveznici: https://www.kernel.org/doc/

**Primjeri:**
Ako primjerice imamo ***Intel Sandy Bridge*** ili ***Ivy Bridge*** procesore; odnosno govorimo o njihovim memorijskim kontrolerima, tada je potrebno imati dva kernel modula:

- `edac_core` koji je centralna EDAC komponenta.
- `sb_edac` koji je kernel modul za naš CPU/memorijski kontroler.

Provjerimo jesu li učitani navedeni kernel moduli, pomoću naredbe `lsmod`:

```
lsmod | grep edac

sb_edac                 16606  0
edac_core               47267  3 sb_edac
```

Vidljivo je kako `edac_core` koristi `sb_edac` kernel modul, što je ispravno.
Svi podaci i statistike EDAC-a se sada nalaze unutar strukture direktorija: `/sys/devices/system/edac/mc/` .
Pri tome `mc` u imenu predstavlja memorijski kontroler odnosno „*Memory Controller*".

Prvo pogledajmo koje sve memorijske utore imamo na sustavu, pomoću naredbe `dmidecode` na sljedeći način:

```
dmidecode -t memory | grep "Locator: DIMM"

        Locator: DIMM_A1
        Locator: DIMM_A2
        Locator: DIMM_A3
        Locator: DIMM_A4
        Locator: DIMM_A5
        Locator: DIMM_A6
        Locator: DIMM_A7
        Locator: DIMM_A8
        Locator: DIMM_B1
        Locator: DIMM_B2
        Locator: DIMM_B3
        Locator: DIMM_B4
        Locator: DIMM_B5
        Locator: DIMM_B6
        Locator: DIMM_B7
        Locator: DIMM_B8
```

Ako gore navedena naredba ne daje rezultat, možemo probati i sa sljedećim nîzom naredbi:

```
dmidecode -t memory | egrep "Locator: A|Locator: B"
```

Dakle ovdje imamo 12 utora za prvi CPU i 12 utora za drugi CPU. Ovo je poslužitelj s dva fizička procesora (*NUMA CPU arhitektura*) i ukupno 24 memorijska utora. Memorijski kontroler na svakom procesoru ima četiri memorijska kanala. Dakle ukupno ih imamo osam.

Sada možemo provjeriti stanje svih memorijskih kanala odnosno ima li unutar nekog od memorijskih kanala grešaka na RAM memorijama:

```
grep "[0-9]" /sys/devices/system/edac/mc/mc*/csrow*/ch*_ce_count

/sys/devices/system/edac/mc/mc0/csrow0/ch0_ce_count:0
/sys/devices/system/edac/mc/mc0/csrow1/ch0_ce_count:0
/sys/devices/system/edac/mc/mc0/csrow2/ch0_ce_count:0
/sys/devices/system/edac/mc/mc0/csrow3/ch0_ce_count:0

/sys/devices/system/edac/mc/mc1/csrow0/ch0_ce_count:0
/sys/devices/system/edac/mc/mc1/csrow1/ch0_ce_count:0
/sys/devices/system/edac/mc/mc1/csrow2/ch0_ce_count:0
/sys/devices/system/edac/mc/mc1/csrow3/ch0_ce_count:0
```

Ispis smo razlomili na dva dijela (kako bi se vidjelo što pripada prvom procesoru, a što drugome).
Preciznije rečeno; sve što se nalazi unutar `/sys/devices/system/edac/mc/mc0/` pripada prvom memorijskom kontroleru (`mc` označava *memorijski kontroler*), a sve što je unutar `/sys/devices/system/edac/mc/mc1/` pripada drugom. S obzirom kako se memorijski kontroleri nalaze unutar sâmog procesora (CPU) i prijašnja izjava je točna.

U našem primjeru sve je u redu, pošto su sva stanja `0`, nema grešaka na ECC memorijama unutar memorijskih kontrolera.

Pogledajmo koje se još statistike (datoteke) nalaze unutar svakog pôddirektorija unutar vršnog direktorija: `/sys/devices/system/edac/mc/mc*/`:

- `ce_count` – ovdje se zapisuje ukupan broj grešaka koje su se dogodile unutra memorijskog modula. Ova datoteka se nalazi u poddirektoriju koji označava konkretan memorijski modul; pr. pôddirektorij: `dimm0/`, `dimm3/`, `dimm6/`, ….
- `ch0_ce_count` – ovdje se zapisuje ukupan broj grešaka koje su se uspjele ispraviti u ovom DIMM kanalu.
- `ch0_dimm_label` – ovo je kontrolna datoteka koja označava točni memorijski modul, a nalazi se u pôddirektoriju koji označava taj memorijski modul; pr. poddirektorij: `dimm0/`, `dimm3/`, `dimm6/`, … .

- `dev_type` ili `dimm_mem_type` – ovo je datoteka koja označava točni memorijski modul i njegovu vrstu, a nalazi se isto u pôddirektoriju koji označava taj memorijski modul; pr. pôddirektorij: `dimm0/`, `dimm3/`, `dimm6/`, … .
  Vrsta memorije može biti primjerice: *Registered-DDR3*.
- `edac_mode` ili `dimm_edac_type` – opisuje vrstu korekcije greške koja se primjenjuje.
- `size_mb` ili `size` – datoteka u kojoj je vidljiva veličina memorijskog modula, ako se nalazi u pôddirektoriju koji označava taj memorijski modul; pr. poddirektorij: `dimm0/`, `dimm3/`, `dimm6/`, … .
- `ue_count` – datoteka u koju se zapisuje broj nepopravljivih grešaka u ECC memoriji.

Postoji i koristan alat, koji je ponekada potrebno instalirati, a dolazi u paketu: `edac-utils`, koji ako nije instaliran, možemo instalirati sa sljedećom naredbom:

```
yum -y install edac-utils
```

S programom `edac-util` na jednostavniji način možemo pronaći koji memorijski modul ima greške. Pokrenimo ga:

```
edac-util -v

mc0: 0 Uncorrected Errors with no DIMM info
mc0: 0 Corrected Errors with no DIMM info
mc1: 0 Uncorrected Errors with no DIMM info
mc1: 0 Corrected Errors with no DIMM info
edac-util: No errors to report.
```

Ovdje vidimo kako niti na jednom memorijskom kontroleru (`mc`) i to niti na jednoj memoriji odnosno memorijskom modulu nema nikakvih grešaka (`No errors to report`).

**Izvori informacija: (308),(311),(993),(1125),(K-9), `man edac-util`, `man dmidecode`, `man yum`, `man lsmod`, `man 5 sysfs`.**

## 12.2.2.2. *Registered* memorija (RDIMM)

*Registered* memorija ili također poznata kao ***Buffered Memory*** ima poseban elektronički sklop zvan *Register*, na sâmom memorijskom modulu, a koji se logički nalazi između memorijskih čipova na modulu i memorijskog kontrolera na matičnoj ploči. Ovaj čip se koristi za stabilizaciju električnog opterećenja prema memorijskom kontroleru te omogućava ugradnju veće količine memorije na matičnu ploču. On je također zadužen i za spremanje kontrolnih linja u međuspremnik (Engl. *Buffer*) prema sâmoj memoriji. *Registered* memorije mogu biti obične, ali i *ECC* memorije. *Registered* memorije neće raditi na matičnim pločama koje ih ne podržavaju.
Što se naziva tiče, običnu memoriju u odnosu na ovu, nazivamo i *Unbuffered memory* ili *Unregistered memory* dok se *Registered* memorija naziva i ***RDIMM*** memorija (vidljiva na slici 91).

**Slika 91. 4GB 240 pinske DDR2-3200 *ECC Registered* memorije.**



## 12.2.2.3. Buffered Memory i Fully Buffered Memory

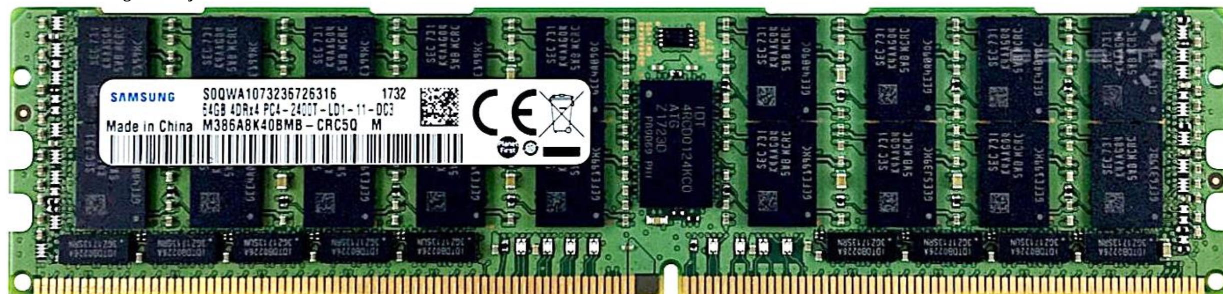
*Buffered Memory* je stari naziv za *Registered* memoriju dok je *Fully Buffered Memory* ili ***FB-DIMM*** koristi *buffer* (među memoriju) prema kontrolnim linijama memorije, ali i prema svim podatkovnim linijama. Stoga ove memorije imaju posebne dodatne čipove koji im dodaju ovu funkcionalnost. Dakle između memorijskih čipova na memorijskom modulu i memorijskog kontrolera na matičnoj ploči (ili u procesoru); a fizički na memorijskom modulu, ugrađen je tzv. **AMB** (engl. *Advanced Memory Buffer*) čip. Za razliku od normalne memorijske sabirnice koja je paralelna pa je i sâma memorija paralelna, kod *FB-DIMM* memorija, **AMB** serijalizira sav promet i dalje ih paralelizira prema memorijskim čipovima na sâmom memorijskom modulu.

Ova operacija pojednostavljuje pristup memoriji sa strane memorijskog kontrolera, ali unosi malo kašnjenje zbog dodatne obrade koju odrađuje *AMB*. Neki *AMB* čipovi sâmi mogu odrađivati ECC funkcionalnost odnosno korekciju greške.
Ova vrsta memorije kontrolira napajanje, slično kao *Registered* memorije.
***FB-DIMM*** radi samo na matičnim pločama koje podržavaju ovu vrstu memorije. ***FB-DIMM*** može biti i **ECC**.

### 12.2.2.4. Load Reduced DIMM (LRDIMM)

*Load Reduced DIMM* (**LRDIMM**) je slična kao *Registered* ili *FB-DIMM* s time da **LRDIMM** ima centralni sklop (čip) koji ima memorijski međuspremnik u koji sprema kontrolne, ali i podatkovne linije u paralelnom načinu rada. Naime za razliku od *FB-DIMM* novija **LRDIMM** memorija koristi tzv. *Isolation Memory Buffer* (**iMB**). **iMB** sprema signale naredbi, adresa i podataka. Dakle on izolira sve električne signale (uključujući podatkovne signale) memorijskih čipova na *LRDIMM* memorijskom modulu, od glavnog memorijskog kontrolera. Stoga centralni memorijski kontroler (na matičnoj ploči ili unutar CPU-a), vidi samo iMB, a ne pojedinačne memorijske čipove na memorijskom modulu. S ovime se zadržala velika brzina rada te mogućnost instalacije velike količine memorije na matičnu ploču uz minimalno kašnjenje zbog dodatne obrade kao što ga primjerice ima FB-DIMM s *AMB* čipom. Ova vrsta memorije također mora biti podržana na vašoj matičnoj ploči. Koristi se uglavnom za velike poslužitelje koji trebaju podržavati najveću moguću količinu RAM memorije. Fizički izgled *LRDIMM* vrste memorije je vidljiv na slici **92**.

**Slika 92. Samsung memorija LRDIMM DDR4-2400 64GB**




**Izvori informacija: (309),(310).**

### 12.2.2.5. Compression Attached Memory Module CAMM(2) i LPCAMM2

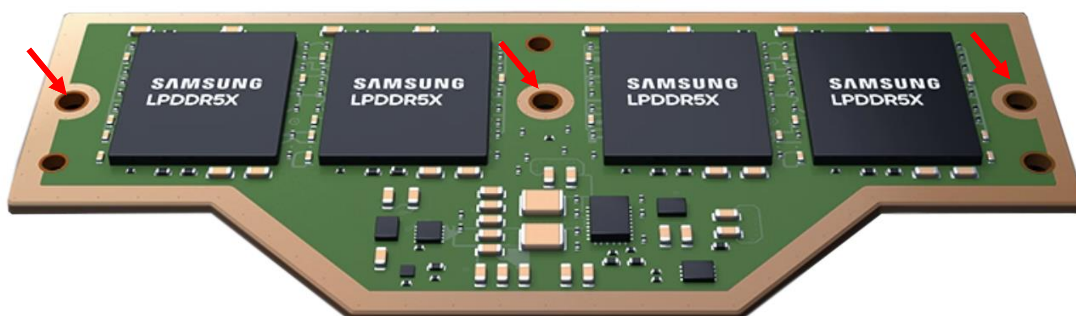
**LPDDR**

Na matičnim pločama prijenosnih računala, tableta i mobitela, zalemljena je i RAM memorija (čipovi), dakle nije ju moguće nadograditi. Ako se radi o uređajima koji traže prvenstveno malu potrošnju memorije, ali i visoke performanse, gotovo jedini odabir proizvođača je bio takozvana *Low-Power Double Data Rate* **(LPDDR)** memorija. Ovakva vrsta memorije se razvijala paralelno sa svim već navedenim **DDR** memorijama za računala i poslužitelje, pa je imala i svoje razne generacije: **LPDDR1**, **LPDDR2**, **LPDDR3**, **LPDDR4, LPDDR5** i **LPDDR5x**. Međutim ove generacije **LPDDR** memorija nisu direktno usporedive s već navedenim **DDR** generacijama memorija za stolna računala i poslužitelje. Primjerice **LPDDR5** je izašao gotovo godinu dana prije **DDR5** memorije. Što se tiče njene propusnosti ona je usporediva odnosno vrlo blizu **DDR** memorijama (prema **MT/s**). Međutim njena mana, zbog niskog napona na kojem radi jest potreba da se zalemi što bliže procesoru odnosno memorijskom kontroleru. Odnosno da vodovi koji ih povezuju budu što kraći, kako ne bi dolazilo do degradacije signala.

**CAMM2 i LPCAMM2**

Zbog potrebe za nadogradnjom memorije, razvijeni su *Compression Attached Memory Module* **(CAMM)** memorijski moduli te pripadajući memorijski utori za prihvat ovakve vrste memorije. ***CAMM***(2) memorijski moduli su i fizički potpuno drugačiji, pošto joj se svi konektori nalaze s donje strane odnosno koriste takozvani *land grid array* **(LGA)** način povezivanja, poput većine današnjih procesora tvrtki *AMD* i *Intel*. Specifičnost fizičkog **LGA** sučelja jest u tome da se na matičnoj ploči nalaze nožice, a na ***CAMM(2)*** ili ***LPCAMM2*** memoriji kontakti. Ovakva memorija se stoga "*uglavljuje*" na svoju poziciju te učvršćuje s vijcima. ***LPCAMM2*** memorijski modul i rupe za vijke na njemu (*označene sa strjelicama*) vidimo na slici (**92.A.**).

**Slika 92.A. LPCAMM2 memorijski modul tvrtke Samsung**




Pošto je **LPCAMM2** RAM memorija, kako joj i ime kaže "*Low Power*", i ona radi na niskom naponu, pa troši vrlo malo energije. Stoga je utor za njenu ugradnju također pozicioniran vrlo blizu procesora, kako ne bi dolazilo do degradacije signala. **LPCAMM2** memorija će sigurno sve više biti zamijenjena za ***SO-DIMM*** memorije na prijenosnim računalim. **LPCAMM2** memorijski moduli sve češće koriste **LPDDR5X** memorijske čipove, a podržavaju i dvokanalni rad. To znači da će jedan **LPCAMM2** memorijski modul mijenjati dva **SO-DIMM** modula, uz manju latenciju i veću propusnost (ako gledamo **MT/s**).

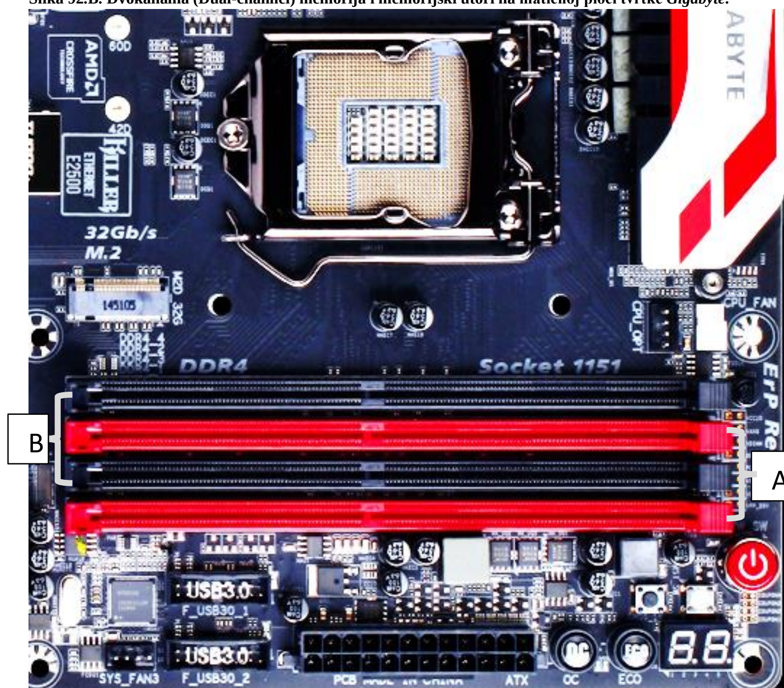
**Izvori informacija: (1673),(1674),(1675),(1676),(1677),(1678).**

## 12.2.2.6. Višekanalna arhitektura memorije

Višekanalna arhitektura memorije je tehnologija koja povećava brzinu prijenosa podataka između DRAM memorije i memorijskog kontrolera, dodavanjem više kanala komunikacije između njih. Teoretski ukupna brzina prijenosa podataka je tim veća što više komunikacijskih kanala se koristi. Dvokanalna memorija (Engl. *Dual channel*) koristi dva kanala, četvero kanalna memorija koristi četiri kanala i tako dalje. Moderni standardni procesori podržavaju dvokanalnu memoriju, dok vrhunski procesori za stolna računala i radne stanice kao što je serija *AMD Ryzen Threadripper* i linija *Intel Core i9 Extreme Edition* podržavaju četvero kanalnu memoriju. Međutim poslužiteljski procesori iz serije *AMD Epyc* i *Intel Xeon* imaju podršku počevši od četvero kanalne pa sve do osmero kanalne arhitekture. Dvokanalna arhitektura zahtijeva dvokanalnu matičnu ploču te upotrebu dva ili više DDR (DDR2, DDR3, DDR4 ili DDR5) memorijskih modula. Memorijski moduli se instaliraju (ugrađuju) u odgovarajuće utore, od kojih svaki pripada drugom memorijskom kanalu. Pri instalaciji odnosno ugradnji memorijskih modula potrebno je proučiti priručnik za konkretnu matičnu ploču. Ugradnja se obično radi tako da se usklađeni par memorijskih modula ugrađuje u utore svakog kanala, obično prvo popunjavajući prvi kanal, pa drugi i tako dalje.

Upotreba višekanalne memorije zahtjeva podršku od strane matične ploče, ali i centralnog procesora (CPU-a).

**Slika 92.B. Dvokanalna (Dual-channel) memorija i memorijski utori na matičnoj ploči tvrtke *Gigabyte*.**




Memorijski moduli koji rade na različitim brzinama mogu se koristiti u dvokanalnom načinu rada, iako će matična ploča tada postaviti sve memorijske module na brzinu najsporijeg modula. Međutim neke matične ploče imaju problema s kompatibilnošću s određenim markama ili modelima memorije kada ih pokušamo koristiti u dvokanalnom načinu rada. Iz tog razloga, općenito se savjetuje korištenje identičnih parova memorijskih modula, zbog čega većina proizvođača memorijske module prodaje kao "komplete" odnosno uparene memorijske module. Nadalje, nekoliko proizvođača matičnih ploča podržava samo konfiguracije u kojima se koristi "odgovarajući par" modula odnosno parovi modula.
Pri tome odgovarajući par modula mora biti iste brzine, kapaciteta, CAS latencije te broja čipova i strana.
Na slici 92.B. vidimo dio matične ploče tvrtke ***Gigabyte GA-Z170X***, s pogledom na memorijske utore u dvokanalnoj izvedbi.
Na ovoj matičnoj ploči memorijski utori svakog kanala su drugačije boje: kanal **A** (crveno) i kanal **B** (crno).

**Teoretska propusnost**

Kako smo već prethodno govorili, pogledajmo koji je izračun brzine prijenosa podataka, DRAM memorija:

$$\textbf{Propusnost memorije}\ (MB/s) = \frac{\textbf{Radni takt memorije}\ (MHz)\ x\ 64\ (\textbf{broj bitova koje se prenose})}{8}$$

Za **SDR** RAM memoriju koja radi na 100MHz je to: $\frac{100\ x\ 64}{8} =$ **800 MB/s**. Dakle imamo propusnost od **800MB/s**.
Međutim, pošto sve **DDR** vrste memorija (DDR2/3/4/5) po svakom taktu prenose dvostruko više podataka, tada bi za isti takt **DDR** memorije od 100 MHz imali drugačiju računicu: $\frac{100\ x\ 64\ x\ 2}{8} =$ **1.600 MB/s**. Ovdje imamo propusnost od **1.600 MB/s.**

**Stoga za DDR vrste memorija vrijedi sljedeća formula:**

$$\textbf{Propusnost DDR memorije}\ (MB/s) = \frac{\textbf{Radni takt memorije}\ (MHz)\ x\ 64\ (\textbf{broj bitova koje se prenose})\ x\ 2}{8}$$

To znači da primjerice DDR4 memorija oznake DDR4-3200, koja zapravo radi na 1.600MHz, ima sljedeću propusnost: $\frac{1600\ x\ 64\ x\ 2}{8}$ = **25.600 MB/s** to jest **25,6 GB/s.**



Ne zaboravimo da kod ***DDR*** memorija, primjerice oznaka **DDR 4 - 2400** ne označava stvarnu frekvenciju od 2.400 MHz, već je njen takt (frekvencija memorijske sabirnice) na 1.200 MHz. Pošto je ovo „*Double Data rate*“ vrsta memorije koja ima dvostruko veći prijenos podataka od ***SDR*** memorije, ona se označava kao da radi na dvostruko većem taktu, to jest 2.400 (*MHz*)!. Za točne frekvencije rada nekih od memorija, pogledajte tablicu u poglavlju: **12.2.1.1.**
**Zbog ovih često zbunjujućih oznaka, u novije vrijeme se sve više koristi jedinstvena oznaka propusnosti DDR2 memorija kao MT/s odnosno Mega Transfera u sekundi !**

Pogledajmo i logičku shemu s razlikama u dizajnu jednokanalne i dvokanalne memorije, na slici 92.C.

**Slika 92.C. Jednokanalna (gornji dio slike) i Dvokanalna memorija (donji dio slike)**



Jednokanalna DDR4 memorija oznake DDR4-3200, koja kako smo prethodno izračunali, ima propusnost od **25,6GB/s** te 64 bitnu sabirnicu. Međutim prelaskom na dvokanalnu memoriju stanje se mijenja, pa sada imamo propusnost od 2 x 25,6GB/s odnosno **51,2GB/s** te ukupno **128** bitni komunikacijski kanal.
Međutim iako u slučaju dvokanalnog rada kombiniranjem dvije 64-bitne sabirnice u jednu 128-bitnu sabirnicu ima logike, ovakav rad se nije svugdje pokazao kao značajno bolji.

On se retrospektivno naziva "skupljeni" (engl. *Ganged*) način rada. Stoga određene implementacije dvokanalnog rada prema zadanim postavkama, koriste način rada koji održava dvije 64-bitne memorijske sabirnice, ali omogućuju neovisni pristup svakom kanalu, kao odličnu podršku višenitnosti, poglavito upotrebom višejezgrenih procesora. S time se još više dobiva na performansama. Još veća razlika je kod poslužitelja, koji imaju četvero ili osmerokanalnu arhitekturu. U praksi razlike kod upotrebe višekanalne memorije su očite i primjetne, ali i variraju, ponajviše ovisno o aplikacijama odnosno primjeni poslužitelja ili računala koje ih koristi.

Pri tome **MMU** komponenta memorijskog kontrolera pretvara virtualne adrese vidljive aplikacijama u fizičke adrese RAM memorije (o njoj kasnije u poglavlju o virtualnoj memoriji).

**Izvori informacija: (1123),(1124),(1125),(1678).**

## 12.2.3. 32. bitno ili 64. bitno adresiranje memorije

Stariji procesori (CPU) imali su sâmo 32 bîtnu arhitekturu te su samim time podržavali 32 bîtno adresiranje RAM memorije. Ovdje se radi o tome da je s 32 bîta odnosno $2^{32}$ moguće adresirati samo 4GB RAM memorije.
S druge strane procesori koji imaju 64 bîtnu arhitekturu, što znači da su im 64 bîtni: podatkovna i adresna sabirnica, registri, i drugo, te podržavaju adresiranje puno veće količine RAM memorije tj. $2^{64}$ što je mogućnost adresiranja do 16 **eksabajta** RAM memorije.
Međutim, u praksi su zbog raznih razloga ta ograničenja drugačija, te je u 64 bîtnom Linuxu moguće adresirati do 64TB RAM memorije. Važno je znati da osim sâmog procesora (CPU) i operativni sustav mora biti 64 bîtan, kao i svi njegovi pripadajući upravljački, ali i drugi programi koji su važni za funkcioniranje operativnog sustava kao cjeline.



Na 64 bîtnom operativnom sustavu, moguće je izvršavati i “stare” 32 bîtne programe, uglavnom bez ikakvih problema.

**S druge strane, 64 bîtne aplikacije NIJE moguće izvršavati na 32 bîtnom operativnom sustavu!**

Pogledajmo usporednu tablicu 32-bitnog vs. 64-bitnog adresiranje memorije:

| Karakteristika | 32-bitno | 64-bitno |
|---|---|---|
| **Adresni prostor** | Maksimalno 4 GB ($2^{32}$ adresa). | Teoretski do 16 EB ($2^{64}$ adresa), u praksi 48-52 bita. |
| **Širina registara CPU-a** | 32-bita. | 64-bita. |
| **Performanse** | Pogodni za osnovne zadatke i male skupove podataka. | Pogodni za aplikacije s velikim skupovima podataka i kompleksnim izračunima |
| **Podrška RAM memorije** | Ograničeno na 4 GB fizičkog RAM-a. | Može koristiti stotine terabajta RAM-a. |
| **Kompatibilnost OS-a** | Podržava samo 32-bitne operativne sustave. | Podržava i 32-bitne i 64-bitne operativne sustave. |
| **Kompatibilnost aplikacija** | Samo 32-bitne aplikacije | Podržava i 32-bitne i 64-bitne aplikacije. |
| **Učinkovitost** | Sporije rukovanje s velikim podacima (npr. 64-bitnim vrijednostima). | Brže i učinkovitije rukovanje velikim podacima. |
| **Potrošnja resursa** | Manja potrošnja memorije i resursa. | Veća potrošnja memorije (64-bitne aplikacije koriste više prostora za podatke i kôd). |
| **Primjena** | Stariji sustavi, ugrađeni uređaji, IoT. | Moderni sustavi, poslužitelji, napredne aplikacije |
| **Kompatibilnost/podrška softvera** | Sve manja podrška za softver. | Standard za moderne aplikacije i operativne sustave. |
| **Ograničenja** | Ograničen RAM, slabije performanse s velikim skupovima podataka. | Veća potrošnja resursa, ali bolji performanse. |

## 12.2.4. Testiranje RAM memorije

U slučajevima kada sumnjamo na greške u RAM memoriji, potrebno je nekako testirati ugrađenu RAM memoriju u računalu.
Međutim to sâmo po sebi nije posve jednostavno, jer je potrebno osigurati se da imamo provjereno stabilan i funkcionalan, ali i minimalan moguć operativni sustav. Naime operativni sustav s kojim testiramo ugrađenu RAM memoriju, mora koristiti minimalnu moguću količinu RAM memorije, kako bi sve preostale regije memorije mogli testirati. Stoga jer je gotovo nemoguće testirati memoriju koja je već u upotrebi od strane raznih programa i komponenti sustava. Srećom za tu namjenu imamo već izrađen mikro operativni sustav, koji se sastoji sâmo od kernela uz koji dolaze upravljački programi za mnoge procesore (CPU) te memorijske kontrolere preko kojih se i pristupa RAM memoriji, te sâmog programa za testiranje RAM memorije. Ovaj sustav se zove **Memtest86+** koji služi za testiranje memorije i testiranje otpornosti na stres na memoriji sa slučajnim pristupom (**RAM**), na x86 arhitekturi računala.

On radi zapisivanjem testnih uzoraka na dostupne memorijske adrese, te čitanjem prethodno zapisanih podataka i usporedbom eventualnih grešaka. On potom pokušava provjeriti hoće li RAM memorija prihvatiti i ispravno zadržati zapisane uzorke zapisanih podataka, da nema grešaka u interakciji različitih bitova memorije te da nema sukoba između memorijskih adresa.
Ovo je vrlo važno i stoga što se provjerom odnosno iščitavanjem prethodno zapisanih uzoraka (praktično nîza nula i jedinica), može otkriti greška čak i na pojedinom bîtu RAM memorije, što kod drugih programa/sustava za testiranje RAM memorije nije moguće.

Korištenje **Memtest86+** sustava na ***RedHat/CentOS*** Linuxu, moguće je na dva načina:
- Pokretanjem s instalacijskog DVD medija (odabirom: ***Troubleshooting → Run a memory test***).
- Instalacijom **Memtest86+** sustava.

Mi ćemo se odlučiti za instalaciju **Memtest86+** sustava. To podrazumijeva instalaciju ovog sustava, na sljedeći način:
```
yum -y install memtest86+
```

Potom je potrebno napraviti inicijalnu konfiguraciju, pokretanjem naredbe:
```
memtest-setup
```

Zatim je za ***RedHat/CentOS 7+*** potrebno kreirati novi unos u ***boot loaderu*** (**GRUB2**).
```
grub2-mkconfig -o /boot/grub2/grub.cfg
```



Vezano za ***boot loader*** (konkretno ***GRUB2***), pogledajte i poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**
**11.1.1.5.1. Rad s GRUB-om prilikom pokretanja sustava (*rescue*).**

Prethodna naredba će ažurirati ***GRUB2*** opcije pokretanja sustava, tako da uključe ***Memtest+*** program/sustav. Sljedeći put kada ponovno pokrenete svoje računalo, ***GRUB2*** će vam ponuditi kao opciju ***Memtest+*** unutar izbornika za podizanje sustava.

Nakon toga, potrebno je restartati računalo, jer će **Memtest86+** sustav biti instaliran i dostupan preko ***boot loadera*** sustava. Dakle nakon prvog restarta sustava, pojavit će nam se nova opcija (na kraju) za pokretanje (***boot***) računala, poput:

```
CentOS Linux (3.10.0-1127.el7.x86_64) 7 (Core)
CentOS Linux (3.10.0-1127.el7.x86_64) 7 (Core)
CentOS Linux Memtest memtest86+-5.01
```

Ako tijekom pokretanja računala odaberemo ovu novu opciju: **Memtest86+**, tada će se i pokrenuti ovaj sustav/program.
On će prvo prepoznati naš procesor (**CPU**) te memorijski kontroler i krenuti s testiranjem RAM memorije.

Sâmo testiranje RAM memorije kreće odmah, s popunjavanjem dijelova RAM memorije s određenim uzorkom [Engl. *Pattern*], poput primjerice: ***FFFFFFFF***, ***00000001***, … te iščitavanjem zapisanog uzorka, potom testom s novim uzorkom i tako sve dok se ne testira sva dostupna RAM memorija.

Kada **Memtest86+** završi s cijelom RAM memorijom, on nastavlja testiranje u novom prolazu [Engl. *Pass*] i tako konstantno, sve dok ga ne prekinemo s tipkom ***ESC***. Ovo ponavljanje je korisno za takozvani stres test računala odnosno njegovu izdržljivost na dugotrajan intenzivan rad s memorijom.

Na slici 92.1 pogledajmo kako izgleda pokrenut program/sustav **Memtest86+**:

**Slika 92.1.** Pokrenuti **Memtest86+** program/sustav:

```
      Memtest86+ 5.01        | Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
CLK: 1800 MHz  (X64 Mode)   | Pass 54% #####################
L1 Cache:   32K 359992 MB/s | Test 49% ###################
L2 Cache:  256K 119997 MB/s | Test #9  [Random number sequence]
L3 Cache: 6144K  66665 MB/s | Testing: 1024K - 1024M   1023M of 1024M
Memory  : 1024M  17646 MB/s | Pattern:   567d166b          | Time:   0:01:49
------------------------------------------------------------------------------
Core#: 0 (SMP: Disabled)              | RAM: 1550MHz (DDR3-3100) - BCLK: 100
State: - Running...                   | Timings: CAS 19-15-15-31 @ 192-bit Mode
Cores:  1 Active /  1 Total (Run: All) | Pass:       0         Errors:      0
------------------------------------------------------------------------------

                                     S  S
(ESC)exit  (c)configuration  (SP)scroll_lock  (CR)scroll_unlock
```

**Izvori informacija: (981),(982),** `man memtest-setup`.

# 12.3. Memorijski menadžment i virtualna memorija

Memorijski menadžment *(Engl. Memory management)* se odnosi na proces upravljanja memorijom na razini operativnog sustava. Njegova osnovna zadaća je osigurati mogućnost dinamičkog alociranja memorije, na zahtjev programa (procesa odnosno aplikacija), ali i osloboditi memoriju u trenutku kada više nije potrebna.
Ova zadaća je vrlo važna i to dodatno s obzirom na činjenicu kako su svi današnji Linuxi, kao i drugi operativni sustavi u kojima više programa/procesa/aplikacija radi u isto vrijeme, optimizirani za korištenje RAM memorije što je više moguće.

To znači da će Linux pokušati iskoristiti što je više moguće RAM memorije na inteligentan način; kako bi se ubrzale sve operacije koje se tako mogu ubrzati. Ovo ubrzanje se uglavnom odnosi na minimalnu upotrebu *swap*-a te na zapisivanje podataka na diskovni sustav tek kada je to stvarno nužno i neophodno. Memorijski menadžment koristi nekoliko metoda kako bi povećao efikasnost korištenja memorije. Sustav virtualne memorije je metoda koja se koristi za razdvajanje virtualnih memorijskih adresa koje koriste programi (procesi), od stvarnih (fizičkih) memorijskih adresa RAM memorije.

Ova metoda omogućava odvajanje procesa u smislu korištenja posebnog memorijskog adresnog prostora te efikasnijeg korištenja RAM memorije pomoću metode koja se zove ***Paging*** odnosno upotreba takozvanih stranica memorije. Osim toga memorijski menadžment može koristiti i takozvani ***Swapping*** na disk, kao proširenje RAM memorije prema potrebi, ali o njemu nešto kasnije. I na kraju, jasno je kako izvedba odnosno kvaliteta memorijskog menadžera ima znatan utjecaj na performanse cijelog sustava.

---

## 12.3.1. Što je virtualna memorija i kako radi

Korištenje virtualne memorije odvaja korištenje fizičke RAM memorije i od same aplikacije odnosno programa (procesa). Upotrebom sustava virtualne memorije, aplikacije koriste virtualni adresni prostor memorije, tako da kada aplikacija treba pristupiti memoriji, zahtjev za memorijskom adresom se preslikava (translatira) iz virtualnog adresnog prostora u fizički odnosno stvarni adresni prostor RAM memorije.

Drugim riječima virtualne adrese memorije se pretvaraju u stvarne (fizičke), od strane kernela, a skriveno aplikacijama. Rad s virtualnom memorijom znači kako vaš program nikada ne koristi stvarne memorijske adrese to jest adrese fizičke RAM memorije, niti ima pristup do njih.



Važno je znati kako se sâmo prevođenje (translacija) virtualnih adresa u fizičke adrese događa unutar memorijskog kontrolera procesora (CPU) i to unutar komponente koja se zove *Memory Management Unit* (**MMU**).

Memorijske adrese su zapravo pokazivači na memorijske lokacije na kojima se nalazi naš (pokrenuti) program ili njegovi dijelovi (pr. biblioteke i slično). Kada program treba nešto zapisati ili čitati u ili iz memorije, on mora provjeriti s koje memorijske adrese će čitati koji dio podataka. O točnom načinu čitanja i zapisivanja ćemo govoriti kasnije kada budemo pričali o *Pagingu*. Dakle upotrebom virtualne memorije, operativni sustav ima potpunu kontrolu nad fizičkom memorijom odnosno nad njenim pristupom. Samim time operativni sustav može ograničiti način na koji procesi (aplikacije) pristupaju memoriji.

Dodatno, operativni sustav na najnižoj razini razgraničuje memoriju na dva dijela: kernel prostor memorije za kernel i kernel programe (tzv. *kernel space*) te na korisnički prostor memorije za korisničke (i druge) programe na sustavu (tzv. *user space*). Pri tome kernel ima pristup svim programima i komponentama i unutar svog adresnog prostora (*kernel space)* i unutar korisničkog prostora memorije. Dok s druge strane programi unutar korisničkog adresnog prostora (*user space*) nemaju direktan pristup programima i komponentama unutar prostora kernela, već se za to koriste posebne (indirektne) metode pristupa.

Vezano za opću kontrolu pristupa memoriji, ona se zove *Memory protection* odnosno sustav zaštite memorije i ona ograničava pristup svakog procesa, na dio memorije koji je alociran sâmo za njega, te mu onemogućava ulazak u adresni prostor nekog drugog procesa (aplikacije). Implikacije ovoga su vrlo jasne: onemogućavanje malicioznih namjernih ili nenamjernih radnji.

Iako svaki proces (aplikacija) radi unutar svog ograničenog adresnog prostora u memoriji, često je potrebno dijeljenje određenog adresnog prostora memorije između više aplikacija. Primjerice potreban je pristup podacima koji se dijele između procesa (aplikacija).
Klasičan primjer su programske biblioteke (engl. *Library*), kojima treba pristupati veći broj procesa odnosno aplikacija istovremeno.



**Metoda pristupa dijeljenim podacima se zove metoda dijeljenja memorije (engl. *Shared Memory)*.**

Sustav dijeljenja memorije (engl. *Shared Memory)* je ujedno i najbrža metoda za komunikaciju između procesa odnosno aplikacija. To se postiže klasifikacijom pristupa memoriji na tkzv. *Primary Storage* i *Secondary Storage* odnosno primarni i sekundarni sustav memorije. Jedan od zadataka *memorijskog menadžera* je i baratanje s ove dvije vrste pristupa memoriji, na razini svakog pokrenutog programa (procesa).

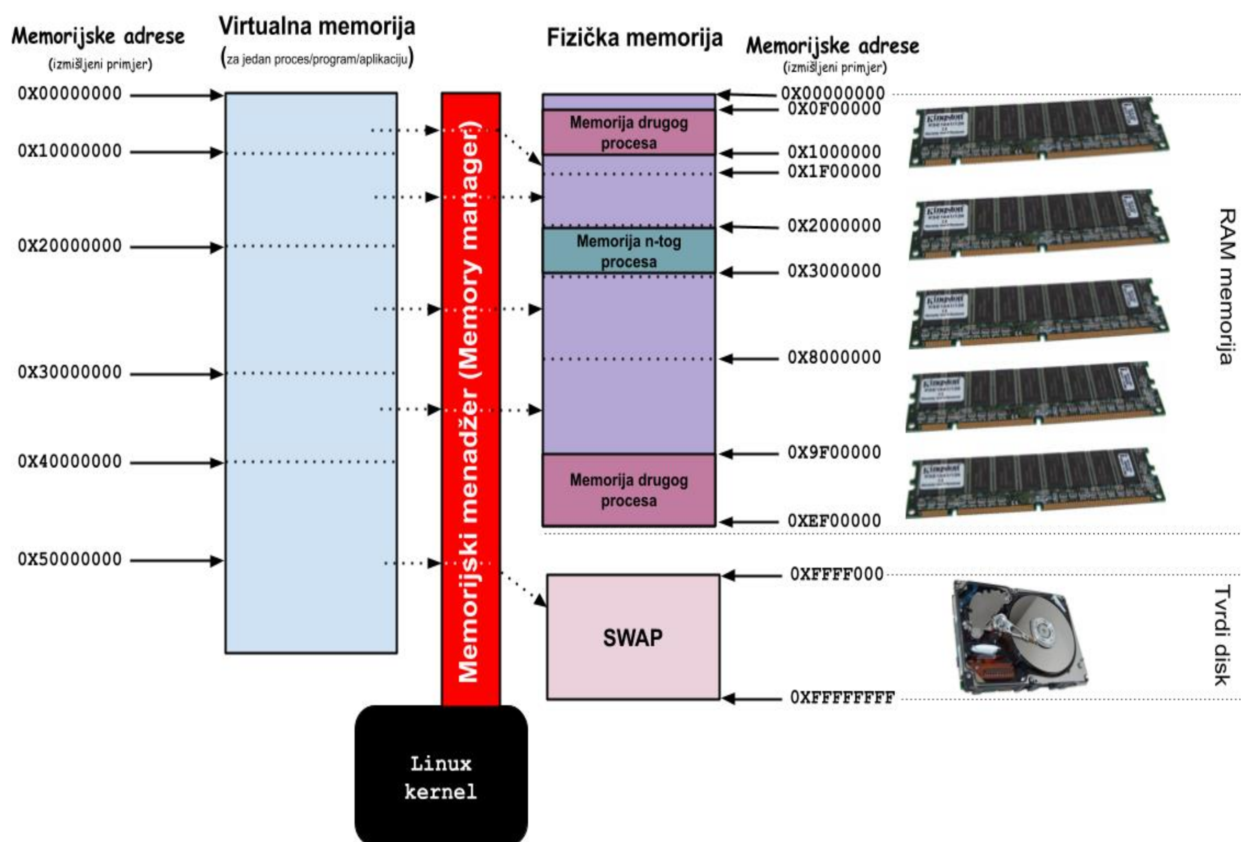
Dakle adresni prostor koji vîdi proces odnosno aplikacija, aplikaciji izgleda kao kontinuirani adresni prostor, a koji se dodatno sastoji od više dijelova odnosno takozvanih zona memorije.



Za detalje o ovim zonama memorije, pogledajte poglavlja:
**12.3.5. Zone RAM memorije.**

Slika 93, prikazuje način preslikavanja (translacije) virtualne memorije dostupne aplikacijama u fizičku RAM memoriju, dostupnu kernelu.

Slika 93. Logička shema sustava virtualne memorije




**Izvori informacija: (853),(855),(1196),(1197),(K-4).**

## 12.3.2. Stranice memorije (*paging*)

***Slijedi napredno poglavlje (12.3.2.x)!***

Prije nego su se uvele stranice memorije odnosno takozvani *Paging*, operativni sustav je morao učitati cijelu aplikaciju (proces) u RAM memoriju, popunivši ju u kontinuiranom nizu.

S obzirom na to da se ista stvar morala raditi za svaku aplikaciju, svaka od njih je zauzela memoriju u kontinuiranom nîzu. Problem je nastajao kada bi se neka aplikacija zatvorila (ugasila), te bi na tom mjestu ostala prazna odnosno nekorištena memorija. Gašenjem odnosno zaustavljanjem i pokretanjem novih aplikacija (programa/*procesa*) nastajale bi sve veće praznine (rupe) u memoriji koje bi uzrokovale probleme u radu sustava. Naime pokretanje novih programa postalo bi problematično, ako bi određeni program trebao onoliku količinu memorije u nizu, koje ne bi bilo dovoljno, iako bi ukupna količina dostupne memorije na sustavu bila dostatna za to. Samim time bi u radu došlo bi se do sve veće pojave takozvane fragmentacije memorije, koja bi se sve više povećavala. Dakle stalnim pokretanjem, a pogotovo zaustavljanjem raznih program i sistemskih servisa, što se stalno događa u radu sustava, nastajala bi sve veća količina dijelova memorije koje su oslobođene zbog zaustavljenih programa. Količina ovakvih dijelova memorije bi se zbog ovog načina alociranja i dealociranja memorije, sve više povećavala i u određenom trenutku bi imali ogroman broj dijelova memorije koji su neiskorišteni, ali premali, da bi se u njih u nizu mogao učitati iole veći program. Vratimo se u današnje vrijeme, u kojem se koristi virtualna memorija, koja je podijeljena na segmente ili takozvane stranice memorije (engl. Memory *Pages*) virtualnog adresnog prostora.

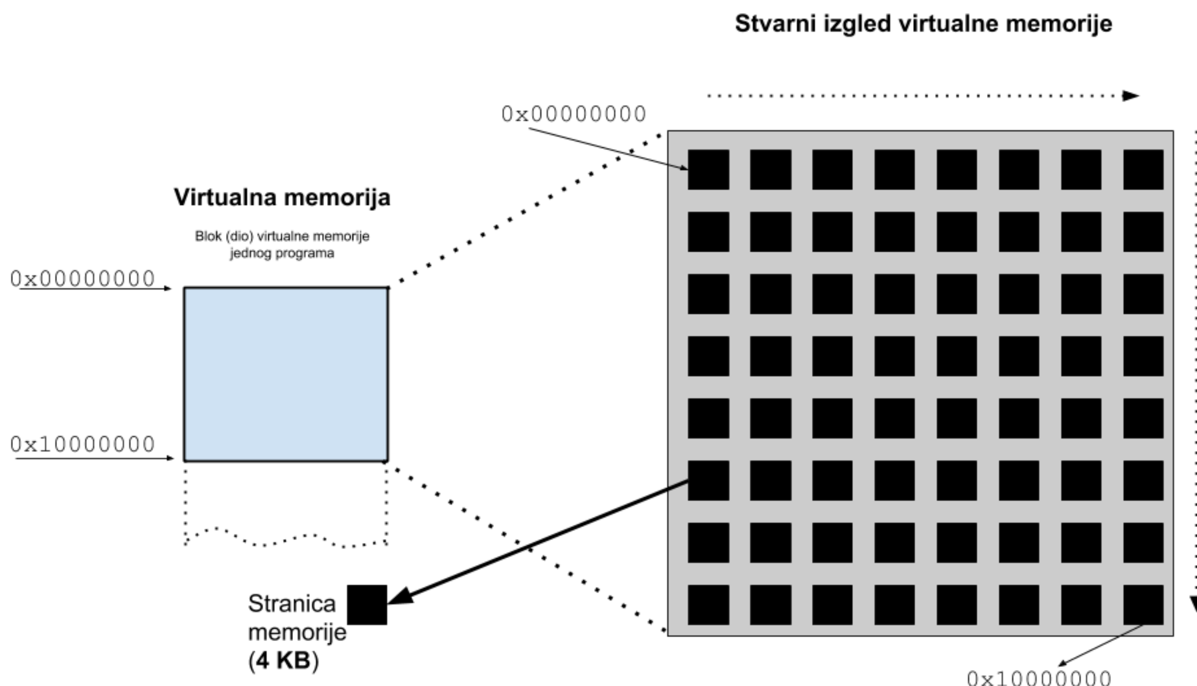
Naime u gotovo svim trenutnim implementacijama sustava virtualne memorije, virtualni adresni prostor memorije podijeljen je na stranice memorije, odnosno memorijske blokove (susjednih) adresa, sustava virtualne memorije.

Ove stranice su uglavnom veličine 4kB (4096 bajta). Dakle sva raspoloživa (virtualna) memorija je razlomljena u stranice memorije standardne veličine 4kB. Slično kao što je površina tvrdog diska razlomljena u sektore, koji su najmanje jedinice za zapis podataka na tvrdi disk. Dakle, ako bi pogledali sliku 93, i to samo lijevi dio slike: **Virtualna memorija**, vidimo da je ona sva podijeljena u blokove tj. segmente memorije koji obuhvaćaju određeni raspon memorijskih adresa.

Svaki od tih blokova memorije je potom podijeljen u stranice memorije (4kB), poput ovih na slici 94 dolje.

Pogledat ćemo sâmo jedan segment virtualne memorije, od memorijske adrese: **0x00000000** do adrese **0x10000000**, kako je vidljivo na slici 94.

**Slika 94. Pogled na jednu cjelinu virtualne memorije.**




Ako gledamo sâmo jedan segment virtualne memorije, od memorijske adrese: **0x00000000** do adrese **0x10000000**, vidljivo je da se on sastoji od većeg broja stranica memorije. Pri tome je (jedna) stranica virtualne memorije (4kB) najmanja jedinica virtualne memorije dostupna sustavu memorijskog menadžmenta, unutar podsustava virtualne memorije svakog današnjeg operativnog sustava. Standardnu veličinu stranice memorije (obično) određuje arhitektura procesora.

Stranice memorije u operativnom sustavu moraju imati jednoliku veličinu, na primjer 4096 bajta (4kB).

Međutim dizajn procesora (CPU) često dopušta dvije ili ponekad više istodobnih veličina stranica memorije, zbog bolje optimizacije rada, o kojoj ćemo govoriti kasnije. Ipak, ovakav rad u kojem se istovremeno mogu koristiti stranice memorije različitih veličina nije uobičajen, a niti jednostavan za implementaciju, pa se često ne koristi.

Više detalja ćete vidjeti u sljedećim cjelinama, a pogotovo u primjerima upotrebe naredbe `pmap`.

**Izvori informacija: (852),(853),(855),(K-4).**

### 12.3.2.1. Paging i dijeljenje memorije (*Shared Memory*)

Korištenjem sustava virtualne memorije, svaka aplikacija (proces) u memoriji, podijeljena je na memorijske stranice (*Memory pages*) i to dodatno tako da je izvršni dio programa u jednom dijelu, a dio koji sadrži podatke u drugom dijelu memorije.
Kako se neki program ugasi odnosno zatvori, oslobađaju se sve stranice memorije koje je zauzimao, te nakon toga one postaju dostupne drugim programima na korištenje.
Međutim postoji i dodatni problem vezan za efikasno iskorištavanje memorije u slučajevima kada dva ili više programa (aplikacije/procesa) žele učitati isti sadržaj (primjerice datoteke) u memoriju. Neefikasno bi bilo da svaki od njih učitava iste datoteke u memoriju jer bi se u jednom trenutku u memoriji našle višestruke kopije istih podataka (datoteka), s čime bi se memorija (RAM) nepotrebno prepunjavala.

Stoga je ovdje uveden još jedan mehanizam: mehanizam dijeljenja memorije. Memorija se dijeli tako što se unutar takozvane tablice stranica memorije (engl. *Page Table*) u kojoj se nalaze memorijske adrese (lokacije) za sve aplikacije (programe/procese) pokazuje na iste stranice memorije koje su dijeljene između njih.
Dijeljenje memorije (engl. *Memory Sharing)* se nadzire od strane komponente sustava koja se zove memorijski management (Engl. *Memory manager)*.

## Kako to radi?

Korištenjem modela dijeljenja memorije, više nema potrebe za dupliciranjem sadržaja i zauzeća memorije. Osim toga na ovaj način aplikacije (procesi) lakše mogu dijeliti podatke među sobom odnosno komunicirati. Dio memorijskog menadžmenta zadužen za dijeljenje memorije odnosno komunikaciju između procesa se zove ***IPC System*** (Engl. *Inter Process Communication*). Pojednostavljeno više aplikacija (procesa) dijeli dio memorije koji se koristi za njihovu međusobnu komunikaciju. Ti dijelovi memorije, zbog toga što se dijele između programa (procesa), moraju imati kontrolu: tko im može pristupiti, a tko ne može. Za to je potreban mehanizam zaključavanja pristupa (engl. *Locking*) tôm konkretnom dijelu memorije.

Ovdje se u priču uključuju i takozvani semafori (Engl. *Semaphores*), a oni su zapravo jednostavne zastavice (engl. *Flags*) koje označavaju u kojem stanju su kôje stranice (dijeljene) memorije.
Semafori odnosno zastavice rade tako da, ako se određeni dio dijeljene memorije trenutno koristi od strane nekog aplikacije (procesa), tada se podiže zastavica i druge aplikacije (procesi) koji bi u istom trenutku također željele pristupiti toj memoriji (na toj adresi), moraju malo pričekati, sve dok se semafor za pristup ne isključi. Naredba koja barata s **IPC** (*Inter Process Communication*) sustavom je `ipcs` mada se uglavnom koristi malo drugačiji način komunikacije između aplikacija (procesa/programa).



Vezano **IPC** (*Inter Process Communication*) sustav, on je objašnjen u poglavlju:
**9.4. Komunikacija između procesa**.

## Napredna konfiguracija dijeljene memorije u Linuxu

Veličina odnosno količina RAM memorije koja se može koristiti kao dijeljena memorija (Engl. *Shared Memory*), je definirana za cijeli operativni sustav, ali se može i povećavati ili smanjivati prema potrebi.
Njena veličina je definirana brojem blokova/stranica memorije, a standardne veličine je 4KB (4096 bajta).

Postoje dva ograničenja na veličinu dijeljene memorije koja je u Linuxu definirana sa sljedećim parametrima:
1. `SHMMAX` – (*sysctl* varijabla `kernel.shmmax`) - je maksimalna veličina jednog (bilo kojeg) segmenta dijeljene memorije.
2. `SHMALL` – (*sysctl* varijabla `kernel.shmall`)- je zbroj svih segmenata dijeljene memorije za cijeli operativni sustav.

Kako provjeriti ove (i par dodatnih) vrijednosti?. Provjera veličine bloka memorije odnosno veličine stranice memorije (*Page size [ona je obično veličine 4096 bajta odnosno 4KB])* radi se s naredbom `sysctl` na sljedeći način:

```
sysctl kernel.shmmni

kernel.shmmni = 4096
```

Provjera `SHMMAX` varijable i njene vrijednosti:

```
sysctl kernel.shmmax

kernel.shmmax = 68719476736
```

Provjera `SHMALL` varijable i njene vrijednosti:

```
sysctl kernel.shmall

kernel.shmall = 4294967296
```

**Pogledajmo izračun i opis navedenih parametara i njihovih vrijednosti te nekoliko primjera njihove promjene.**

### SHMALL

Ova (`SHMALL`) vrijednost može biti jednaka cijeloj RAM memoriji, jer želimo omogućiti upotrebu cijele RAM memorije kao potencijalno dijeljene memorije.

Izračun za **SHMALL**: $SHMALL = \frac{RAM\ (GB) \times 1024 \times 1024 \times 1024}{4096\ (SHMMNI)}$

Primjerice za 8GB RAM = 8 x 1024 x 1024 x 1024 = 8589934592 / 4096 = 4194304
Dakle `SHMALL` tada treba biti veličine 4194304 stranica memorije od 4kB (4096 *bajta*)

Dodajmo ovu vrijednost trajno, u datoteku `/etc/sysctl.conf` dodavanjem sljedećeg retka teksta (linije):

```
kernel.shmall = 4194304
```

### SHMMAX

Ova (`SHMMAX`) vrijednost definira maksimalnu veličinu u bajtima, pojedinog segmenta dijeljene memorije koji se uopće može alocirati u adresnom prostoru (preko Linuxovog sustava virtualne memorije).
Ova vrijednost, naravno, mora biti manja od `SHMALL`. U prosjeku, dovoljno je imati mogućnost kako bi se polovica RAM memorije mogla koristiti kao jedan segment dijeljene memorije.
Vrijednost `SHMMAX` je gornja granica, koliko maksimalno dijeljene memorije pojedini proces (aplikacija) može alocirati.

Izračun za **SHMMAX** je: $\boldsymbol{SHMMAX = RAM\ (GB) \times 1024 \times 1024 \times 1024}$

Dakle ako imamo 8GB RAM i želimo da ova granica bude 4GB tada je računica sljedeća:
4 GB x 1024 x 1024 x 1024 = 4294967296

Sada postavimo ovu vrijednost (u novi red) u `/etc/sysctl.conf` :

```
kernel.shmmax = 4294967296
```

Potom, nakon što smo napravili ove dvije promijene, potrebno je učitati ih i postaviti na sustav (za `SHMALL` i `SHMMAX)` :

```
sysctl -p
```

Nakon prvog restarta računala sve ove promjene će biti učitane automatski.



Vezano za **sysctl** varijable i postavke pogledajte i poglavlje**:**
**4.5.8. Naredba *sysctl*.**

**Izvori informacija: (221),(852),(853),(854),(855),(K-4), `man sysctl`,`man 5 sysctl.conf`.**

### 12.3.2.2. Uloga tablice adresa (*Page Table)*

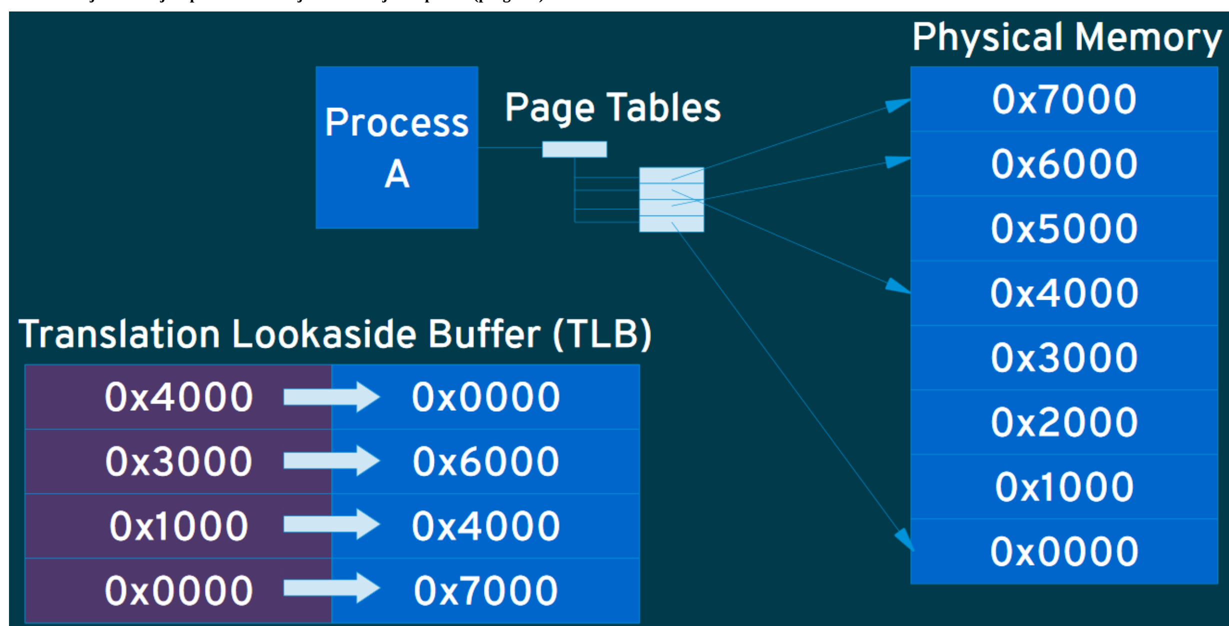
Tablica memorijskih stranica (adresa) odnosno tzv. ***Page Table*** je strukturirana tablica u RAM memoriji, u kojoj se nalaze sve poveznice između virtualnih i fizičkih memorijskih adresa. U operativnim sustavima koji koriste virtualnu memoriju, a to su svi moderni sustavi, svaki pokrenuti program (proces) vidi svoju memoriju koju koristi kao kontinuiranu. Zapravo je ta memorija raspoređena po fizičkoj RAM memoriji i eventualno se još nalazi dijelom na *Swap* particiji tvrdog diska. Kada aplikacija (proces) traži pristup memoriji u nadležnosti je operativnog sustava da prvo pogleda u „*Page Table*“ tablicu te napravi povezivanje (translaciju) virtualne memorijske adrese u fizičku memorijsku adresu te dohvati tražene podatke u RAM memoriji odnosno pročita ih ili snimi, a što je izvan područja znanja odnosno ovlasti sâme aplikacije, jer se ova procedura odvija u pozadini na razini kernela. Svaki unos u ovoj tablici je oblika: ***Virtualna adresa* ↔ *Fizička adresa***. Svaki pojedini unos u ovoj tablici se zove „*Page table entry*“ (***PTE***). Dakle u ovoj tablici svaka pojedina stranica virtualne memorije ima svoju adresu koja se povezuje sa stvarnom (fizičkom) adresom u RAM memoriji u koju se u konačnici i sve zapisuje od strane operativnog sustava, bez znanja sâmog programa/aplikacije. Naime program/proces/aplikacija vidi i može koristi samo virtualnu memoriju jer mu pristup prema i od nje osigurava Linuxov kernel. To je slučaji i na svim drugim modernim operativnim sustavima!.

### 12.3.2.3. Proces translacije adresa

U radu odnosno upotrebom sustava virtualne memorije, koju jedino i mogu koristiti svi programi i aplikacije (procesi), postoji i jedna dodatna brza tablica koja sadrži listu translacija memorijskih adresa RAM memorije: virtualne ← → fizičke (stvarne) kako bi se ova procedura dohvaćanja adresa ubrzala. Ovo ubrzanje je izvedeno tako da se ova nova među tablica zapravo nalazi unutar sâmog procesora. Dakle zapisivanje i čitanje u i iz nje se odvija vrlo velikom brzinom. Konkretno unutar procesora (CPU) se nalazi memorijski kontroler, a unutar njega *Memory Managemet Unit* (***MMU***) koji ima priručnu memoriju (engl. *Cache*) u kojoj čuva unose iz *Page* tablice, koji su nedavno korišteni. Ovdje se radi o internoj vrlo maloj memoriji odnosno međuspremniku procesora. Ovaj međuspremnik se zove “*Translation Lookaside Buffer*” (***TLB***). Naime svaki puta kada se virtualna adresa treba prevesti u fizičku, prvo se provjerava *TLB memorija*. Ako je unos pronađen i fizička adresa je poznata, nastavlja se s pristupom memoriji. Ako traženi unos ne postoji u *TLB* memoriji, tada dolazi do promašaja (ovo je tkzv. “*TLB miss*”), pa se nastavlja s čitanjem originalne velike “*Page*” tablice iz RAM memorije, i traži se potreban unos, a cijeli proces se zove “*Page walk*”.
Kada se unos pronađe on se odmah upisuje u *TLB* međumemoriju u nadi da će nekome ponovno zatrebati. Ako nekome nakon toga zatreba isti unos, pokreće se proces čitanja iz *TLB* memorije i dohvaćanje navedenog već pohranjenog unosa.

Potom se od kernela standardno nastavlja s pristupom fizičkoj memoriji (RAM) odnosno točnoj adresi u njoj, gdje se nalaze traženi podaci. Ovo je proces koji se stalno ponavlja tijekom svakog čitanja ili pisanja prema RAM memoriji. Međutim u *TLB* memoriju modernih procesora (CPU) stane tek nekoliko stotina unosa. *TLB* je prema tome specijalizirana ekstremno brza priručna memorija procesora čija je namjena brzo dohvaćanje memorijskih adresa za sustav virtualne memorije. Stoga moramo biti svjesni da što se više memorije alocira i dodjeli aplikacijama (procesima), “*Page table*“ tablica sve više i više raste.
Pošto je veličina stranica memorije (*Page size*) samo 4KB, to praktično znači da je za svakih alociranih 1GB RAM memorije potrebna *Page* tablica s 262.144 unosa, koji se stalno pretražuju, a oni zadnji korišteni se stavljaju u *TLB* međuspremnik, kako je gore objašnjeno. Ako primjerice imate 8GB RAM koja je potpuno iskorištena, to znači kako će *Page* tablica narasti na 2.097.152 unosa. Jasno je da to unosi znatne probleme kod aplikacija koje koriste veću količinu RAM memorije, jer za svaki pristup svakoj pojedinoj memorijskoj lokaciji slijedi proces od gore; dakle pretraživanje stotina tisuća ili milijuna unosa, za svako čitanje ili pisanje po svakoj lokaciji memorije, koja je standardne veličine 4KB.

**Slika 95. Pojednostavljeni proces translacije adresa za jedan proces (program).**



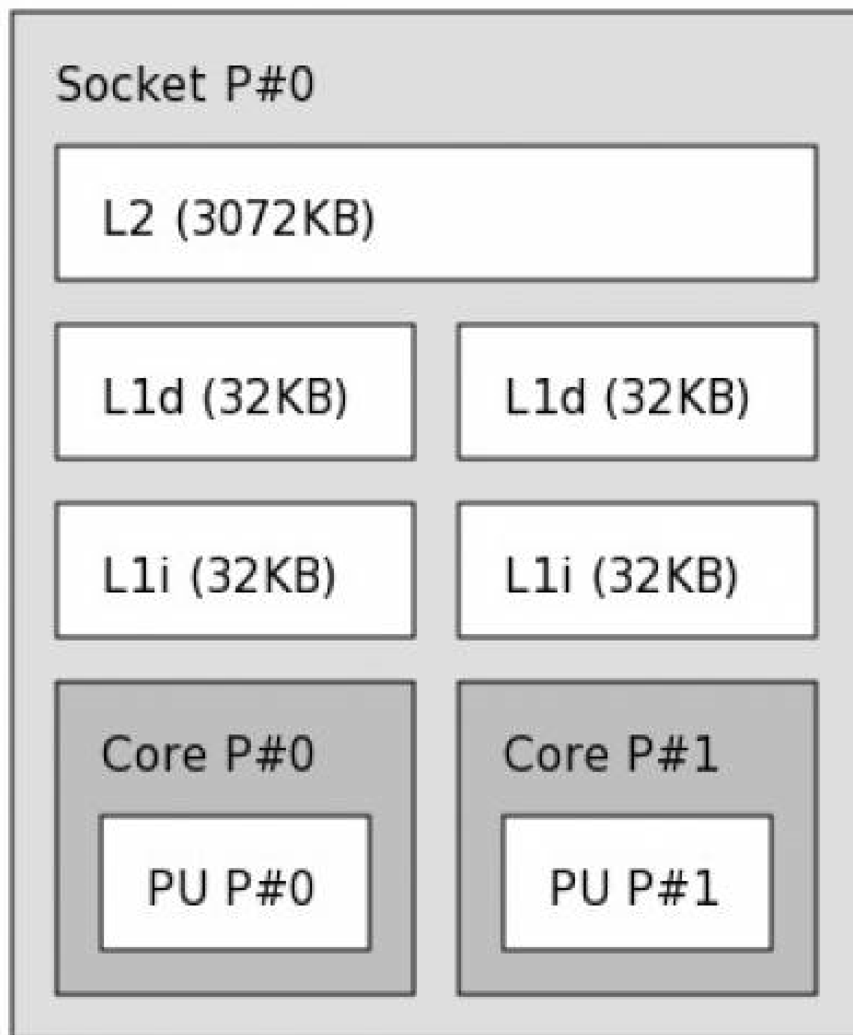

Izvor fotografije: https://people.redhat.com/jcm/talks/FOSDEM_2018.pdf

**Kolika je TLB međumemorija modernih procesora?.**
Za primjerice ***Intel*** ***Nehalem*** mikro arhitekturu odnosno generaciju CPUa, stanje po pitanju ***TLB*** memorije je sljedeće.
Ova generacija procesora ima dvo razinski ***Cache/TLB***: manji **L1** (***ITLB1*** i ***DTLB1***) koji je i najbrži te potom veći **L2** (***DTLB1***), koji je i malo sporiji.

Pogledajmo koje su njihove veličine odnosno kapaciteti:
1. **L1** (**DTLB1**): 64 unosa za stranice memorije veličine 4KB (standardna veličina) i 32 unosa za stranice memorije veličine 2 ili 4 MB (o ovim veličinama u naprednom dijelu knjige).
2. **L1** (**ITLB1**): 128 unosa za stranice memorije veličine 4KB (standardna veličina) i 14 unosa za stranice memorije veličine 2 ili 4 MB (o ovim veličinama u naprednom dijelu knjige).
3. **L2** (**DTLB**): 512 unosa za stranice memorije veličine 4KB (standardna veličina).


U ovakvoj arhitekturi (koja je danas standard), postoje odvojeni virtualni adresni prostori za instrukcije i za podatke, stoga i postojanje ***ITLB*** i ***DTLB*** memorije. Pri tome je:
- **ITLB**=*Instruction translation lookaside buffer - ovo je TLB memorija za instrukcije.*
- **DTLB**=*Data translation lookaside buffer - ovo je TLB memorija za podatke.*

Pogledajmo sada procesor: ***Intel Core 2 Duo E7500***, sa dvije fizičke jezgre, na slici **96.**
**Slika 96. Pogled na Intel Core 2 Duo E7500, njegove jezgre i priručne memorije (L1 i L2)**

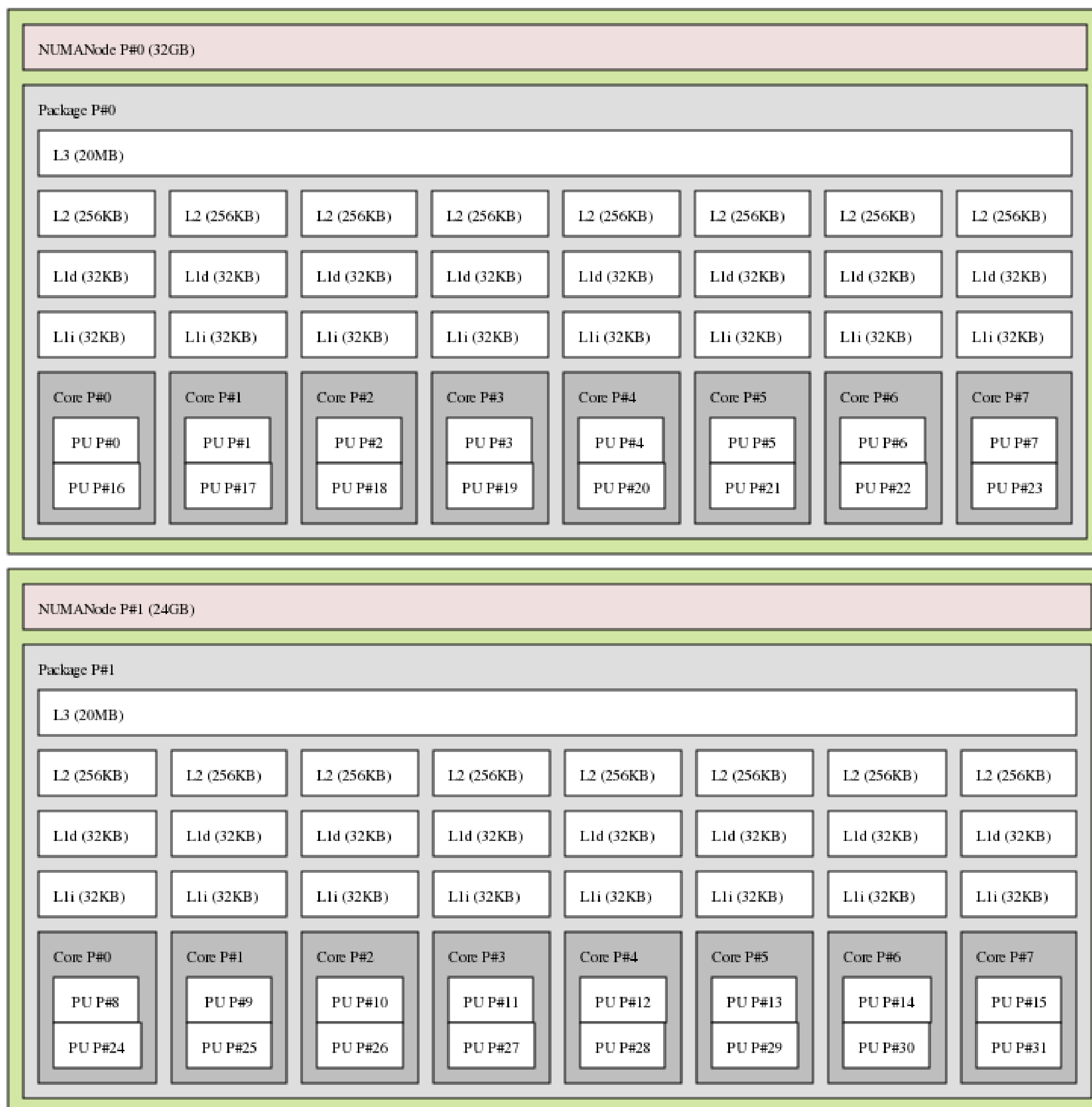



A sada pogledajmo i noviji poslužiteljski procesor ***Intel Xeon*** i to model **E5-2658.** Slika dolje prikazuje sustav s 54GB RAM, od kojih 32GB pripada prvom fizičkom procesoru a 24GB drugom. Ova logička shema je dobivena s naredbom `lstopo`:

```
lstopo --whole-system --no-io
```

**Slika 97. Logički izgled procesora (CPU) *Intel Xeon E5-2658*:**

Machine (56GB total)




Ovisno o arhitekturi sâmog procesora, ***TLB*** se nalazi unutar **L1**, **L2** ili unutar ***MMU*** dijela memorijskog kontrolera procesora (CPU-a). Na slici 97. vidljivo je kako prvi fizički procesor (**Socket P#0**) ima osam fizičkih jezgri (**P#0** do **P#7**), a unutar svake od njih se nalaze po dvije logičke jezgre (to su takozvane *Hyper Threading* jezgre), oznaka: ***PU P#"BROJ jezgre"***.

Istu konfiguraciju ima i drugi fizički procesor. Naime ovo je poslužiteljska matična ploča koja ima dva procesora ***Intel Xeon E5-2658*** sa ukupno 32 jezgre; 16 na prvom, a 16 na drugom fizičkom procesoru. Dakle ovo je **NUMA** arhitektura procesora.

## Vratimo se na TLB

Sada ćemo malo detaljnije objasniti rad TLB-a.

Dobro je znati kako se proces koji slijedi kod *TLB* promašaja (engl. *TLB miss*) koji smo opisali, može odraditi na dva načina:

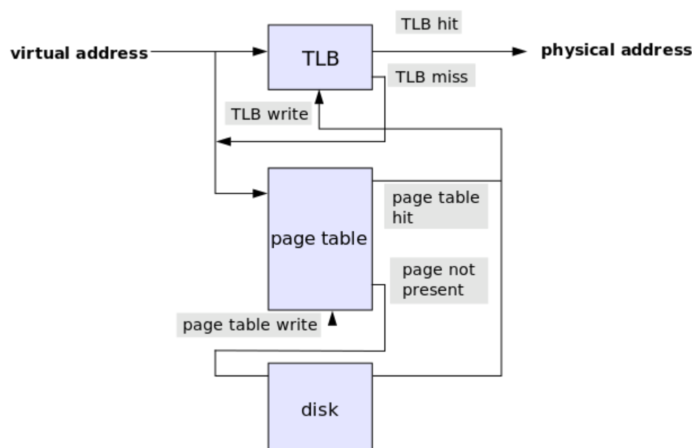
1. Hardverski - korištenjem registara procesora (CPU) CR3 registar na x86 arhitekturi; CPU odrađuje pretraživanje, a ostatak operativni sustav.
2. Softverski - operativni sustav odrađuje sve: od popunjavanja *TLB*-a, pretraživanja (*page walk*) i drugog.

***Koje su prosječne performanse TLBa:***

- Vrijeme dohvata unosa: **0,5** - **1** takt procesora (engl. *clock cycle*).
- Promašaj: **10** - **100** taktova procesora (engl. *clock cycles*). *TLB* promašaj (ili *TLB miss*) događa se kada procesor pokuša pristupiti virtualnoj adresi koja nije trenutno pohranjena u *TLB međumemoriji*. To se može dogoditi: ako se prvi puta pristupa određenog virtualnoj adresi pa unos ne postoji u *TLB* već se mora dohvatiti iz velike ***Page Table*** tablice iz RAM memorije ili ako je *task scheduler* obradu prebacio na novi proces (*context switch*) pa je potrebno osvježavanje tablice za novi proces to jest program ili jer se dogodilo čišćenje zastarjelih unosa u TLB-u.
- Prosječni postotak promašaja iznosi: **0,01** - **1** %.

Primjerice, ako je vrijeme dohvata 1 takt, a promašaj 30 taktova te postotak promašaja vršnih 1% tada je potrebno 1,3 taktova procesora za dohvaćanje jednog unosa u ***TLB*** tablici.

**Slika 98. Proces dohvaćanja podataka iz *Page* tablice i TLB memorije.**




**Zbog čega toliko priče o TLB-u?**
Znamo kako operativni sustav istovremeno obrađuje više programa (procesa) u pozadini. To „*istovremeno*" baš i nije potpuno točno. Naime jedan procesor, odnosno u novijim arhitekturama se to odnosi na jednu jezgru procesora u bilo kojoj jedinici vremena, u stanju je obrađivati samo jedan tôk informacija odnosno jedan proces ili nît procesa (programa/aplikacije), potom se prebacuje na drugi, pa na sljedeći i tako dalje u krug. Ovo prebacivanje se zove *Context Switching*. Ono nam daje mogućnost ili privid višezadaćnosti odnosno *Multitaskinga*: privid da jedan procesor može obrađivati više zadataka.
Kod ovog prebacivanja između procesa (programa/aplikacija), ***TLB*** tablica se u većini slučajeva mora isprazniti (engl. *TLB Flush*), pošto je vrlo vjerojatno da će većina, ako ne i gotovo svi unosi u tablici biti nevažeći za drugi proces/program, jer se svaki proces nalazi u drugom (izoliranom) adresnom prostoru. Nevažeći zbog toga što je *TLB* memorija prilično malog kapaciteta, a svaki proces koristi druge dijelove memorije, pa prema tome poveznice koje su ostale u *TLB* memoriji pokazuju na translacije (virtualna memorija ←→ fizička memorija), za adrese koje novi proces ne koristi jer se on nalazi u nekom drugom opsegu memorijskih adresa koje su rezervirane za njega. Svako novo prebacivanje između procesa odnosno svaki *Context Switch* prazni *TLB* memoriju te ju ponovno popunjava pretraživanjem glavne *Page* tablice. S obzirom na to kako se svake sekunde događa ogroman broj ovih prebacivanja, jasno je koliko ovo može utjecati, a i utiče na performanse sustava. Ovo nije jedini problem koji uzrokuje *Context switching*, isto tako se uz pražnjenje TLB-a, briše i većina registara procesora, koji se moraju ponovno učitati i popuniti tijekom svake nove promjene odnosno prebacivanja na izvršavanje programa (procesa).

**Virtualizacija i ugniježđena virtualizacija**
Svi programi u radu koriste virtualne adrese RAM memorije. Kada program traži pristup memoriji, procesor prevodi virtualne adrese u fizičke adrese RAM memorije, pomoću ***Page table*** tablice, te u konačnici preko specijalizirane **TLB** tablice/memorije u procesoru. Prilikom pokretanja virtualnog računala, ono ima alociranu virtualnu memoriju fizičkog računala (***hipervizora***) koja služi kao fizička memorija za virtualno računalo, a isti proces prevođenja adresa odvija se i unutar virtualnog računala. Ova operacija povećava troškove pristupa memoriji budući da se prevođenje adresa mora izvršiti dvaput – jednom unutar virtualnog računala (pomoću softverski emulirane ***Page table*** tablice za virtualna računala) i još jednom unutar glavnog sustava fizičkog računala (*hipervizora*). Kako bi ovo prevođenje adresa radilo učinkovitije, bilo je potrebno implementirati softversku tablicu prevođenja adresa. Ova softverska tablica prevodi virtualnu memoriju virtualnog računala izravno u fizičke memorijske adrese *hipervizora*. Pri tome svako virtualno računalo mora imati svoju zasebnu tablicu, a *hipervizor* je zadužen za upravljanje s njima. Međutim gubitak na performansama je tada vrlo velik je svaki put kada virtualno računalo ažurira svoju tablicu, pokrenut će *hipervizor* da upravlja dodjelom svoje tablice i njezinim promjenama.
Stoga je u novije procesore uvedena **SLAT** tehnologija. Ona tretira svaku adresu unutar virtualnog računala kao virtualnu adresu fizičkog računala (*hipervizora*). Upotrebom ove dodatne tablice (**SLAT**) drastično se ubrzao rad prilikom prevođenja adresa.

Dodatni problem je ugniježđena virtualizacija, koja se odnosi na virtualizaciju koja se izvodi unutar već virtualiziranog okruženja. Drugim riječima, to je mogućnost pokretanja *hipervizora* unutar virtualnog računala, koje i samo već radi na *hipervizoru*. Međutim, da bi i takva vrsta virtualizacije radila dovoljno brzo, potrebna je navedena hardverska podrška ugrađena unutar centralnog procesora. I ovdje se radi o ***Second level address translation*** (**SLAT**) odnosno upotrebi **TLB** tablice druge razine.

Tvrtka **AMD** podržava ***SLAT*** tehnologiju kroz *Rapid Virtualization Indexing* (**RVI**) tehnologiju od predstavljanja svoje treće generacije *Opteron* procesora (kodno ime *Barcelona*). Dok je **Intelova** implementacija ***SLAT***-a, poznata kao *Extended Page Table* (**EPT**), predstavljena u mikro arhitekturi *Nehalem*. Proširenje **ARM** arhitekture procesora s podrškom za ***SLAT***, poznato je kao ***Stage-2*** tablica (***Stage-2 page-tables***).

**Izvori informacija: (221),(852),(853),(854),(855),(1469),(K-4),(K-5).**

## 12.3.2.4. KSM (*Kernel same-page merging*)

***KSM*** (Engl. *Kernel same-page merging*) odnosno spajanje istih to jest potpuno identičnih stranica memorije je funkcionalnost kernela koja omogućuje sustavu dijeljenje identičnih memorijskih stranica između više procesa ili u slučaju virtualizacije: između više virtualnih računala. Naime u slučajevima upotrebe ove tehnologije kod virtualizacije na Linuxu, preko Linux ***KVM*** (Engl. *Kernel Virtual Machine*) komponente, ***KVM*** može koristiti ***KSM*** mehanizam između više virtualnih računala.

**Kako *KSM* radi?**
*KSM* odrađuje dijeljenje memorije s istim sadržajem, skeniranjem cijele memorije i pronalaženjem dupliciranih memorijskih stranica. Prisjetimo se sustava virtualne memorije u kojem se sve što se čita ili zapisuje u sustav virtualne memorije, zapisuje u najmanje logičke cjeline koje nazivamo stranice memorije.

Te stranice su zapravo najmanji „blokovi podataka“ koji se zapisuju u RAM memoriju. U normalnom radu, često se događa da više aplikacija (programa/procesa) u nekom trenutku rada imaju određene blokove podataka koji su identični. Ovakav scenarij postaje još vidljiviji kod virtualizacije.
Zamislimo slučaj kada na jednom poslužitelju pokrećemo 10 virtualnih računala, koja imaju identični operativni sustav, iste upravljačke programe, te veliki broj identičnih sistemskih programa koje pokreću. Dakle u RAM memoriju se učitava cijeli nîz podataka koji je identičan za svih ovih 10 virtualnih računala. Zašto bi se onda isti sadržaj kopirao deset puta u RAM memoriju. Zadaća KSM-a je da u određenim vremenskim intervalima skenira cijelu RAM memoriju i pronalazi blokove podataka koji su 100% identični. U našem slučaju *KSM* bi naišao na cijeli nîz blokova stranica memorije, koji su identični:

- Počevši od kernela operativnog sustava.
- Upravljačkih programa te sistemskih servisa (*daemoni* u Linux terminologiji), drugih programa itd. …

*KSM* bi potom obrisao sve pronađene kopije i ostavio sam jednu od njih, a na pozicijama svih kopija bi stavio sâmo poveznicu na original, koji se dijeli između svih njih. Možemo reći kako se sve kopije spajaju (Engl. *Merge*) u jednu, koja se onda dijeli među njima odnosno radi se nešto poput „*Soft Linkova*“ na originalne podatke.
Sve te memorijske stranice koje je *KSM odradio* tako jer su duplikati, se označavaju kao „*copy-on-write*“ (**COW**). Zbog ovih oznaka kernel je u bilo kojem trenutku svjestan kako se radi o *KSM* stranicama i ako na bilo kojem procesu ili Virtualnom računalu koje koristi te memorijske stranice, dođe do bilo koje promjene, one se počinju zapisivati na druge memorijske adrese.
Podrška za *KSM* je uvedena od Linux kernela 2.6.32.x koji je objavljen krajem 2009 godine.

**Postoje i neke stvari KSM-a za koje se trebamo zapitati, koliko su dobre**
Zbog efikasnosti upotrebe ovakvog mehanizma, kernel mora proceduru pronalaska duplikata u memoriji raditi u određenim vremenskim intervalima, ali ne prečesto, kako ne bi trošio previše resursa, i to prvenstveno CPU resursa računala. Pošto je ovo prilično zahtjevan proces za CPU, potreban je mehanizam koji će paziti na korištenje CPU resursa s jedne strane, a s druge strane odraditi ovaj zadatak dovoljno brzo. S obzirom na činjenicu kako se stanje memorije stalno mijenja, i ovaj proces se mora pokretati dosta često, ali opet ne prečesto. Ovdje se radi o balansu između CPU resursa i RAM resursa:

- Koliko dobivamo više slobodne RAM memorije zbog ovog procesa.
- A koliko koristimo (zagušujemo) CPU i/ili pristup RAM memoriji.

Postoji i bojazan da ***KSM*** može predstavljati sigurnosno rizik, prema dokumentu na poveznici:
https://staff.aist.go.jp/k.suzaki/EuroSec2011-suzaki.pdf

**Mjerenja**
Testiranja od strane tvrtke ***RedHat***[638] rađena su na način da su pokrenuta 52 virtualna računala Windows XP, a svakom od njih je dodijeljeno 1GB RAM memorije. Svih 52 virtualna računala su uspješno pokrenuta na poslužitelju sa samo 16GB RAM, na kojem je bio uključen ***KSM***. Autorova osobna iskustva su iz područja upotrebe ***KSM*** mehanizama za virtualizaciju, gdje su pokazala solidne rezultate, bez vidljivih usporavanja sustava. Testni sustav iz 2015.godine je bio:

- *Proxmox VE v.3.4.x – platforma za virtualizaciju koja koristi Linux KVM/QEMU hipervizor za virtualizaciju.*
- *2 x Xeon E5-2650 (NUMA): 32 jezgre.*
- *128GB RAM.*
- *Oko 20 Virtualnih računala Linux (CentOS ili RedHat 6.x): svakom je dodijeljeno između 4 i 16 GB RAM.*
- *Virtualizacija: Linux KVM + QEMU.*

**Slika 99. Statistika zauzeća RAM memorije i upotrebu KSM memorije u Proxmox VE sustavu v.3.4. iz 2015.g.**

| Status | |
|---|---|
| Uptime | 9 days 01:55:28 |
| Load average | 0.69, 0.47, 0.51 |
| CPUs | 32 x Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz (2 Sockets) |
| CPU usage | 37.77% |
| IO delay | 0.00% |
| RAM usage | Total: 125.87GB<br>Used: 39.73GB |
| SWAP usage | Total: 17.00GB<br>Used: 0 |
| KSM sharing | 70.71GB |
| HD space (root) | Total: 33.34GB<br>Used: 1.59GB |

Iz slike 99. je vidljivo kako je:

- Stvarno zauzeće RAM memorije oko 39.7 GB.
- Da ***KSM*** odrađuje oko 70.7 GB RAM.

To znači kako bi zauzeće RAM-a bez ***KSM***-a bilo 70.7GB, a s njim u upotrebi je samo 39.7GB.
U slučaju ovakve primjene ***KSM***-a, on je očito dobrodošao.
***KSM*** postavke su dostupne iz ***sysfs*** datotečnog sustava, kao datoteke unutar vršnog direktorija: `/sys/kernel/mm/ksm/`.

**Izvori informacija: (638),(682),(683),(1407),`man 5 sysfs`.**

## 12.3.3. Osnovne naredbe za provjeru stanja memorije

### 12.3.3.1. Naredba *free*

Naredba free daje nam ispis stanja iskorištenosti RAM memorije i *SWAP* memorije (diska). Pogledajmo ispis na računalu s 3.5 GB RAM-a. Naredbu smo pokrenuli s prekidačem -m koji nam daje prikaz memorije u MB, te s prekidačem -w za detalje:

```
free -mw

               total        used        free      shared     buffers       cached   available
Mem:            3479        1521        1958          25         160         1050        1942
-/+ buffers/cache:           309        3169
Swap:           1999           0        1999
```

Vidljivo je da je ukupno dostupno 3497MB (oko 3.4GB) [total] jer je razlika od 3.5GB do 3.4GB iskorištena od strane Linux kernela. Od toga je iskorišteno 1521 MB [used], a slobodno je 1958 MB [free].
Pri tome, ono što je iskorišteno za posebnu namjenu je ono što je označeno kao:

- Shared - 25 (MB) memorija je koja se dijeli između procesa (dijeljena memorija).
- Buffers - 160 (MB) je međumemorija kategorije ***buffers***, za kernel i u konačnici za programe.
- Cached - 1050 (MB) memorija koja se automatski oslobađa za pokretanje novih programa (procesa) prema potrebi, a koristi se kao međumemorija za spremanje datoteka odnosno kao disk tj. *page cache* i kao *slabs* međumemorija.
- buffers/cache - prikazuje koliko se od ***buffer*** i ***cache*** memorija koristi odnosno koliko je maksimalno dostupno.
- available - prikazuje koliko je memorije dostupno za aplikacije jer je dio memorije rezerviran* od sustava za posebne aplikacije (uglavnom kernel procese ali i posebne programe), definirano u *vm.min_free_kbytes.
- Swap - prikazuje da li i koliko se koristi *Swap* na tvrdi disk. Cilj je da upotreba (Used) bude na nula jer, ako je on u upotrebi to je znak da nam je ponestalo RAM memorije te kako se umjesto nje počeo koristiti tvrdi disk, odnosno *swap particija* kao proširenje RAM memorije, a koja je ekstremno spora u odnosu na RAM memoriju.

Memorija koja je označena kao "***Buffers***" se koristi kao međumemorija za blok uređaje (Engl. *Block devices*), a sadrži meta podatke od datotečnog sustava kao i praćenje stranica memorije. Pod meta podacima se ovdje misli na podatke o tome što se nalazi unutar direktorija, koje ovlasti (engl. *permissions*) su nad tim datotekama kao i na podatke o memoriji s koje, ili na koju se čita s blok uređaja. Međutim ne odnosi se i na sâme podatke poput sadržaja datoteka odnosno sâmih datoteka.
Naime kada sustav zatraži stvarne podatke (sadržaj datoteka) iz *cache* predmemorije, kernel prvo traži metapodatke u *buffers* međumemoriji, pomoću kojih dolazi do stvarnih podataka (pr. datoteka) koji su pohranjeni u *cache* međumemoriji.
Dakle međumemorija označena kao "*Cache*" pohranjuje sadržaj sâmih datoteka. Ako se sjetimo da je na Linux/Unix sustavima i svaki uređaj neka vrsta datoteka ("*Everything is a file*") i činjenice da je RAM memorija tisuće puta brža od diska, te da je Linux optimiziran da sve što može drži u RAM memoriji, a tek kada je stvarno potrebno, to i sprema na tvrdi disk, ove "*Cache*" i "*Buffer*" memorije počinju sve više imati smisla i jasno ubrzavaju rad sustava. Statistike koje su vidljive pod: buffers/cache i available ovise o inačici kernela odnosno inačici *procps* paketa.

**Izvori informacija: (K-1), man free, man 5 procfs, man 5 sysfs.**

### 12.3.3.2. Sadržaj /proc/meminfo datoteke

Sadržaj /proc/ direktorija je specifičan po tome da su u njemu virtualne datoteke koje pokazuju na parametre, vrijednosti i informacije raznih dijelova sustava odnosno kernela. Čitanjem sadržaja datoteke /proc/meminfo dobivamo podatke o RAM memoriji koje nam prikuplja kernel. Pogledajmo kako to izgleda (ispisano u tri stupca):

```
cat /proc/meminfo

MemTotal:         3562724 kB   Mapped:             38140 kB    HardwareCorrupted:      0 kB
MemFree:          1952044 kB   Shmem:              26300 kB    AnonHugePages:      22528 kB
Buffers:           167456 kB   Slab:              174840 kB    HugePages_Total:        0 kB
Cached:           1115380 kB   SReclaimable:      144312 kB    HugePages_Free:         0
SwapCached:             0 kB   SUnreclaim:         30528 kB    HugePages_Rsvd:         0
Active:            614820 kB   KernelStack:         3648 kB    HugePages_Surp:         0
Inactive:          758876 kB   AnonPages:          90856 kB    Hugepagesize:        2048 kB
Active(anon):       91284 kB   Writeback:              0 kB    DirectMap4k:         8832 kB
Inactive(anon):     25872 kB   Dirty:                  0 kB    DirectMap2M:      3821568 kB
Active(file):      523536 kB   PageTables:         12188 kB
Inactive(file):    733004 kB   NFS_Unstable:           0 kB    VmallocTotal:   34359738367 kB
Unevictable:            0 kB   Bounce:                 0 kB    VmallocUsed:        297852 kB
Mlocked:                0 kB   WritebackTmp:           0 kB    VmallocChunk:   34359435124 kB
SwapTotal:        2047996 kB   CommitLimit:      3829356 kB
SwapFree:         2047996 kB   Committed_AS:      395020 kB
```

Na ispisu su vidljivi razni podaci o RAM memoriji, slično kao kod naredbe free, ali s puno više detalja.

**Dodatni izvor informacija: man 5 procfs.**

Objasniti ćemo samo neke od njih:

- `MemTotal` – ukupna dostupna RAM memorija, (u kilobajtima (kB)).
- `MemFree` – ukupna dostupna RAM memorija u kB, neiskorištena od strane sustava (slobodna).
- `Buffers` – ukupna dostupna RAM memorija u kB, iskorištena za "***buffer***"-e.
- `Cached` – ukupna dostupna RAM memorija u kB, iskorištena kao "***cache***" memorija.
- `SwapCached` – količina "*Swap*" memorije u kB, koja se koristi kao "*cache memory*".
- `Active` – ukupna količina "*buffer*" ili "*page cache*" memorija u kB, koja je u aktivnoj upotrebi.
- `Inactive` – ukupna količina "*buffer*" ili "*page cache*" memorije u kB, koja je slobodna i dostupna.
- `SwapTotal` – ukupna količina "*Swap*" memorije u kB.
- `SwapFree` – ukupna količina "*Swap*" memorije u kB koja je još slobodna.
- `Dirty` – ukupna količina memorije u kB, koja čeka kako bi se snimila na disk.
- `Writeback` – ukupna količina memorije u kB, koja je aktivno snimljena na disk.
- `Mapped` – ukupna količina memorije u kB, koja se koristi za mapiranje uređaja, datoteka, ili biblioteka korištenjem `mmap` sistemske naredbe.
- `Slab` – ukupna količina memorije u kB, u upotrebi od strane kernela kao međumemorija podatkovnih struktura
- `PageTables` – ukupna količina memorije u kB, rezervirana za "*Page*" tablicu (*tablicu za translaciju adresa*).
- `VMallocTotal` – ukupna količina memorije u kB, odnosno cjelokupni adresni prostor za virtualnu memoriju.
- `VMallocUsed` – ukupna količina memorije u kB, iskorištenog adresnog prostora za virtualnu memoriju.
- `VMallocChunk` – najveća količina kontinuiranog bloka memorija u kB od dostupnog virtualnog adresnog prostora.
- `HugePages_Total` – ukupni broj tzv.. "*Huge pages*" stranica, dostupnih za računalo.
- `HugePages_Free` – ukupni broj tzv.. "*Huge pages*" stranica, slobodnih za upotrebu.
- `Hugepagesize` – veličina za svaki blok "*Huge pages*"-a. Standardna veličina je 2MB.

---

### 12.3.3.3. Naredba *pmap* i mapiranje memorije

**Slijedi napredna cjelina!**
U slučajevima kada nas zanima zauzeće memorije, od strane bilo kojeg procesa odnosno programa, to možemo vidjeti i s naredbama `ps`, `top`, `htop` ili `atop`. Međutim njihovi prikazi nisu potpuno detaljni i u potpunosti precizni, već s njima možemo brzo i okvirno vidjeti zauzeće memorije za pojedini program tj. proces. Naime svaki program koji pokrećemo s diska, se učitava u virtualnu i u konačnici u RAM memoriju, odnosno njegov se "sadržaj" pohranjuje i *preslikava* u virtualnu memoriju koja je namijenjena tom programu/procesu, a o čemu se brine sâm kernel.
Kako smo već naučili, program, kada se učitava u virtualnu memoriju i u konačnici u RAM memoriju, zapisuje se u malim jedinicama koje se nazivaju stranice memorije odnosno *memory pages*. Te stranice memorije predstavljaju najmanju jedinicu podataka koja se može zapisati u sustav virtualne memorije. Cijelo vrijeme govorimo o sustavu virtualne memorije, a ne sâmo o RAM memoriji, jer sustav virtualne memorije čine i RAM memorija i *swap* particija kao njen produžetak, mada u ovoj priči možete zamisliti da se sve događa samo i isključivo u RAM memoriji.

Dakle svaki program koji se učitava u RAM memoriju (da pojednostavimo priču s virtualnom memorijom), učitava se u stranice memorije, koje su standardne veličine 4kB, a koje nam osigurava linux kernel. Kernel nam preko sustava virtualne memorije daje memorijski adresni prostor za naš program (proces) u koji se naš konkretan program može učitati. Ovaj adresni prostor kojem je dozvoljen pristup za pojedini proces/program se zove virtualni adresni prostor. Njega kernel povezuje sa stvarnim adresnim prostorom i stvarnim adresama RAM memorije, ali je to u radu bilo kojeg programa skriveno od strane kernela. Naime bilo koji program (proces) u linuxu, ne vidi stvarni adresni prostor u RAM memoriji, već samo ovaj virtualni, jer u Linuxu koristimo virtualnu memoriju; kao i u svim operativnim sustavima razvijenim zadnjih 40 godina. Malo se ponavljamo, ali je ovo sve važno dobro razumjeti za nastavak ove cjeline.

Naš program (proces) alocira virtualnu memoriju, kako nam dozvoljava linux kernel.
Odnosno, program se učitava u memoriju, u jedinicama memorijskih stranica o čemu se brine kernel. Te memorijske stranice, kernel grupira u regije odnosno blokove memorijskih adresa, sustava virtualne memorije, a čije *preslikavanje* se zapisuje u datoteku, specifičnu za svaki proces/program: `/proc/PID/maps`.

Poveznica u ovoj tablici odnosno navedenoj datoteci koja predstavlja tablicu je prema sljedećoj logici:

***memorijska adresa (početna-završna) ↔ blok podataka ↔ datoteka/objekt/program/proces***

Ova tablica (datoteka) nalazi se, za svaki proces (prema ***PID*** broju procesa), u datoteci: `/proc/PID/maps`.

**Malo komplikacija**
Zapravo se program ne mora učitati u cijelosti u virtualnu memoriju, već se prvo može učitati njegov inicijalni dio, a ostatak može ostati na disku, diskovnoj međumemoriji ili *page cache* međumemoriji. Kako se program dalje izvršava, može doći i do takozvanog linux *page fault* stanja, jer ostatak programa nije učitan u sustav virtualne memorije koja završava u RAM memoriji, ali je preslikan (*mapiran*) u virtualni adresni prostor. Pod pojmom *mapiran* podrazumijevamo poveznicu: adresni prostor ↔ blok (dio) programa, biblioteke ili drugog objekta. Tada linux kernel dohvaća i učitava ostatak program s diskovnog podsustava ili iz *page cache* međumemorije u stvarnu virtualnu memoriju. Sada aktivni linux proces (naš program), koristi virtualnu memoriju u koju je pohranjen, a dodatno koristi i određeni dio dijeljene memorije (*shared memory*) te dodatne mapirane datoteke (*mapped files*) kao i dio koji je vidljiv kao rezidentna memorija.
Naredba `pmap` može nam dati znatno više detalja o zauzeću sustava virtualne memorije za određeni program odnosno proces.

Drugim riječima, naredba `pmap` nam prikazuje tablicu mapiranja memorije za željeni program. Ova tablica pokazuje regije u sustavu virtualne memorije u kojima se nalaze, kako blokovi (dijelovi) našeg programa, tako i blokovi biblioteka i drugih objekata koje naš program koristi, bez obzira jesu li stvarno već učitani u sustav virtualne memorije ili ih kernel tek treba dohvatiti s diska.



Vezano za ovo dohvaćanje blokova odnosno dijelova programa s diska, pogledajte sljedeća poglavlja:
**12.3.3.4 Naredbe** ps **i** top **za memoriju.**
**12.4 Anatomija programa u memoriji.**

Naredbi `pmap` je u osnovnom načinu rada, dovoljno sâmo proslijediti ***PID*** broj procesa/programa čija memorijska statistika nas zanima. Poput sljedeće naredbe u kojoj tražimo detalje za proces/program/aplikaciju čiji ***PID*** broj je `18342`:

```
pmap 18342
```

Pri tome je `18342` ***PID*** od programa koji analiziramo (konkretno analiziramo program `mc`). Tada ćemo dobiti nešto poput:

```
...                  ...   ...    ...
0000000000400000   1044K r-x-- mc
0000000000705000     20K r---- mc
000000000070a000     20K rw--- mc
00007f62f2fb4000     60K r-x-- libbz2.so.1.0.6
00007f62f2fc3000   2044K ----- libbz2.so.1.0.6
00007f62f31c2000      4K r---- libbz2.so.1.0.6
00007f62f31c3000      4K rw--- libbz2.so.1.0.6
     ...             ...   ...     ...
00007f62f31c4000    148K r-x-- liblzma.so.5.2.2
00007f62f31e9000   2044K ----- liblzma.so.5.2.2
00007f62f33ea000     92K r-x-- libelf-0.166.so
00007f62f3805000      4K r---- libattr.so.1.1.0
     ...             ...   ...     ...
00007f62fe028000      4K rw---   [ anon ]
00007f62fe029000      4K r---- ld-2.17.so
00007f62fe02a000      4K rw--- ld-2.17.so
00007f62fe02b000      4K rw---   [ anon ]
00007ffee5275000   1156K rw---   [ stack ]
00007ffee53e3000      8K r-x--   [ anon ]
ffffffffff600000      4K r-x--   [ anon ]
 ----------------------------------------------
total             182088K
```

Ispis smo znatno skratili, ali vidljivo je slijedeće (prema stupcima):

- Memorijska adresa na kojoj se nalazi dio (blok) programa: biblioteka, sam program, dijeljeni objekt, … kao što je primjerice lokacija: *`00007f62f2fb4000`*.
    - Dakle konkretno, prvo vidimo za naš izvršni program `mc`, vidljiv u prva tri reda, kako je naš program razlomljen u tri bloka adresa:
        - `0000000000400000` - ovaj prvi blok je veličine 1044KB, i vidimo kako su mu ovlasti **`r`** i **`x`** po čemu je jasno kako je ovo izvršni dio našeg programa (jer ima ***eX****ecute* ovlasti).
        - `0000000000705000` - ovaj drugi blok je veličine 20KB i vidimo kako ima samo **`r`** (*Read*) ovlasti, dakle iz ovog bloka se može nešto čitati.
        - `000000000070a000` - ovaj treći blok je veličine isto 20KB i vidimo kako ima **`r`** i **`w`** po čemu možemo zaključiti kako se u ovaj dio memorije može i nešto zapisivati i iz njega čitati. Anatomiju linux programa/procesa u memoriji sada nećemo objašnjavati.
          Za detalje pogledajte slijedeće poglavlje.
    - Potom slijedi biblioteka koju koristi naš program (`libbz2.so.1.0.6`) koja je isto razlomljena u nekoliko blokova. Zatim slijede druge biblioteke i objekti, koji su isto razlomljeni u određene blokove memorijskih adresa, na kojima se nalaze.
- U drugom stupcu je veličina u KB (pr. *148K*).

- Zatim slijede ovlasti (pr. `r-x-`). Pri tome ovlasti mogu biti:
    - `r` - *read* – prava čitanja.
    - `w` - *write* – prava zapisivanja.
    - `x` - *execute* – pravo za izvršavanje.
    - `s` - *shared* – označava kako je dijeljena (s drugim procesima/programima).
    - `p` - *private* – označava kako je on tzv. „*copy on write*“ objekt.
- I u krajnjem stupcu je objekt/program/dijeljena biblioteka.
- ***Na samom kraju ispisa vidimo ukupnu veličinu programa u memoriji*** (`total 182088 KB`) - ovo je ***VIRT*** statistika.



Za detalje o ***VIRT*** statistici zauzeća memorije pogledajte poglavlje:
**12.6.1. Drugi granični slučajevi.**

Ako želimo malo više detalja, ovu naredbu možemo pozvati i s prekidačem `-x`, poput:

```
pmap -x 18342
```

Moguće je vidjeti i više detalja, s prekidačem `-X` ili još više s prekidačem -`XX`.

Sada ipak pogledajmo primjer izlistanja datoteke `/proc/18342/maps` (ispis smo maksimalno skratili):

```
less /proc/18342/maps

00400000-00505000         r-xp 00000000 08:02 84321294    /usr/bin/mc
00705000-0070a000         r--p 00105000 08:02 84321294    /usr/bin/mc
0070a000-0070f000         rw-p 0010a000 08:02 84321294    /usr/bin/mc
0070f000-00747000         rw-p 00000000 00:00 0
0239a000-02464000         rw-p 00000000 00:00 0           [heap]
7f62f2fb4000-7f62f2fc3000 r-xp 00000000 08:02 125863971   /usr/lib64/libbz2.so.1.0.6
7f62f2fc3000-7f62f31c2000 ---p 0000f000 08:02 125863971   /usr/lib64/libbz2.so.1.0.6
7f62f31c4000-7f62f31e9000 r-xp 00000000 08:02 125863966   /usr/lib64/liblzma.so.5.2.2
7f62f33e9000-7f62f33ea000 rw-p 00025000 08:02 125863966   /usr/lib64/liblzma.so.5.2.2
7f62f33ea000-7f62f3401000 r-xp 00000000 08:02 125832555   /usr/lib64/libelf-0.166.so
7f62f3401000-7f62f3600000 ---p 00017000 08:02 125832555   /usr/lib64/libelf-0.166.so
7f62f3600000-7f62f3601000 r--p 00016000 08:02 125832555   /usr/lib64/libelf-0.166.so
7f62f3601000-7f62f3602000 rw-p 00017000 08:02 125832555   /usr/lib64/libelf-0.166.so
7f62f3602000-7f62f3606000 r-xp 00000000 08:02 125834298   /usr/lib64/libattr.so.1.1.0
```

Statistike iz datoteke: `/proc/18342/maps` su u formatu:

```
address          perms offset  dev   inode       pathname
```

Prema navedenim stupcima to znači sljedeće:

- `address` – je memorijska adresa, kao i kod prijašnje naredbe, s time da točno vidimo koja je početna i koja završna memorijska adresa, svakog bloka memorijskih adresa.
- `perms` – ovo su ovlasti, koje smo već objasnili.
- `offset` – ovo je je *offset* odnosno adresa od koje kreće alociranje memorije.
- `dev` – je stupac u kojemu se za svaki objekt, ako je on neka vrsta datoteke na virtualnom datotečnom sustavu (**VFS**), vidi kojem uređaju pripada (ovo je *linux device*). Pogledajte poglavlje: **11.1.2 Uređaji (devices) ukratko** .
- `inode` – je stupac u kojem se za svaki objekt, ako je on neka vrsta datoteke na virtualnom datotečnom sustavu (**VFS**), vidi koji je njen ***inode*** broj.
- `pathname` – ovo je putanja ili naziv objekta/datoteke.



Za više detalja oko optimizacije sustava virtualne memorije, pogledajte poglavlje:
**12.6. Dodatne optimizacije sustava virtualne memorije.**

**Izvori informacija: (246),(247),(248),(249),** **`man pmap`**,**`man 5 procfs`**.

### 12.3.3.4. Naredbe *ps* i *top* za memoriju

Naredba ps se najčešće koristi za ispisivanje odnosno izlistanje aktivnih procesa, ali moguće je saznati i neke dodatne informacije o samim procesima, vezanim za memoriju. Pogledajmo i primjere ovakve upotrebe.

**1. Izlistajmo sve aktivne procese, uz pripadajuće vrijednosti *minor* i *major page faults* poruka, te naredbu s kojom su pokrenuti.**

```
ps -eo min_flt,maj_flt,cmd | tail -n +2 | sort -rn
```

Dobit ćemo sortirane (`sort -rn`) i poredane procese prema ***page faults*** porukama u prvom stupcu (`MINFL`):

```
 MINFL    MAJFL   CMD
63717194     0    cmaperfd -p 30 -s OK -l /var/log/hp-snmp-agents/cma.log
22752507 12662    /usr/lib/systemd/systemd-journald
17537037     9    /usr/sbin/irqbalance --foreground
12967513    49    /usr/lib/systemd/systemd --switched-root --system --deserialize 24
11710504     0    /bin/bash /usr/sbin/ksmtuned
```

Opis parametara koje smo pozvali s naredbom `ps -eo`:

- `min_flt` – ispiši broj ***minor page faulta***. Za detalje pogledajte poglavlje: **12.4. Anatomija programa u memoriji**.
- `maj_flt` – ispiši broj ***major page faulta***. Za detalje pogledajte poglavlje: **12.4. Anatomija programa u memoriji**.
- `cmd` – ispiši naredbu odnosno ime pokrenutog programa (procesa).

**1.1. Pogledajmo zauzeće memorije i druge podatke o specifičnom procesu, prema njegovom *PID* broju.**
U slučaju kada nas zanima zauzeće memorije točno određenog procesa, primjerice ***PID*** broja `18254` (konkretno je to KVM/QEMU proces) to možemo vidjeti sa naredbom ps na sljedeći način (skratili smo ispis):

```
ps aux q 18254

USER    PID   %CPU %MEM    VSZ     RSS    TTY   STAT START   TIME    COMMAND
root    18254 11.6  6.1 2660476  489572   ?      Sl 10:57    2:30    /usr/bin/kvm -id 100
```

Dakle u stupcu (`%MEM`) vidimo zauzeće od ukupne količine memorije, dok je u (`VSZ`) to prikaz ***VIRT*** statistike memorije.
Dok je u (`RSS`) djelu statistike vidljivo stvarno odnosno takozvano rezidentno (***RES***) zauzeće memorije u KB.



***Ako imate potrebu, možete poredati pokrenute programe (procese), primjerice prema iskorištenju memorije, ovako:***

```
ps -eo pid,cmd,%cpu,vsz,rss,%mem --sort=-%mem
```

**2. Pogledajmo pripadajuće vrijednosti *minor* i *major page faulta* te naredbu s kojom je pokrenut točno određeni proces, prema njegovom PID broju.**
Nas sada zanima naš *Apache httpd* servis čiji ***PID*** broj je `6441` pa ćemo koristiti sljedeće opcije programa ps:

```
ps -o min_flt,maj_flt,cmd 6441

 MINFL  MAJFL CMD
  1932      0 /usr/sbin/httpd
```

Podsjetimo se: „***major page faults***" treba biti što manji (težiti nuli) ili imamo nekih problema koje smo opisali ranije.
Naredba top također nam može prikazati ***MAJFL*** i ***MINFL*** vrijednosti.

Pokrenimo ju, s opcijom s kojom kažemo kako želimo poredati sve procese prema ***MAJOR FAULT***:

```
top -o +vMj
```

Stisnite slovo **f** i pojavit će se izbornik, s poljima koja želimo da budu prikazana.
Sada s kursorskim tipkama odaberite polja: *vMj* (stisnite tipku razmak) te idite na polje: *vMn* i stisnite razmak.

Nakon toga oba polja moraju imati znak * te promijeniti boju u bijelo, kao na slici 100.

**Slika 100. Konfiguracija programa `top` u radu.**

```
Fields Management for window 1:Def, whose current sort field is vMj
   Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
   'd' or <Space> toggles display, 's' sets sort.  Use 'q' or <Esc> to end!

* PID     = Process Id               ENVIRON = Environment vars
* USER    = Effective User Name    * vMj     = Major Faults delta
* PR      = Priority               * vMn     = Minor Faults delta
* NI      = Nice Value               USED    = Res+Swap Size (KiB)
* VIRT    = Virtual Image (KiB)      nsIPC   = IPC namespace Inode
* RES     = Resident Size (KiB)      nsMNT   = MNT namespace Inode
* SHR     = Shared Memory (KiB)      nsNET   = NET namespace Inode
* S       = Process Status           nsPID   = PID namespace Inode
* %CPU    = CPU Usage                nsUSER  = USER namespace Inode
* %MEM    = Memory Usage (RES)       nsUTS   = UTS namespace Inode
* TIME+   = CPU Time, hundredths
* COMMAND = Command Name/Line
```

Sada stisnite tipku ***ESC*** i s desne strane će se pojaviti oba polja (*Major Faults delta* i *Minor Faults delta*).

**3. Ako želimo vidjeti samo osnovne podatke o sustavu virtualne (i RAM) memorije za svaki pojedini proces/program, samo normalno pokrenimo naredbu `top` i dobit ćemo nešto poput:**

```
top - 10:25:29 up 107 days, 23:08,  1 user,  load average: 0.06, 0.07, 0.01
Tasks: 557 total,   1 running, 556 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.8 us,  0.5 sy,  0.0 ni, 98.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  98989844 total, 86677952 used, 12311892 free,  1974044 buffers
KiB Swap: 91488252 total,   128660 used, 91359592 free, 62241620 cached

    PID USER       PR  NI  VIRT  RES  SHR S  %CPU %MEM     TIME+  COMMAND
 489972 48         20   0  234m 126m 2596 S  18.2  0.1   0:19.97 ruby
 634197 500        20   0 6568m 2.5g 6872 S   9.9  2.6  11189:50 java
   3881 root       20   0 13.6g  13g 3828 S   4.0 13.8   6522:53 kvm
   3842 root       20   0  208m  39m 4324 S   2.6  0.0   1347:25 pvestatd
   2274 root       10 -10  305m  98m 5996 S   0.7  0.1 564:37.30 ovs-vswitchd
   3168 root       20   0  355m  26m  25m S   0.7  0.0 405:51.43 pmxcfs
```

Memorije označene kao `buffers` i `cached` se odnose na razne međumemorije. Pogledajte poglavlja: **12.3.6.**, **12.5.** i **12.6.1**.
Ispis je skraćen zbog potrebe prikaza, a mi ćemo se sada fokusirati na donji dio prikaza za označene stupce, pri čemu za svaki proces pod ovim statistikama vidimo:

- `VIRT` označava virtualnu veličinu memorije koju proces može koristiti. Ona je zbroj memorije koju zapravo program (proces) i koristi (`RES`) uz datoteke na disku koje se mapiraju u njega, a ovo se odnosi uglavnom na dijeljene (`SHR`) biblioteke ili memoriju koja se dijeli s drugim procesima, ali i drugim otvorenim datotekama uz *SWAP* memoriju, ako se koristi. Dakle ovo je zbroj svih navedenih memorija uz dodatak kako nam ova veličina praktično govori koliko maksimalno memorije program može alocirati u datom trenutku. To ne znači da se sva ova memorija stvarno i koristi, već koliko bi je proces trenutno, maksimalno mogao koristiti, ako mu zatreba ili koliko je parcijalno otvorenih datoteka čija puna veličina ulazi u ovu statistiku. Pogledajte poglavlje: **12.6. Dodatne optimizacije sustava virtualne memorije** te: **12.6.1. Drugi granični slučajevi**, a pogledajte i poglavlje: **12.3.3.3. Naredba pmap i mapiranje memorije.**
- `RES` označava veličinu rezidentne memorije, što je točan prikaz koliko stvarne fizičke memorija proces troši. To također odgovara izravno stupcu `%MEM`. Ova veličina će uvijek biti manja od `VIRT` veličine.
- `SHR` označava koliko je od `VIRT` memorije moguće dijeliti s drugim procesima (Engl. *Share*), zapravo se radi ili o dijeljenoj memoriji ili dijeljenim bibliotekama (Engl. *Libraries*). U slučaju dijeljenih biblioteka, to ne znači nužno da je čitava biblioteka rezidentna (pohranjena direktno u RAM memoriju). Na primjer, ako program upotrebljava sâmo nekoliko funkcija u biblioteci, cijela je biblioteka mapirana i bit će ubrojena u `VIRT` i `SHR`, ali sâmo će oni dijelovi ove biblioteke s funkcijama koje se koriste zapravo biti učitani u memoriju i u konačnici se i računati pod `RES`.
- `%MEM` označava postotak od ukupne raspoložive memorije koju koristi određeni proces.



Za optimizacije sustava virtualne memorije, kao i detalje o njoj pogledajte poglavlje:
**12.6. Dodatne optimizacije sustava virtualne memorije.**
Za osnove korištenja naredbe **`top`** pogledajte poglavlje:
**10.7.2.3.2. Naredba *top*.**

Za druge primjere korištenja naredbe `ps` pogledajte poglavlja:
**9. Proces menadžment.**
**9.1. Prioriteti i procesi.**

**Izvori informacija: (628),(629),(630),(K-1), `man top`, `man ps`.**

### 12.3.3.5. Naredba *vmstat*

**Slijedi napredna cjelina!**
Naredba `vmstat` daje nam informacije o memoriji, disku i aktivnosti diska te aktivnosti procesora (CPU).
Sada ćemo se fokusirati na memoriju. Naredbu `vmstat` možemo pokrenuti samu ili s nekim od prekidača.

Mi ćemo ju pokrenuti s prekidačem koji će nam dati tablični ispis, što znači da pokrećemo: `vmstat -s`

```
vmstat -s

      3562724  total memory
      1854168  used memory
       706144  active memory
       881404  inactive memory
      1708556  free memory
       187628  buffer memory
      1290908  swap cache
      2047996  total swap
            0  used swap
      2047996  free swap
    492176813 pages paged in
    614859854 pages paged out
        27888 pages swapped in
       221618 pages swapped out
```

🡐 *skratili smo ispis na osnovne statistike*
Ovdje možemo vidjeti da je dio statistike za memoriju vrlo sličan kao i iz prethodnih naredbi. Vrijednosti su u kilobajtima (kB).

Dodatno su nam zanimljive statistike u drugom dijelu (izrezanog) ispisa. Većina vrijednosti ovdje je kumulativna:

- `total memory` – ukupna količina dostupne RAM memorije (obično u kB).
- `used memory` – ukupna količina RAM memorije u upotrebi (obično u kB).
- `inactive memory` – ukupna količina neaktivne RAM memorije (obično u kB). Neaktivne stranice memorije su one koje se nisu koristile u posljednjih nekoliko sekundi, ali još uvijek su u memoriji i još uvijek se mogu brzo ponovno aktivirati.
- `free memory` – ukupna količina slobodne (neiskorištene) RAM memorije (obično u kB).
- `buffer memory` - ukupna količina RAM memorije koja se koristi kao prva faza predmemorije za čitanje ili zapisivanje na disk (obično u kB).
- `ages paged in` – je količina memorije u stranicama memorije pročitanih s diska u "***page cache buffer***". Standardna veličina stranice je 4KB. Ovo je praktično količina podataka koji se u konačnici čitaju s diska preko "***page cache***" memorije. Znatniji promet/podaci ovdje znače kako se traženi podaci ne nalaze u "***page cache***" memoriji te se moraju čitati s diska. Ovdje se isto teži da imamo što manje prometa odnosno podataka.
  **Ovo nisu vrijednosti vezane za *Disk I/O* već za virtualnu memoriju!.**
- `pages paged out` – je količina memorije u stranicama memorije koje se trebaju zapisivati na disk tj. u "***page cache buffer***". Standardna veličina stranice je 4KB. Ovo je praktično količina podataka koji se u konačnici zapisuju na disk preko "***page cache***" memorije. **Ovo isto nisu vrijednosti vezane za *Disk I/O* već za virtualnu memoriju!.**
- `pages swapped in` i `pages swapped out` se odnose na stranice memorije koje se čitaju ili zapisuju na *Swap* disk (kumulativno). Ovdje težimo manjem broju na oba mjesta jer ne želimo da dođe do toga kada se mora početi koristiti *swap* memorija.



Za informacije o ***SWAP*** sustavu, pogledajte poglavlje:
**13.8.2. Swap particija.**

*Ako naredbu* `vmstat` *pokrenemo bez prekidača tada ćemo dobiti i statistike o trenutnoj upotrebi SWAP diska; pod:* *--swap--* *i to redom:* `si` – koliko se čita sa *SWAP* diska te `so` što označava koliko se snima sa *SWAP* disk.

Naredba `vmstat` se inače najčešće koristi i za statistike rada diska, ili statistike od pojedine particije diska.



Naredba **`vmstat`** statistike vezane za memoriju izvlači iz datoteka: `/proc/meminfo`, `/proc/vmstat` i `/proc/stat`.

Vezano za RAM memoriju kao diskovnu međumemoriju, pogledajmo i još jedan ispis naredbe `vmstat`:

```
vmstat -S M
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 4  0      0 102028   1023  14958         0     0   3    17  378  666  1  1 95  0  0
```

Ovdje ćemo se fokusirati na dio ispisa koji smo zatamnili.

Naime Linux sustav prilikom čtanja podataka s diskovnih uređaja (HDD, SSD, NVMe, USB) te podatke prvo zapisuje u slobodne segmente RAM memorije koji se tada koriste kao predmemorija, iz koje ih je, ako je to kasnije porebno, moguće vrlo brzo dohvatiti. Isto se događa i prilikom zapisivanja, odnosno podaci se prvo zapisuju u ovu predmemoriju, a tek kasnije, tek kada je to najoptimalnije, i na površinu diska.

Ova predmemorija u RAM memoriji je podijeljena u dvije cjeline odnosno razine:

- Prva razina jest za podatke koji su tek pristigli (pr. za zapisivanje), a u ovoj statistici je vidljiva kao `buff`. Ova međumemorija je nešto manja i namjena joj je privremena pohrana blokova podataka, ali i pohranai određenih struktura virtualnog datotečnog sutava odnosno **VFS**-a, a koje su samo jedan dio takozvane **SLAB** međumemorije.

  Ova (`buff`) memorija je vidljiva u datoteci `/proc/meminfo`, primjerice s naredbom:

```
cat /proc/meminfo | grep Buffers
Buffers:         1051908 kB
```

- Blokovi podataka (sadržaj datoteka) iz prve razine se potom prebacuju u drugu (i veću) predmemoriju. Ova druga predmemorija je vidljiva kao `cache`, a naziva se *Page cache* predmemorija. *Page cache* se potom koristi za spremanje podataka koji su pročitani s diska ili trebaju biti zapisani na disk, kako bi se unrzale operacije čitanja ili zapisivanja na disk. Ukupnu količinu ove međumemorije možemo vidjeti sa:

```
less /proc/meminfo | grep Cached
Cached:         15545840 kB
```

Pogledajmo usporednu tablicu koja prikazuje razlike između upotrebe *buffer* i *cache* međumemorija.

| Značajka | Buffer (`buff`) | Cache (`cache`) |
|---|---|---|
| **Namjena** | Privremena pohrana koja se koristi za prijenos podataka između diska i RAM-a, olakšavajući učinkovite I/O operacije. | Pohranjuje podatke kojima se često pristupa. |
| **Način zapisivanja i pražnjenja** | Obično slijedi strategiju **FIFO** ili **FCFS**. | Koristi složene algoritme (**LRU**, **LFU**, **SLRU**, **ARC**) na osnovu kojih se procjenjuje što predmemorirati (i koliko dugo). |



Za informacije i detalje o predmemoriji vezanoj za datotečni sustav, proučite cijelo poglavlje:
**14.1. Diskovni ulazno/izlazni sustav (*I/O*).**

Posebno proučite i poglavlje:
**14.1.2. Optimizacija *Page Cache* sloja.**

**Izvori informacija: (857),** **`man vmstat`**, **`man 5 proc`**.

## 12.3.3.6. Naredba sar

**Slijedi napredna cjelina!**
Naredba `sar` dolazi unutar softverskog paketa `sysstat` koji smo do sada već instalirali. Pokrenimo ju na sljedeći način:

```
sar -r
```

Ova naredba ima mogućnost praćenje raznih statistika, od kojih nam je sada najzanimljivija statistika sustava virtualne memorije. U ovu svrhu ćemo koristiti prekidač `-r`. Sada ćemo snimati statistiku svake sekunde (**1**) i to četiri puta za redom (**4**).

```
sar -r 1 4

10:17:05 PM kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact
10:17:06 PM    345400 131569804     99.74    123800  61290092  94246900     55.98  93872232  34640924
10:17:07 PM    345196 131570008     99.74    123804  61290100  94246900     55.98  93871728  34640924
10:17:08 PM    345336 131569868     99.74    123804  61290100  94246900     55.98  93871728  34640924
10:17:09 PM    345220 131569984     99.74    123804  61290108  94247288     55.98  93871728  34640924
Average:       345288 131569916     99.74    123803  61290100  94246997     55.98  93871854  34640924
```

Opis našeg ispisa slijedi:

- `kbmemfree` - je ukupna količina slobodne memorije u kB.
- `kbmemused` - je količina memorije u upotrebi (u kB). Ovdje se ne računa memorija koju koristi sâm kernel.
- `%memused` - je postotak memorije koja je u upotrebi. Naime Linux pokušava iskoristiti svu moguću RAM memoriju za razne ***cache buffere***, koju također pomoću ugrađenih automatizama može osloboditi za potrebe procesa/programa/aplikacija. Pogledajte `sar -B` u poglavlju:**14.2 Praćenje performansi I/O sustava** - ***Primjer* 1.1.**
- `kbbuffers` je količina memorije koju sam kernel koristi za kernel ***buffer***, količina je u kB (kilobajtima).
- `kbcached` je količina memorije koja se koristi kao ***cache*** memorija za spremanje podataka, količina je u kB.
- `kbcommit` je količina memorije koja je trenutno potrebna za funkcioniranje sustava (u kB). Ovo mjerenje na osnovu kojeg sustav utvrđuje ima li dovoljno RAM memorije, uz eventualnu kombinaciju sa SWAP diskom, za normalan rad sustava, koji neće dovesti do stanja ***Out of memory***.
- `%commit` je postotak memorije potreban za trenutno opterećenje sustava u odnosu na ukupnu virtualnu memoriju (***RAM*** + ***SWAP***).
- `kbactive` je količina aktivne memorije u kB, memorije koja je nedavno još bila u upotrebi i obično nije oslobođena/oduzeta (engl. *Reclaimed*) osim, ako bi to bilo prijeko potrebno. Pod oduzeta mislimo na proces oslobađanja memorije od strane sustava (OS-a).
- `kbinact` je količina aktivne memorije u kB, memorije koja neko duže vrijeme nije bila u upotrebi i za koju je vjerojatnije da će moći biti oslobođena od strane sustava, za druge potrebe.

Najčešće ćemo ovdje pratiti stanja: `kbmemfree` i `kbmemused` jer nas kod problema s memorijom zanima je li sveukupna memorija u nekom trenutku naglo zauzeta ili oslobođena. Naredba `sar -R` daje nam neke dodatne statistike korištenja memorije. Također ćemo pratiti statistiku svake sekunde (**1**) i to četiri (**4**) puta za redom.

```
sar -R 1 4

Linux 2.6.32-42-pve (server1)  11/20/2015      _x86_64_        (4 CPU)

10:58:52 PM    frmpg/s    bufpg/s    campg/s
10:58:53 PM     -58.00       0.00       0.00
10:58:54 PM      43.00       0.00       0.00
10:58:55 PM       7.00       0.00       0.00
Average:        -97.75       0.00       0.25
```

Opis:

- `frmpg/s` - je broj stranica u memoriji (sjetimo se da je svaka stranica memorije veličine 4kB) koje je sustav oslobodio u svakoj sekundi. Negativna vrijednost predstavlja broj stranica memorije alociranih od strane sustava.
- `bufpg/s` - je dodatni broj stranica memorije korištenih kao ***buffer*** memorija od strane sustava, u svakoj sekundi. Negativna vrijednost predstavlja oduzimanje stranica koje su bile alocirane kao ***buffer*** memorija.
- `campg/s` - je dodatni broj stranica memorije korištenih kao ***cache*** memorija od strane sustava, u svakoj sekundi. Negativna vrijednost predstavlja oduzimanje stranica koje su bile alocirane kao ***cache*** memorija.

Ovdje možemo pratiti stanje ***buffer*** i cache memorije u odnosu na alocirane ili dealocirane stranice memorije za normalnu upotrebu. Ovisno o opterećenju sustava te vrsti pokrenutih servisa ili programa, ove vrijednosti mogu varirati od primjera do primjera. Velike oscilacije su moguće i u slučaju korištenja ili oslobađanja veće količine RAM memorije.

***Moguće je izvući i statistike (vezane za zauzeće memorije) iz već snimljenih SA datoteka; primjerice za 20-ti dan u mjesecu:***

```
sar -r --human -f /var/log/sa/sa20
```



Pogledajte poglavlje: **10.7.2.3.4. Naredba *sar* (NUMA ili SMP).**
Pogledajte i poglavlje: **25.3. Statistike, analiza i praćenje mrežnih paketa** te primjere s naredbom **sar.**

**Izvori informacija:** **`man sar`**.

### 12.3.3.7. Naredba *dmidecode*

**Slijedi napredna cjelina!**
Naredbu `dmidecode` smo već objasnili (**A**, **B**, **C**), pa sada sâmo ukratko: ona nam daje podatke o hardveru.
Sada nas zanimaju podaci o hardveru i to konkretno o RAM memoriji i memorijskom kontroleru.

Pogledat ćemo ispis s poslužitelja koji ima ***ECC*** memoriju i memorijski kontroler koji podržava *ECC* memoriju.

```
dmidecode -t memory

# dmidecode 2.11
SMBIOS 2.7 present.
Handle 0x1000, DMI type 16, 23 bytes
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
        Error Correction Type: Single-bit ECC
        Maximum Capacity: 192 GB
        Error Information Handle: Not Provided
        Number Of Devices: 9
Handle 0x1111, DMI type 17, 34 bytes
Memory Device
        Array Handle: 0x1001
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 8192 MB
        Form Factor: DIMM
        Locator: PROC 2 DIMM 9
        Bank Locator: Not Specified
        Type: DDR3
        Type Detail: Synchronous
        Speed: 1333 MHz
        Manufacturer: Not Specified
        Rank: 2
        Configured Clock Speed: 800 MHz
```

(Ispis je skraćen samo na trenutno najzanimljivije dijelove)

Što je ovdje vidljivo:

- `Error Correction Type: Single-bit ECC` – vidimo da je ovo *ECC* memorija koja podržava korekciju sâmo jednog bîta podataka.
- `Maximum Capacity: 192 GB` – maksimalan kapacitet RAM memorije koju podržava ovaj poslužitelj je 192 GB.
- Dalje vidimo podatke za svaki memorijski utor: pr.: `Total Width: 72 bits` – prvi memorijski modul je 72. bîtan, što je zbog *ECC*-a tj. uporabe memorije s *ECC* korekcijom.
- Dalje vidimo podatke za svaki memorijski utor: pr.: `Data Width: 64 bits` – prvi memorijski modul je 64. bîtan, što je ostatak od 72. bîta, što znači da je memorija 64.bitna (više o tome u poglavlju o ECC memoriji).
- Dalje vidimo podatke za svaki memorijski utor; pr.: `Size: 8192 MB` – veličina je 8GB.
- Dalje vidimo podatke za svaki memorijski utor; pr.: `Type: DDR3` – vrsta memorije je DDR3.
- Dalje vidimo podatke za svaki memorijski utor; pr.: `Speed: 1333 MHz` – brzina je 1333 MHz.
- Dalje vidimo podatke za svaki memorijski utor; pr.: `Locator: PROC 2 DIMM 9` – pozicija memorijskog utora na matičnoj ploči pripada uz CPU 2 (*NUMA arhitektura*), utor 9. Ovo je poslužitelj koji ima dva procesora (CPU-a).

**Izvori informacija:** `man dmidecode`.

## 12.3.4. Zbog čega toliko priče o *Page Table-u*

**Slijedi napredno poglavlje (12.3.4.x)!**
U poglavljima o virtualnoj memoriji često smo govorili o *page* tablici (*Page table*) i njenoj važnosti. Pogledajmo poslužitelj sa 128 GB RAM, od koje je u upotrebi većina odnosno oko ~127 GB.

Filtrirat ćemo sve iz statistike upotrebe RAM memorije `/proc/meminfo` što je vezano za pojam „***Page***“, na sljedeći način:

```
cat /proc/meminfo | grep Page

AnonPages:       47143856 kB
PageTables:        109852 kB
AnonHugePages:          0 kB
HugePages_Total:        0
HugePages_Free:         0
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        2048 kB
```

Prva stvar koja je vidljiva je ta da je *Page tablica* (`PageTables`) narasla na **107MB (109.852 kB)** što znači kako se u tablici nalazi oko **28.049.408** unosa za 4kB stranice memorije.
Ovdje se koristi standardna veličina stranice (*Memory page*) od 4KB.

Za više detalja o datoteci `/proc/meminfo`, pokrenite naredbu:

```
man 5 proc
```

Problem je u tome što se kod svakog pristupa RAM memoriji, koja je za naše aplikacije razlomljena u komadiće od 4KB, potrebno svaki puta pogledati u "*Page Table*" tablicu te pronaći (jednu po jednu) memorijsku lokaciju od 4KB te nakon toga i doći do nje.

Zamislimo da vaša aplikacija treba nešto upisati ili čitati u 40MB RAM memorije koja je zapravo razlomljena u blokove od 4KB, što znači da je potrebno čitanje **10.000** unosa u tablici kako bi se došlo do svih memorijskih lokacija s kojih treba čitati ili u koje treba nešto upisati.
U prethodnom primjeru s velikom količinom RAM memorije u upotrebi stvari postaju još gore.

Rješenje je upotreba tzv. "***Huge Pages***" odnosno prelazak s 4KB blokova na puno veće stranice memorije.
Za *x86* arhitekturu procesora je tu o pravilu 2MB.
S ovom metodom smanjili smo broj unosa u tablicu i sâmim time broj pregledavanja i dohvaćanja unosa u "*Page*" tablicu, popriličnо, točnije **512 puta**.
I naravno uvijek postoji problem; ovo nije moguće napraviti za cijeli sustav već samo za točno određeni dio memorije.

I još malo komplikacije: *Huge Pages* neće raditi sâm od sebe, već vaša aplikacija tj. programski jezik u kojem ste ju napisali, mora podržavati ovakav način rada. Dobra strana je to što većina ozbiljnijih aplikacija zna koristiti *Huge Pages*, potrebno je "*samo*" sve znati konfigurirati, što ponekad (zapravo često) nije jednostavno.
Postoji i rješenje koje je negdje između, a zove se: ***Transparent Huge Pages***.



Vezano za ***Transparent Huge Pages*** pogledajte poglavlje:
**12.3.4.1. Transparent Huge Pages.**



Vezano za ***Huge Pages*** pogledajte poglavlje:
**12.3.4.2. Huge Pages.**

**Izvori informacija: (652),(653),(856),(1067),(K-4),(K-5).**

#### 12.3.4.1. Transparent Huge Pages

Zbog želje da se ***Huge pages*** način alociranja memorije napravi što jednostavnijim, te kako bi se izbjegla potreba za posebnom konfiguracijom, ali i posebnim razvojem programa koji uopće znaju koristiti *Huge pages*, razvijen je sustav koji koristi tzv. ***Transparent Huge Pages***. Ova metoda koristi kernel proces (*kernel thread*) koji se zove `khugepaged` koji u pozadini, svako malo, pokušava pronaći dijelove RAM memorije (u kontinuitetu/slijedu) koji bi se mogli koristiti kao *Huge Pages*.
Potom ih počinje dodjeljivati procesima (programima/aplikacijama) koje bi inače u normalnom radu koristile normalne *stranice* memorije od 4kB, a ne one koje zovemo *Huge pages*, dakle one od 2MB.

Ovdje sa strane sâmog procesa/programa/aplikacije nisu potrebne nikakve modifikacije u sâmom programskom kôdu, već je cijeli ovaj proces transparentan, a odatle i naziv *Transparent Huge Pages* tj. **THP**.



**THP** je moguće koristiti samo za tzv. *Anoymous* regije memorije, a u budućim inačicama kernela se najavljuje i upotreba **THP** za `tmpfs` (poseban datotečni sustav u memoriji) i `Page Cache` vrstu memorije.

Dakle upotreba *Anoymous* regija memorije je slična kao i upotreba *normalnih* dijelova memorije, uz neke specifičnosti o kojima nećemo dublje raspravljati.
Zanimljivo za znati je sâmo to kako postoji nekoliko metoda pomoću kojih je moguće pristupiti memoriji pomoću metode preslikavanja datoteka odnosno takozvane *„**File mapping**“* metode, a to su:

- Privatno odnosno izolirano preslikavanje (mapiranje) datoteka *(Engl. Private file mapping).*
- Dijeljeno preslikavanje (mapiranje) datoteka *(Engl. Shared file mapping).*

*File mapping* metode predstavljaju datoteke na disku koje simboliziraju određenu količinu podataka (bajta) preslikanih u RAM memoriji. Dakle ove metode podrazumijevaju posebnu datoteku (*file deskriptor*) na koju se može referencirati neki uređaj (*device*), objekt u dijeljenoj memoriji (*shared memory object*) ili neki drugi resurs kojemu pristupamo preko *file deskriptor*a.
Pri tome je procedura sljedeća:

1. Otvaranje datoteke (*open*) kako bi se dobio *File deskriptor*.
2. Operacija u kojoj se treba dobiti veličina memorije za korištenje (`fstat`) pomoću tôg *File deskriptor*a.
3. Alociranje datoteke (`mmap`), korištenjem *File deskriptor*a koji smo dobili u prvom koraku pomoću operacije *open*.
4. Kasnije slijedi zatvaranje (*close*) tog *File deskriptor*a.
5. Te konačno dolazi na red i rad s ovom *File mapped* datotekom.

Kada se datoteka preslikava kao „*Private*“, promijene na njoj se ne propuštaju prema dolje (*na sloj datotečnog sustava*) već je sve što se radi, radi u RAM memoriji, i to na kopiji ove datoteke. Kada se datoteka preslikava kao „*Shared*“, tada se sve promijene u njoj (u RAM memoriji) prosljeđuju prema diskovnom I/O sustavu i disku, i to automatski od strane kernela.

U cijeloj ovoj proceduri kada pričamo o datotekama koje su preslikane (mapirane) u RAM memoriju, govorimo o direktnom radu s RAM memorijom koji indirektno može ili ne mora završiti na disku. Sjetimo se da je u UNIX ili Linux svijetu sve (i uređaji/hardver) neka datoteka, pa će nam sada postati još bliža ova uzrečica o Unixu odnosno Linuxu i datotekama.

Naredba `lsof` daje nam listu svih otvorenih datoteka na sustavu. Mi ćemo ju sada ograničiti sâmo na popis svih otvorenih datoteka koje je otvorio proces odnosno servis `sshd`, a on je servis koji je zadužen za udaljeni pristup na naše računalo, preko SSH protokola (TCP port 22). Stoga ga pogledajmo s novom naredbom: `lsof`

```
lsof -c sshd

COMMAND PID USER   FD   TYPE   DEVICE SIZE/OFF  NODE NAME
sshd   1073 root  cwd    DIR   253,0      4096     2 /
sshd   1073 root  rtd    DIR   253,0      4096     2 /
sshd   1073 root  txt    REG   253,0    569280 12282 /usr/sbin/sshd
sshd   1073 root  mem    REG   253,0     12792  4283 /lib/libutil-2.12.so
sshd   1073 root  mem    REG   253,0    113912  4254 /lib/libnsl-2.12.so

sshd   1073 root    2u   CHR     1,3       0t0  3960 /dev/null
sshd   1073 root    3u  IPv4  8651         0t0   TCP *:ssh (LISTEN)
sshd   1073 root    4u  IPv6  8653         0t0   TCP *:ssh (LISTEN

sshd   1221 root  cwd    DIR   253,0      4096     2 /
sshd   1221 root  rtd    DIR   253,0      4096     2 /
sshd   1221 root  txt    REG   253,0    569280 12282 /usr/sbin/sshd

sshd   1221 root   3r   IPv4  9751         0t0   TCP  192.168.100.68:
                                           ssh->192.168.100.254:2488(ESTABLISHED)
sshd   1221 root   4w   unix 0xdf8e5040 0t0   9799  socket
sshd   1221 root   5u   FIFO   0,8       0t0   9802  pipe
```

Iz ispisa vidimo kako su učitane razne biblioteke, vidljive kao obične datoteke poput `/lib/libnsl-2.12.so`.
Nakon toga su otvorene datoteke (stupac TYPE je `IPv4` i `IPv6`) kojima pod NODE stoji `TCP`, što znači da su otvorene posebne datoteke koje predstavljaju otvoreni TCP port na lokalnom računalu "`*:ssh`".

To znači TCP port 22 (***ssh*** protokol radi na *TCP portu 22*, koji je odmah prepoznat pa je prikazan kao "*:ssh").
Potom slijedi nîz datoteka koje predstavljaju nekoliko trenutnih radnih direktorija (TYPE je DIR). Osim njih učitana je i datoteka čiji je **TYPE=IPv4** uz koju vidimo i dvije IP adrese.
IP adresa: *192.168.100.68* je naša IP a *192.168.100.254* je IP adresa računala s kojeg smo se spojili preko ***ssh*** protokola.

Dakle ova datoteka čiji **TYPE** je **IPv4**, a pod **NAME** vidimo dvije IP adrese (to je TCP/IP veza između te dvije IP adrese) predstavlja tzv. *IP Socket* odnosno mehanizam koji omogućava komunikaciju između procesa (programa) preko mreže.
Na kraju, tu je i datoteka koja predstavlja *UNIX Socket* odnosno komunikacijski kanal kroz koji programi mogu komunicirati. Dakle *UNIX socket* koristi **IPC** (*Inter Process Communication*) mehanizam koji omogućava dvosmjernu (*bidirekcionalnu*) komunikaciju odnosno razmjenu podataka između više procesa.
U zadnjim nîzu koji smo prikazali (izlistanje je filtrirano sâmo na osnovne nama zanimljive dijelove) vidimo datoteke koje predstavljaju mrežne konekcije. Sada je vjerujemo jasno zbog čega stalno govorimo da je u *UNIXu* ili *Linuxu* sve (što radimo) zapravo datoteka.
Konkretno ovo su datoteke koje otvara sâmo SSH *servis*, ali vidljivo je što se događa u pozadini te postaje jasnija veza prema *File Mapping* metodi mapiranja memorije koja završava ili ne završava na datotekama, koje u konačnici ne moraju biti pohranjene na (fizičkom) disku. Sjetimo se da postoje i posebne vrste datoteka koje se zapravo nalaze u RAM memoriji. Pod stupcem DEVICE su navedeni uređaji na kojima završavaju (na kojima se nalaze) ove datoteke. Primjerice u našem slučaju " 253, 0 " (***Major*** i ***Minor*** brojevi uređaja) predstavlja naš tvrdi disk (/dev/sda), a ostali brojevi i oznake označavaju posebne uređaje za koje smo rekli kako se u konačnici niti ne moraju nalaziti na stvarnom tvrdom disku, već negdje u RAM memoriji: *direktno* ili *indirektno*.

Pogledajmo sada posebne otvorene datoteke koje predstavljaju otvorene ili ostvarene mrežne konekcije.
Za ovu namjenu koristimo naredbu lsof na sljedeći način:

```
lsof -i

COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd     1073   root    3u  IPv4   8651    0t0   TCP *:ssh (LISTEN)
sshd     1073   root    4u  IPv6   8653    0t0   TCP *:ssh (LISTEN)
httpd    1095   root    4u  IPv6   8715    0t0   TCP *:http (LISTEN)
httpd    1098 apache    4u  IPv6   8715    0t0   TCP *:http (LISTEN)
httpd    1099 apache    4u  IPv6   8715    0t0   TCP *:http (LISTEN)
sshd     1221   root    3r  IPv4   9751    0t0   TCP 192.168.100.68:ssh-
>192.168.100.254:2488 (ESTABLISHED)
```

Dakle vidimo otvorene ssh (*TCP port 22*) i http (*TCP port 80*) portove te jednu ostvarenu konekciju; zadnja u nîzu s vidljivim IP adresama. I pogledajmo još jedan primjer iz mreža, preko otvorenih posebnih datoteka.

Saznajmo tko je otvorio posebne datoteke koje predstavljaju otvorene mrežne konekcije na portu 22 (to je *ssh* servis):

```
lsof -i :22

COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd     1073 root   3u   IPv4   8651      0t0  TCP  *:ssh (LISTEN)
sshd     1073 root   4u   IPv6   8653      0t0  TCP  *:ssh (LISTEN)
sshd     1221 root   3r   IPv4   9751      0t0  TCP  192.168.100.68:ssh-
>192.168.100.254:2488 (ESTABLISHED)
```

Iz ovoga je vidljivo da je proces sshd (COMMAND stupac) taj koji je otvorio **TCP port 22 (**ssh**)**.

**Vratimo se na preslikavanje (mapiranje) memorije.**
Upoznajmo se i sa zadnjom metodom preslikavanja (mapiranja) memorije; **Anonymous mapping** metodama, koje mogu biti:

- *Privatno anonimno preslikavanje (mapiranje) (Engl. Private Anonymous mapping).*
- *Dijeljeno anonimno preslikavanje (mapiranje) (Engl. Shared Anonymous mapping).*

*Anonymous mappings* način preslikavanja RAM memorije s druge strane nema nikakve veze s datotekama na disku, niti *File deskriptorima*. Dakle ovo je samo druga metoda alociranja RAM memorije preko *sustava* (menadžmenta) *virtualne memorije*.

U *RedHat/CentOS* distribucijama Linuxa od inačice 6.x ***THP*** je uglavnom automatski u upotrebi. Kako bismo provjerili je li **THP** aktivan pogledajmo sadržaj datoteke: /sys/kernel/mm/transparent_hugepage/enabled.

```
cat /sys/kernel/mm/transparent_hugepage/enabled

[always] madvise never
```

Ako je označeno s [always] ili [madvise] tada je **THP** aktivan, a u suprotnom, ako imamo [never], tada nije.

Naime čim je jedan od gore navedena dva parametra uključen, kernel automatski pokreće proces khugepaged koji pronalazi memorijske lokacije na kojima bi se mogle koristiti *Huge Pages* stranice, te ih pokušava propagirati procesima/programima odnosno aplikacijama.

Kako bismo vidjeli koliko je anonimnih preslikavanja (Engl. **Anonymous mappings**) trenutno alocirano od strane khugepaged, pogledajmo datoteku: /proc/meminfo.

Pri ispisu ćemo filtrirati sâmo red koji nas zanima (`AnonHugePages`):

```
grep AnonHugePages /proc/meminfo

AnonHugePages:    444416 kB
```

Dakle u upotrebi je `444416` kB tj. oko 434 MB RAM-a za *Anonymous Huge Pages* odnosno kao **THP**.

Pogledajmo i drugu statistiku iz datoteke: `/proc/vmstat`:

```
egrep 'trans|thp' /proc/vmstat

nr_anon_transparent_hugepages 216
thp_fault_alloc 248
thp_fault_fallback 0
thp_collapse_alloc 235
thp_collapse_alloc_failed 0
thp_split 61
```

Ovdje vidimo kako se određeni broj stranica memorije koristi kao *Anonymous* tj. **THP**.
Bacimo još sâmo jedan pogled na `khugepaged` servis (*daemon*) koji je zaslužan za ovaj sav rad oko **THP**-a.

```
ps -ef | egrep "UID|khugepaged" | grep -v grep

UID        PID  PPID  C STIME TTY          TIME CMD
root        64     2  0 Nov22 ?        00:00:00 [khugepaged]
```

**Što je ovdje zanimljivo?**
`PPID` - *Parent **PID*** tj. roditeljski proces koji je pokrenuo `khugepaged` je **2**. `PPID` broj **2** je `kthreadd` proces koji je zadužen za pokretanje *kernel thread* procesa (pogledajte poglavlje o procesima: **9.**).



Vezano procese pogledajte poglavlje :
**9. Proces menadžment.**

Sjetimo se da je ***PID 1*** od `init` procesa koji pokreće sve procese i servise. S druge strane ***PID 2*** je ***PID*** `kthreadd` kernel procesa koji pokreće posebne sistemske *thread* procese.

To su procesi koji se moraju izvršavati na najnižoj razini sustava, koji se nalaze u tzv. *kernel space* adresnom prostoru (rezervirani i izolirani memorijski adresni prostor za kernel), a koji na najbrži mogući način komuniciraju s kernelom.

Sada pogledajmo sve procese na sustavu koji koriste THP memoriju, a želimo vidjeti za svaki proces i koliko memorije se koristi kao *THP*.

U primjeru je nîz naredbi koje treba pozvati u jednom redu (ispis je skraćen zbog lakšeg razumijevanja):

```
grep -e AnonHugePages  /proc/*/smaps | awk  '{ if($2>4) print $0} ' |  awk -F "/"
'{print $0; system("ps -fp " $3)}'

UID       PID PPID  C STIME TTY TIME     CMD
root     1193    1  0 Nov22 ?   00:00:00 s3fs       /proc/1275/smaps:AnonHugePages:   2048 kB
root     1275    1  0 Nov22 ?   00:00:00 automount  /proc/1459/smaps:AnonHugePages:   2048 kB
haproxy 1459    1  0 Nov22 ?   00:00:03 haproxy    /proc/1571/smaps:AnonHugePages:  63488 kB
nxuser   1571 1556  2 Nov22 ?   00:12:23 java       /proc/1571/smaps:AnonHugePages:  26624 kB
```

Dakle upotreba **THP** je dobra u određenim slučajevima, ali je lošija za sve programe koji koriste imalo više memorije zbog dinamičke alokacije i stalnog "vršljanja" po RAM memoriji od strane kernel procesa `khugepaged`.

Druga opcija je prebacivanje **THP** u [`madvise`] način rada koji je manje agresivan za sustav, i koji je stoga u nekim slučajevima **(1067)** bolji izbor. Međutim u određenom postotku slučajeva uopće ne želimo koristiti **THP**.

Kako isključiti **THP** i kako umjesto **THP** koristiti statički ***Huge Pages*** način rada, pogledajmo u sljedećem poglavlju.

**Izvori informacija: (652),(653),(654),(655),(856),(1067),(K-4),(K-5)**,`man lsof`,`man 5 proc`.

### 12.3.4.2. Huge Pages

***HugePages*** je značajka memorijskog menadžmenta unutar operativnog sustava Linux koja omogućava korištenje velikih stranica memorije, što je korisno za aplikacije koje zahtijevaju velike količine memorije ili imaju intenzivan pristup memoriji. Naime. standardne stranice memorije na Linuxu obično su veličine 4KB, dok upotrebom *HugePages* možemo koristiti stranice memorije od 2MB ili čak 1GB, ovisno o konfiguraciji i podršci sustava odnosno procesora. Koje su sve prednosti upotrebe ***HugePages***:

- Smanjenje preopterećenja TLB memorije – manji broj velikih stranica memorije smanjuje broj TLB unosa, što smanjuje broj dohvaćanja stranica u TLB te TLB promašaja. Samim time se ubrzava pristup memoriji.
- Poboljšanje performansi za određene primjene – *HugePages* je koristan za aplikacije poput baza podataka (npr. *Oracle Database*, *PostgreSQL*), drugih aplikacija koje intenzivnije koriste veće količine RAM memorije i virtualizacijskih rješenja koja intenzivno koriste memoriju. Korištenjem velikih stranica memorije smanjuje se broj zahtjeva za rad s memorijom (zapisivanje/čitanje iz RAM memorije).

Primjerice kada koristimo *HugePages* od 2MB umjesto standardnih stranica od 4kB za upravljanje memorijom, dobivamo znatne uštede u broju stranica memorije te smanjenje *Page tablice* (`PageTables`) u RAM memoriji. Samim time i smanjenje potrebnog broja TLB unosa u TLB međumemoriji procesora. To znači da je za adresiranje (i korištenje) iste količine RAM memorije potrebno 512 puta manje stranica memorije (**2MB** vs **4kB** za svaku stranicu memorije).

U ovom primjeru ne želimo koristiti **THP** već *Huge Pages*, pa krenimo s ručnim radom. Međutim prvo moramo imati podršku unutar našeg procesora (CPUa) za ovakve povećane stranice memorije (zastavica/flag **PSE**), što možemo provjeriti sa:

```
grep pse /proc/cpuinfo
```

Dakle zastavica `pse` odnosno ova funkcionalnost mora postojati unutar našeg procesora (*CPU*) kako bi to uopće radilo.
Ako je sve u redu imamo ovu podršku, možemo trenutno onemogućiti upotrebu **THP**a:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Nakon nekoliko trenutaka će sustav u pozadini shvatiti da ne želimo koristiti **THP** te će kernel nît (*thread)* proces `khugepaged` biti zaustavljen.

```
ps -ef | egrep "UID|khugepaged" | grep -v grep

UID         PID   PPID  C STIME TTY          TIME CMD
```

Ovdje vidimo da proces `khugepaged` stvarno više ne postoji (jer nije izlistan u popisu pokrenutih procesa/programa).
Ali svi procesi koji su koristili *THP* i dalje rade i koriste *THP* jer su već alocirali *THP* memoriju pomoću procesa `khugepaged`.
Sada imamo nekoliko opcija:

1. Pronaći sve procese koji koriste *THP*, zaustaviti ih te ponovno pokrenuti.
2. Napraviti trajnu promjenu ***GRUB***-a kako bi se *THP* isključio od trenutka podizanja operativnog sustava.
3. Napraviti trajnu promjenu ***GRUB***-a u kojoj će se uz *THP* alocirati i definirani broj stranica kao *Huge Pages*.
4. Napraviti skriptu u kojoj će se isključivati upotreba *THPa*, nakon podizanja operativnog sustava. Ovo bi moglo zahtijevati naknadne promjene kao pod točkom **1.** jer ovisi kojim redoslijedom će se izvršiti naša skripta. Točnije hoće li se prije ili poslije naše skripte pokretati određeni servisi ili programi.

**Opcija 1**
Nađimo sve procese koji trenutno koriste *THP* sa sljedećim nîzom naredbi (ovo je nîz naredbi u jednom redu) (ispis je skraćen):

```
grep -e AnonHugePages  /proc/*/smaps | awk  '{ if($2>4) print $0} ' | awk -F "/"
'{print $0; system("ps -fp " $3)}'
```

```
UID         PID  PPID  C STIME TTY      TIME  CMD
root       1193     1  0 Nov22 ?    00:00:00 s3fs      /proc/1275/smaps:AnonHugePages:    2048 kB
root       1275     1  0 Nov22 ?    00:00:00 automount /proc/1459/smaps:AnonHugePages:    2048 kB
haproxy    1459     1  0 Nov22 ?    00:00:03 haproxy   /proc/1571/smaps:AnonHugePages:   63488 kB
nxuser     1571  1556  2 Nov22 ?    00:12:23 java      /proc/1571/smaps:AnonHugePages:   26624 kB
```

Zapravo su to svi oni procesi koji su i prije koristili *THP*. Možemo ih restartati jedan po jedan, ali moramo znati koji prije kojega i na koji način. Sada nećemo objašnjavati kako, već idemo na opciju **2.**

**Opcija 2 (isključivanje THP u kernelu)**
Napravimo trajnu promjenu (OPREZNO). Ovaj korak je vrlo kritičan jer krivim promjenama možemo imati problema sa pokretanjem operativnog sustava kod sljedećeg pokretanja računala. U ovom slučaju možemo uređivati datoteku `/etc/grub.conf` ili dodati željenu kernel boot opciju.



Za detalje oko konfiguracije opcija ***GRUB***-a, pogledajte **poglavlje:**
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

Zbog jednostavnosti, mi ćemo sada samo dodati željenu opciju (`transparent_hugepage=never`):

```
grubby --update-kernel=ALL --args=transparent_hugepage=never
```

A potom je potrebno regenerirati *GRUB2*:

```
grub2-mkconfig
```

I sada restartajmo cijeli sustav.
Ovime smo rekli kernelu da tijekom podizanja sustava NE koristi ***THP***.

Sada pogledajmo statistike:

```
grep AnonHugePages /proc/meminfo
AnonHugePages:          0 kB
```

Zatim pogledajmo i druge statistike:

```
egrep 'trans|thp' /proc/vmstat
nr_anon_transparent_hugepages 0
thp_fault_alloc 0
thp_fault_fallback 0
thp_collapse_alloc 0
thp_collapse_alloc_failed 0
thp_split 0
```

Sada još provjerimo je li nekim čudom zaostao neki proces koji ipak koristi *THP* (*navedeni nîz naredbi je u jednom retku*):

```
grep -e AnonHugePages  /proc/*/smaps | awk  '{ if($2>4) print $0} ' |  awk -F "/"
'{print $0; system("ps -fp " $3)}'
```

Nema odgovora, odnosno pronađenih procesa (pokrenutih programa), što znači da je sve u redu.
Jasno je vidljivo da ***THP*** više nije u upotrebi, konačno kako smo i planirali.

### Opcija 3 (THP + Huge Pages)

Uz ***THP***, fiksno rezervirajmo velike stranice memorije (*Huge Pages*) od 2MB, tijekom pokretanja sustava, također dodavanjem opcija u konfiguraciju *GRUB-a*. Stoga ćemo otvoriti datoteku `/etc/grub.conf` te u dio koji smo mijenjali u „***Opciji 2***“, za kernel za koji to radimo, u retku odnosno liniji koja počinje sa: `kernel /boot/vmlinuz…` na kraj ćemo nadodati i sljedeće:

```
hugepagesz=2M hugepages=512
```

Iste opcije, ali na jednostavniji način možemo postići s programom `grubby`.



Za detalje pogledajte poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

Kod novijih inačica sustava (***RedHat/CentOS* 7+**) ove opcije se navode nakon ključne riječi: **`args=""`** (unutar navodnika). Navedene opcije znače da omogućujemo upotrebu *Huge Pages* stranica od 2MB i to njih 512 komada, što je ukupna rezervacija 1GB RAM memorije za ovu namjenu. Nakon toga moramo konfigurirati željene aplikacije da ju koriste; preskočite početak cjeline: „***Opcije 4***“ te pogledajte nastavak, kako to napraviti.



**Za upotrebu stranica memorije od 1GB (ako to vaš CPU podržava), moguće je koristiti *GRUB* opcije:** `default_hugepagesz=1GB hugepagesz=1G hugepages=4` **što će ih alocirati ukupno 4GB (**`4` **x 1GB).**

U slučaju kada ne želimo fiksirati broj ***HugePages*** stranica memorije u *GRUB-u* odnosno tijekom pokretanja sustava (Linuxa), tada možemo izostaviti opcije: `hugepagesz=2M hugepages=XY` te njihovu veličinu možemo definirati sâmo upotrebom ***sysctl*** varijable: `vm.nr_hugepages` kako ćete vidjeti u sljedećoj opciji, to jest opciji: "**Opcija 4**“.

### Opcija 4 (isključen THP, samo Huge Pages)

Isključimo ***THP*** u skripti, koja se podiže automatski s podizanjem našeg Linuxa. U ovo primjeru koristit ćemo skriptu: `/etc/rc.local`. Dakle dodajmo sljedeći redak (liniju), na kraj datoteke `/etc/rc.local` i to ovako:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Ova skripta se pokreće zadnja, tijekom pokretanja sustava, tako da je velika vjerojatnost kako bi se mnogi servisi koji koriste ***THP***, već podignuli i upotrebljavali *THP*. Tako da ovo vjerojatno nije najbolje mjesto za ovakve promijene.
Stoga ovo promatrajte sâmo kao primjer.

Tek sada idemo dalje s upotrebom *Huge Pages*-a. Provjerimo koristi li se *Huge Pages*:

```
grep -i HugePages_Total /proc/meminfo
HugePages_Total:        0
```

Dakle *Huge Pages* nije u upotrebi.

Pošto se *Huge Pages* koristi statički, moramo prvo odlučiti koliko *Huge Pages* stranica (od 2MB) želimo statički alocirati za naše procese/programe/aplikacije koje su pisane odnosno programirane tako da ih uopće znaju i mogu koristiti.

Recimo da imamo poslužitelj sa 8GB RAM, od kojih želimo 6GB dodijeliti na korištenje preko *Huge Pages*a.

***Računica: 6GB = 6144 MB / 2MB stranice = 3072 stranica***.

To znači da želimo alocirati 3.072 stranica za *Huge Pages*.
Otvorimo datoteku: `/etc/sysctl.conf` i dodat ćemo novu liniju (redak) koja će sadržavati sljedeće:

```
vm.nr_hugepages=3072
```

Snimimo ovu datoteku te pokrenimo ponovno postavljanje sustava na osnovu naše nove opcije, pomoću naredbe `sysctl`:

```
sysctl -p
```

Sada provjerimo je li to stvarno učitano:

```
sysctl vm.nr_hugepages

vm.nr_hugepages = 3072
```

Vidimo da je učitano, jer imamo postavljenu novu vrijednost `3072`. Provjerimo i statistike sustava:

```
grep -i HugePages_Total /proc/meminfo

HugePages_Total:    3072
```

Te provjerimo koliko je *HugePages* u upotrebi odnosno koliko je slobodno:

```
grep -i HugePages /proc/meminfo

AnonHugePages:         0 kB
HugePages_Total:    3072
HugePages_Free:     3072
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

Dakle sve što smo alocirali (`3072`) je i dostupno.

Vidimo i informaciju da je veličina *HugePages* stranica 2048kB tj. 2MB. Sada pogledajmo trenutnu veličinu *Page* tablice:

```
grep -i PageTables /proc/meminfo

PageTables:       164068 kB
```

Vidimo da je tablica veličine 160 MB za 4kB stranice (to je negdje za oko 41.000 stranica od 4kB). Dakle nije sva memorija niti alocirana. Ponovno ćemo pogledati ovu tablicu nakon upotrebe *Huge Pages*a.
Sada je potrebno optimizirati i druge parametre memorije da bi se *HugePages* stranice uopće mogle koristiti.

## Tko ima pravo koristiti *Huge Pages*?

Standardno samo `root` korisnička grupa (***GID***=0) ima ova prava, ali pogledajmo još jednom imamo li vrijednost `0` ili:

```
cat /proc/sys/vm/hugetlb_shm_group

0
```

Ako imamo potrebu da ih koriste korisnici koji su u nekoj dugoj korisničkoj grupi, to je potrebno i konfigurirati. Ako recimo imamo grupu čiji Group ID (***GID***) je `1500`, tada moramo dodati novu opciju (redak) u `/etc/sysctl.conf` kako slijedi:

```
vm.hugetlb_shm_group=1500
```

Ponovno učitajmo ovu novu konfiguraciju:

```
sysctl -p
```

Zamislimo da je korisnik koji je u ovoj grupi ***GID***=1500 imena "*hugetable*". Za tog korisnika je potrebno i definirati koja su mu memorijska ograničenja tj. koja je maksimalna količina RAM memorije koju mu dozvoljavamo da može koristiti.
Ova ograničenja se definiraju u datoteci: `/etc/security/limits.conf`.

Dodajmo sljedeća dva retka, s kojim ćemo ograničiti korisnika na maksimalno 6GB RAM memorije kao *Huge pages*:

```
hugetable hard memlock 6291456
hugetable soft memlock 6291456
```

Napomena: `6.291.456` = 6GB RAM.

Sada se logirajmo na sustav kao korisnik "*hugetable*" i provjerimo ograničenja (pomoću naredbe `ulimit`):

```
ulimit -l

6291456
```

Dakle ograničenja koja smo postavili su dobra.
Provjerite jeste li dobro konfigurirali ostale parametre memorije iz prijašnjih poglavlja: `kernel.shmall` i `kernel.shmmax`

**Teži dio**
Sada je potrebno konfigurirati (ili napisati) program/aplikaciju koja zna koristiti *Huge Pages*. U našem slučaju koristili smo bazu podataka, koju je bilo potrebno sâmo rekonfigurirati i pokrenuti kako bi znala koristiti *Huge Pages*. Dakle koristimo SQL bazu podataka koja zna raditi s *Huge Pages stranicama*. Konfiguraciju SQL baze podataka nećemo objašnjavati; malo se potrudite te istražite kako se konfigurira ili kompilira vaš program, kako bi mogao koristiti *Huge Pages* stranice memorije.

U slučaju drugih programa potrebne su njihove optimizacije ili omogućavanje upotrebe *Huge Pages* rada, poput primjerice programskog jezika ***Java***, u kojemu, kada pokrećemo *Java* program, moramo ga pokrenuti minimalno sa sljedećim opcijama:

- `-XX:+UseLargePages` – s ovime uključujemo podršku za *Huge Pages* za ***Java*** program koji pokrećemo.
- `-XX:LargePageSizeInBytes=2m` – s ovime definiramo da je veličina *Huge Page* stranica 2MB.

Vratimo se na naš primjer sa SQL bazom. Pokrenimo ju te pričekajmo desetak minuta da naša baza alocira svu potrebnu memoriju. Potom pogledajmo statistike za *Huge Pages* stranice memorije.

```
grep -i HugePages /proc/meminfo

AnonHugePages:         0 kB
HugePages_Total:    3072
HugePages_Free:        2
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

Vidimo kako je od iskoristivih 3072 *Huge Pages* iskorištena većina te su ostala samo dva (`2`) slobodna. Ovo znači da smo uspješno konfigurirali i sustav i aplikaciju da koriste *Huge Pages* odnosno da umjesto standardne veličine od 4kB naša željena aplikacija koristi stranice od 2MB.

S ovime smo drastično ubrzali operacije koje koriste RAM memoriju jer smo veličinu stranica memorije povećali 512 puta i isto toliko puta smanjili *Page* tablicu.

Pogledajmo i *Page* tablicu, koja je bila veličine 160MB za 4kB stranice:

```
grep -i PageTables /proc/meminfo

PageTables:        39924 kB
```

Sada je *Page* tablica oko 40MB, što znači kako se tablica poprilično smanjila, a alocirana je puno veća količina RAM memorije nego prije. Dio tablice je i dalje za 4kB stranice, a drugi dio je za nove 2MB (2048 kB) stranice.
Postigli smo što smo htjeli.

Kako smo već prije spomenuli, veličina stranice sustava virtualne memorije ovisi o sâmoj arhitekturi procesora i naravno podršci operativnog sustava, pa tako imamo sljedeće veličine stranica virtualne memorije:

| Arhitektura procesora | Standardna veličina stranice memorije | Koje veće stranice memorije sustav podržava |
|---|---|---|
| 32 bitni x86 | 4 kB | 4MB u **PSE*** načinu rada<br>2MB u **PAE**** načinu rada |
| 64 bitni x86 (x86-64) | 4 kB | **2MB** [`pse` – CPU zastavica]<br>**1GB** [`pdpe1gb` – CPU zastavica] |
| ARM v7 | 4 kB | **64 kB** ili **1MB** ili **16MB** |

**CPU** zastavice koje podržava naš procesor (CPU) možemo vidjeti unutar datoteke: `/proc/cpuifo`.
***PSE** (*Page Size Extension*) označava dodatnu funkcionalnost procesora koja omogućava stranice memorije veće od 4kB.
****PAE** (Physical Address Extension) označava dodatnu funkcionalnost procesora koja omogućava direktno adresiranje memorije preko granice od 4GB (za 32 bitni sustav) ali i stranice memorije veće od 4kB.

---

**Za RedHat/Centos7+:** Kako bi aplikacije uopće mogle koristiti *HugePages*, one se oslanjaju na poseban datotečni sustav ***hugetlbfs*** povezan s posebnim uređajem (`/dev/hugepages`) koji predstavlja ove stranice memorije. Međutim da bi to sve uopće radilo, kernel mora biti kompiliran s opcijom: **CONFIG_HUGETLBFS**, što možete provjeriti sa sljedećom naredbom:

```
grep CONFIG_HUGETLBFS /boot/config-`uname -r`
```

Čim ste aktivirali *HugePages* stranice, montira se sljedeća točka montiranja, bez potrebe za ručnim dodavanjem u datoteku: `/etc/fstab` (kako je bilo na starijim inačicama ***RedHat/CentOS*** Linuxa **6.x**). Dakle vidjet ćete ju kao (s `mount`):

```
hugetlbfs on /dev/hugepages type hugetlbfs
```



Statistike sustava virtualne memorije se zapisuju u posebnu datoteku: `/proc/vmstat`[(857)] i u `/proc/meminfo`.

**Izvori informacija: (656),(657),(658),(659),(660),(857),(1765)**,`man sysctl`,`man ulimit`,`man 5 proc`.

## 12.3.5. Zone RAM memorije

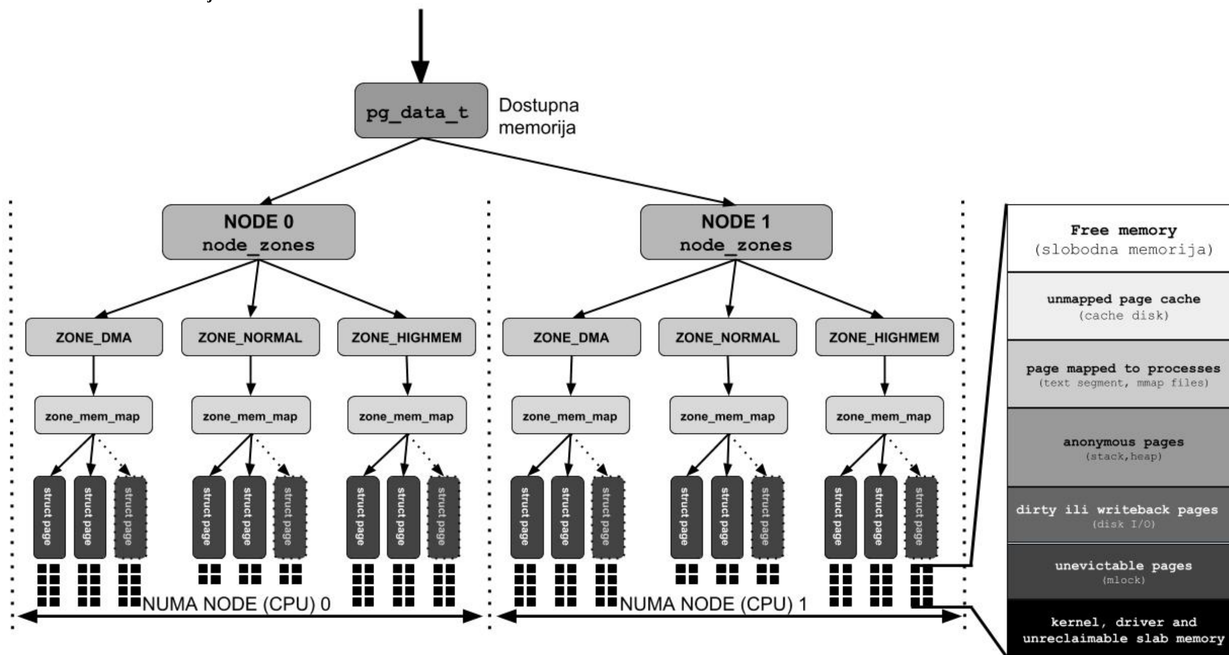
**Slijedi napredno poglavlje!**

Kako bismo posve razumjeli način na koji Linux koristi RAM memoriju, moramo razumjeti što su **zone** RAM memorije i čemu služe. Naime, Linux kernel ne promatra cijelu dostupnu RAM memoriju kao jednu cjelinu. Naprotiv, Linux kernel dijeli RAM memoriju u nekoliko regija, koje se zovu ***zone*** RAM memorije. Vrste *zona* RAM memorije ovise o tome je li računalo 32 bîtno ili 64 bîtno. Mi ćemo dalje govoriti o 64 bîtnom računalu odnosno Linuxu. Zone RAM memorije za 64 bitna računala, podijeljene su u sljedeće zone:

- `DMA` zonu, u kojoj se nalazi prvih 16 MB RAM memorije. Ova zona postoji iz povijesnih razloga, iz vremena kada je hardver koji koristi ***DMA*** pristup mogao pristupati samo do donjih 16 MB RAM memorije.
- `DMA32` zonu, koja se nalazi unutar prvih 4 GB RAM memorije i postoji stoga što određeni hardver može koristiti ***DMA*** pristup samo do prvih 4GB RAM memorije. Dakle uređaji koji rade 32 bitno mogu koristiti i ovu zonu memorije za ***DMA*** prijenos podataka.
- `Normal` zonu, koja koristi RAM memoriju od 4GB RAM-a na više. Dakle, 64 bitno Linux računalo koje ima sâmo primjerice 2 GB RAM, uopće nema dostupnu `Normal` zonu RAM memorije, dok računalo s 4GB RAM-a ima sâmo vrlo malo memorije u ovoj zoni.
- `HIGMEM` je zona koja označava „visoku“ memoriju koja nije stalno mapirana u adresni prostor kernela. Ova zona memorije je u nekim arhitekturama mapirana u svu dostupnu memoriju iznad 896MB, dok u većini arhitektura ona nije u upotrebi jer je sva dostupna memorija direktno mapirana i koristi se unutar zone `Normal`.

U normalnom radu kada neki program (aplikacija/proces) zatraži pristup RAM memoriji, obično joj se dodjeljuje RAM memorija iz zone `Normal` jer su druge zone memorije rezervirane za druge namjene (npr. za upotrebu ***DMA***). U slučajevima kada se normalno zatraži korištenje `Normal` zone, ali u njoj nema više slobodne memorije, Linux kernel će programu dozvoliti korištenje i inače rezerviranije zone `DMA32` ili `DMA` zone memorije u kojima ima mjesta za upotrebu. Dodatno na sustavima koji imaju **NUMA** arhitekturu, odnosno sustava koji imaju više fizičkih procesora (*CPU*), zone memorije su dodatno povezane s fizičkim procesorom odnosno njegovim memorijskim kontrolerom i njegovom pripadajućom RAM memorijom.
Ovo je vidljivo kao zone RAM memorije koja pripadaju određenom *node*-u a koji se asocira s fizičkim *CPU*om. Dakle, ovdje se prema osnovnim postavkama sustav brine kako bi pokrenuti program koristio RAM memoriju što bližu CPU jezgri fizičkog procesora, na kojemu se nalazi i memorijski kontroler RAM memorije koju program treba koristiti, kako bi sve radilo uz što manje usporavanja uzrokovanih eventualnim udaljenim dohvaćanjem RAM memorije. Pogledajte i poglavlje: **10.7.2.2 NUMA**.

**Slika 101. Zone RAM memorije.**




Kako je vidljivo na slici 101 dostupna memorija je dostupna tako da postoji razlika u tome pripada li fizička RAM memorija NUMA *node*-u (čvoru) nula (**0**), odnosno prvom fizičkom procesoru ili NUMA čvoru jedan (**1**) odnosno drugom fizičkom procesoru; ako imamo takav sustav s dva ili više fizičkih procesora. Ako imamo sustav sa sâmo jednim fizičkim procesorom, tada imamo praktično sve isto kao da imamo sâmo jedan NUMA čvor. RAM memorija koja pripada određenom NUMA čvoru je svaka cjelina memorije unutar pripadajućeg NUMA čvora koja je označena kao: `node_zone`. Ova cjelina memorije se dijeli na zone kako smo već objasnili (obično *DMA*, *NORMAL*, a negdje i *HIGHMEM*).
Unutar svake od zona memorije postoji polje memorije definirano kao `zone_mem_map`, a tek unutar njega se nalazi grupa stranica memorije (`struct page`) unutar koje se nalaze stranice memorije (*memory page),* vidljive kao:■ .
Ako pogledamo desni dio slike vidimo izgled stranica memorije (*memory page*) te kako su one podijeljene odnosno iskorištene i/ili označene.

Prisjetimo se da su stranice memorije najmanje *cjeline* memorije, standardne veličine 4kB!.

Pri tome neke od njih mogu biti slobodne (`free memory`), neke se mogu koristiti kao međumemorija prema diskovnom podsustavu (`unmapped page cache`) ili kao stranice memorije koje su vezane za sâme procese (programe/aplikacije). Odnosno koriste se kao memorija za te procese i to: (`text`) dio u kojem se nalazi izvršni programski kôd programa te (`mmap files`) odnosno memorijski mapirane datoteke koje su u ***byte-to-byte*** korelaciji s datotekama na virtualnom datotečnom sustavu (***VFS***).

Što znači kako se može raditi o preslikavanju (mapiranju) na klasične datoteke, *file deskriptore* ili druge slične objekte. Drugi dijelovi mogu biti:

- `Anonymous pages` – u koji se upisuju `stack` (programski stôg i *LIFO* struktura) i `heap` (dio memorije koji se može dijeliti među nîtima programa, dijeljenim bibliotekama i dinamički učitanim modulima unutar procesa).
- `Dirty` ili `writeback pages` – se obično koriste za diskovne ulazno izlazne operacije (disk I/O), a označavaju stranice memorije koje se mogu snimiti na disk, kako bi se memorija oslobodila.

U svakom slučaju, sve vrste stranica memorije o kojima smo do sada govorili, mogu se čistiti odnosno osloboditi, ako i kada je potrebno.
A sada dolazimo do dvije vrste stranica memorije koje se ne mogu osloboditi ili očistiti u svrhu oslobađanja memorije za rad korisničkih programa. To su:

- `Unevictable` te `kernel, driver (kernel modules) pages` (stranice) - su one stranice memorije koje se ne mogu koristiti od strane sustava za optimizaciju memorije. Isto tako i dio koji smo označili kao `unevictable` je dio procesa (programa) koji je zaključan (***mlock***), što znači kako je to dio procesa unutar RAM memorije koji se ne može nikako prebaciti u *SWAP* memoriju jer je to obično krucijalni dio programa koji mora biti rezidentan u RAM memoriji zbog načina rada sustava i programa, ali i zadržavanja brzine rada programa.

**Vratimo se na zone memorije**
Statistike zona RAM memorije su dostupne odnosno zapisuju se u sljedeće datoteke:

- `/proc/pagetypeinfo` - ovdje su opće statistike upotrebe zona RAM memorije.
- `/proc/zoneinfo` - ovdje su brojači koji pokazuju trenutno stanje zauzeća RAM memorije, po zonama.
- `/proc/buddyinfo` - ovdje je sumarizirana statistika upotrebe zona RAM memorije.

Pogledajmo kako to izgleda za jedan NUMA poslužitelj sa dva fizička procesora (*NUMA* CPU **node0** i **node1**) i 128 GB RAM. Pri tome je 64 GB RAMa na prvom fizičkom CPU (NUMA node0) i 64 GB RAMa na drugom fizičkom CPU (NUMA node1):

```
cat /proc/buddyinfo

Node 0, zone      DMA      2      2      1      1     2   1   0    0    1    1    3
Node 0, zone    DMA32 110974  43884  11989   2026   346   9   0    0    0    1    0
Node 0, zone   Normal 344428 384766  78046    695    19   3   0    0    0    0    1
Node 1, zone   Normal 487961 330878 108763  12006  1279   4   0    0    0    1    1
```

Ovdje je vidljivo kako je RAM memorija na NUMA sustavu raspodijeljena na sljedeći način:

- Prvih 64 GB RAM memorije preko prvog fizičkog procesora odnosno *CPUa* (*node 0*).
- Drugih 64 GB RAM-a preko drugog fizičkog procesora odnosno *CPUa* (*node 1*) što čini ukupno 128GB RAM.

Dakle prvi fizički *CPU* - **Node 0** prvi alocira RAM memoriju pa je na njemu alocirana RAM memorija od 0 do 64 GB, a onda se od 64 GB do 128GB alocira preko drugog fizičkog *CPUa* (*node1*).
To isto znači kako se na *node 0* procesoru (prvi fizički procesor) nalaze sljedeće zone memorije:

- `DMA`
- `DMA32` te prvi dio od slijedećeg djela.
- `Normal` što je od 4GB na dalje.

Potom se ostatak `Normal` zone proteže preko drugog fizičkog procesora tj. preko njegovog RAM kontrolera i pripadajuće RAM memorije; od 64 GB RAM-a do 128GB RAM.

**Što znače ovi brojevi iz datoteke** `/proc/buddyinfo` **?**
Za svaku zonu memorije, svaki stupac označava broj slobodnih stranica memorije (*memory pages*) u određenim blokovima.

- Prvi stupac označava broj slobodnih (dostupnih) stranica memorije standardne veličine (4kB).
- Drugi stupac označava broj slobodnih (dostupnih) blokova stranica memorije (od 4kB), pri tome je veličina bloka veličine 8kB (blok od 2 x 4kB stranica).
- Treći stupac označava broj slobodnih (dostupnih) blokova stranica memorije (od 4kB), pri tome je veličina bloka veličine 16kB (blok od 4 x 4kB stranica).
- Četvrti stupac označava broj slobodnih (dostupnih) blokova stranica memorije (od 4kB), pri tome je veličina bloka veličine 32kB (blok od 8 x 4kB stranica).
- … i u našem slučaju, tako sve do slobodnih blokova veličine od 4096 kB (1024 x 4kB stranica).

Iako je osnovna jedinica za alociranje u sustavu virtualne memorije stranica memorije (*memory page*) od 4kB, svako malo, neki program može zatražiti alociranje veće količine RAM memorije u nîzu i to u većim blokovima (engl. *Memory chunks*), koji se sastoje od više blokova koji potom sadrže više stranica od 4kB.

Stoga ti blokovi mogu biti sljedećih veličina:

- **8** kB - čime se koriste dvije stranice od 4kB.
- **16** kB - za korištenje četiri stranice od 4kB.
- **32** kB - za korištenje osam stranica od 4kB.
- **64** kB - za korištenje 16 stranica od 4kB.
- **128** kB - za korištenje 32 stranice od 4kB.
- **256** kB - za korištenje 64 stranice od 4kB.
- **512** kB - za korištenje 128 stranica od 4kB i tako dalje.

Dakle sve ovisno koliko je stranica memorije potrebno koristiti (alocirati), toliko će se blokova i koristiti.

Zbog toga i Linux kernel vodi statistiku i brine se o ovim blokovima unutar kojih se i dalje sve zapisuje u stranicama od 4kB, kao najmanjem elementu za čitanje ili zapisivanje u RAM memoriju preko sustava virtualne memorije.
Između ostaloga kernel konstantno vodi brigu i o tome koliko je koja zona memorije iskorištena, i to pomoću nekog od tri pokazivača stanja:

1. `pages_low` - ostalo je vrlo malo slobodnog prostora za adresiranje (*memory pages*).
2. `pages_min` - ostalo je malo prostora za adresiranje (*memory pages*).
3. `pages_high` - ostalo je dovoljno prostora za adresiranje (*memory pages*).

Ovisno o stanju svake zone, a koje se stalno osvježava, poseban Linux *daemon* (sistemski servis), zvan `kswapd` (*Kernel Swap Daemon*) poduzima određene korake (između njegovih drugih zadaća). Naime ovaj *servis* kada primijeti kako se određena zona memorije popunjava i dolazi do stanja `pages_min`, on pokreće jednu od svojih procedura za pronalaženje blokova memorije koji se više ne koriste te ih oslobađa. Ako je određena zona memorije još više iskorištena, sustav pokreće neagresivnu metodu oslobađanja memorije te na kraju kada više nema opcija niti zona memorije koje su slobodne, počinje koristiti ***swap*** particiju kao nadogradnju RAM memorije.
I naravno, kako se memorija oslobađa on odrađuje čišćenje odnosno oslobađanje slobodne RAM memorije.



Za više detalja o ***swap*** funkcionalnosti i ***swap*** disku, pogledajte poglavlje:
**13.8.2.1. Malo više detalja o *swap* sustavu i particiji.**

U slučaju kada se memorija više ne može očistiti, a prijeko je potrebna, u graničnim slučajevima može doći i do stanja kritične razine sveukupne dostupne virtualne memorije (RAM+SWAP), koje se naziva ***OOM*** (*Out of Memory*) stanje.



Za više detalja o ***OOM*** stanju i funkcionalnosti, pogledajte poglavlje:
**12.4 Anatomija programa u memoriji.**

U ovoj cjelini govorit ćemo o tome kako se programi učitavaju u (virtualnu) memoriju. Prije nego se program uopće krene izvršavati, njegovi dijelovi se učitavaju u sustav virtualne memorije koji čine RAM memorija i eventualno *swap* memorija. Kako se dijelovi programa učitavaju u dijelove virtualne memorije, zapisuju se i njihove memorijske lokacije odnosno adrese u takozvanu ***memory mapping*** tablicu konkretnog pokrenutog programa odnosno procesa s pripadajućim ***PID*** brojem.
Vezano za oslobađanje memorije i ovdje je moguće napraviti određene optimizacije, od kojih ćemo spomenuti sâmo jednu od potencijalno važnijih (ili problematičnijih). Naime linux obično alocira svu dostupnu memoriju kako bi se što više koristila i kao predmemorija odnosno međumemorija za podatke koji se kasnije mogu ponovno upotrijebiti bez potrebe za sporim dohvaćanjem istih podataka s diska ili mreže. Kada memorija počne biti vršno iskorištena odnosno kada je počne nedostajati bit će upotrijebljena procedura za pronalaženje stranica memorije koje nisu upotrijebljene ili koje se neće vjerojatno upotrebljavati u skorije vrijeme (Engl. ***Reclaim***). Koliko će resursa oduzeti ova procedura premještanja stranica iz memorije i za ponovno otvaranje novih stranica memorije prema potrebi, ovisi o vrsti stranica memorije. Linux prvo preferira izbacivanje stranica memorije s diska koje nisu mapirane u adresni prostor koji koristi određeni program/aplikacija/proces jer je lakše odbaciti sve reference na takve stranice memorije. Takve stranice memorije se mogu ponovno pročitati s diska, ako je to potrebno kasnije.
S druge strane stranice memorije koje su mapirane u direktan adresni prostor procesa zahtijevaju da se stranica prvo ukloni iz tog adresnog prostora prije nego što se stranica može ponovno upotrijebiti. Stranica memorije koja nije kopija stranice s diska; takozvane anonimne odnosno ***anonymous*** stranice, može se izbrisati sâmo, ako je stranica prvo označena za razmjenu i nalazi se u *swap* prostoru (što je prilično sporo jer je ovaj prostor proširenje RAM memorije koji se nalazi na tvrdom disku). Postoje i stranice memorije koje se uopće ne mogu izbrisati odnosno „čistiti“, kao što su stranice koje se koriste za kernel ili za podatke ili meta podatke koje koristi kernel.



***Ovdje govorimo o standardnom mehanizmu čišćenja odnosno oslobađanja memorije u slučajevima kada je ima premalo. Međutim postoji i mogućnost agresivnijeg čišćenja memorije o kojem ćemo sada govoriti.***

Utjecaj povrata odnosno oslobađanja memorije od podataka spremljenih u međumemoriju stoga se očito razlikuje. U *NUMA* sustavima, na svakom *NUMA* čvoru[1] se dodjeljuje više vrsta memorije. Količina slobodnog memorijskog prostora na svakom *NUMA* čvoru[1] će naravno varirati s vremenom.

Dakle, ako postoji zahtjev za memorijom, a memorija na lokalnom *NUMA* čvoru[1] će zahtijevati povrat zauzete međumemorije, a u slučaju kada drugi *NUMA* čvor ima dovoljno memorije za zadovoljavanje zahtjeva bez vraćanja/čišćenja svoje memorije, kernel ima dva izbora:

1. Pokrenuti zahtjev za povratom memorije (*reclaim*) na lokalnom *NUMA* čvoru[1], uzrokujući dodatno procesiranje od strane kernela, a zatim alociranje lokalne memorije *NUMA* čvora.
2. Dovoljno je alocirati memoriju od drugog *NUMA* čvora[1] koji ne treba povratni zahtjev odnosno procesiranje. Memorija stoga neće biti na lokalnom *NUMA* čvoru[1], ali se izbjegavaju česte provjere i procesiranja, kao u prvom slučaju. Osvježavanje odnosno proces provjere će se izvršavati samo kada sve zone memorije imaju vrlo malo slobodne memorije. Ovaj pristup smanjuje učestalost ponovnog oslobađanja memorije.

Za male *NUMA* sustave, kao što su sustavi s dva (2) *NUMA* čvora[1] (dva fizička procesora) kernel je konfiguriran da koristi drugi (***2.***) navedeni pristup. Za veće *NUMA* sustave (četiri ili više NUMA čvorova[1]) koristi se drugačiji pristup.
U kernelu postoji opcija koja određuje kako će se sustav ponašati u *NUMA* arhitekturi u slučaju kada želimo agresivnije čišćenje memorije od normalnog, a definirana je na dva mjesta (koja označavaju istu metodu) vidljiva u tablici dolje.

| *Sysctl* vrijednost | */proc* datoteka |
|---|---|
| `vm.zone_reclaim_mode` | `/proc/sys/vm/zone_reclaim_mode` |

Ovdje možemo imati nekoliko vrijednosti:

- `0` znači kako se ne bi trebalo vršiti nikakav lokalni povrat (*reclaim*) međumemorije nauštrb *page cache* odnosno međumemorije sustava. **Ovo je standardna vrijednost koju se preporučuje NE DIRATI** jer u pravilu želimo da sustav koristi što više RAM memorije kao predmemoriju ili međumemorije; primjerice za operacije prema diskovnom podsustavu (rad s datotekama i slično). Ova opcija je postavljena i za ***SMP*** sustave.
- `1` govori kernelu kako bi trebao pokrenuti proceduru čišćenja odnosno povrata memorije (*reclaim*) na lokalnom *NUMA* čvoru[1], bez obzira ako na bilo kojem drugom *NUMA* čvoru[1] postoji (udaljena) slobodno RAM memorija.
- Postoje i vrijednosti `2` i `4`, koje se odnose na *reclaim* proceduru za „*dirty pages*“ (`2`) te za *reclaim* za *swap pages*. Ove varijante su vrlo specifične te ih nećemo više objašnjavati.

**Izvori informacija: (491),(492),(493),(494),(K-5),`man 5 proc`.**

**1.** NUMA *node* odnosno NUMA čvor je fizički procesor (CPU) sa svim svojim pripadajućim CPU jezgrama, svojim memorijskim kontrolerom te svojom lokalnom RAM memorijom.

---

## 12.3.6. VM - Slabs

U ovom **naprednom** poglavlju, upoznat ćemo se s još jednim mehanizmom unutar sustava virtualne memorije, koji se naziva ***slab*** sustav. On je zadužen za efikasno alociranje predmemorije (*cache*) za potrebe kernela odnosno kernel programa. Jedan od mehanizama ***slab*** sustava brine se o tome kako uslijed alociranja i dealociranja ove memorije ne bi došlo do fragmentacije. Ovdje se radi o potrebi da kernel, kada treba alocirati memoriju u koju će se pohraniti neki podaci, a koje će trebati dohvaćati i drugi programi unutar kernela, to odradi na što efikasniji način. Ova funkcionalnost se naziva i *object pool*, unutar memorije. Naime kernel programi u *object pool* zapisuju podatke razdijeljene u blokovima memorije koje nazivamo objektima.

Nešto slično poput stranica memorije (*pages*). Alokacija ovakve memorije prvo je osmišljena u UNIX operativnom sustavu ***Solaris***, 1994. godine, a potom je uvedena u ***FreeBSD*** Unix te nešto kasnije i u *Linux*. tijekom razvoja ovog mehanizma pojavilo se i nekoliko njegovih varijanti, iako se naziv **SLAB** često ne odnosi na konkretnu implementaciju već sâm mehanizam alociranja/dealociranja memorije. Tako ćemo i mi koristiti pojam ***SLAB*** u nastavku knjige, ne kao konkretnu implementaciju, već kao naziv ovog mehanizma.

Pogledajmo sažetak najčešćih implementacija ovog mehanizma:

- **SLAB** (engl. *slab allocator*) – već objašnjen mehanizam za efikasnu alokaciju objekata u memoriji koji pokušava smanjiti fragmentaciju memorije u slučajevima kada se memorija dealocira te ponovno alocira.
- **SLOB** (engl. *simple list of blocks*) – optimiziran za sustave s ekstremno malom količinom memorije, ali pati od problema fragmentacije memorije.
- **SLUB** (engl. *unqueued slab allocator*) – on je poboljšana inačica **SLAB** mehanizma s poboljšanom podrškom za *debugging*, uz zadržavanje dobrih osobina kod fragmentacije memorije odnosno minimalizaciju fragmentacije. ***Konkretna SLUB implementacija je odabrana kao zadana od Linux kernela 6.5.***

Osim što operativni sustav na najnižoj razini od kernel funkcionalnosti, koje ćemo nazvati kernel programima i kernel modulima (*driverima*), koriste *slab* memoriju, postoji i nekoliko aplikacijskih programa koji ju koriste.
Jedan od njih je *`memcached`* koji je zamišljen kao priručna RAM memorija (*cache*) za mnoge druge programe, poput *Apache* i *nginx* web poslužitelja, ali i drugih programa, kojima na taj način ubrzava dohvaćanje podataka i u konačnici im ubrzava rad.

Tijekom alokacije i dealokacije RAM memorije u rezerviranom prostoru kernela odnosno RAM memorije za spremanje raznih objekata, dolazi do problema u performansama, jer su *troškovi* resursa/vremena kod operacija alociranja odnosno dealociranja i ponovnog alociranja memorije prilično veliki. Gledano s više razine, kreiranje i brisanje, te popunjavanja raznih objekata u memoriji na razini kernela, događa se vrlo veliki broj puta u sekundi, a računajući koliko je sâmo vremena potrebno za kreiranje i brisanje tih objekata, ispostavilo se kako to u postotku oduzima značajan dio vremena i drugih resursa (CPU), odnosno u konačnici dovodi do lošijih performansi. Stoga je uvedena nova funkcionalnost izrade posebne tehnologije zvane *slab* i njenih međuspremnika (*cache*) koji se brinu o kreiranju tih logičkih objekata, kao jedinica (blokova) za pohranu podataka, radu s njima i njihovom brisanju, ali na znatno efikasniji način. S ovakvim alociranjem *slab cache* memorije, memorija se dijeli u posebne cjeline (*chunks*) čija veličina je optimalna za spremanje i dohvaćanje objekata, točno određene veličine koja odgovara potrebama konkretnog kernel programa koji ih treba. Ovi kernel programi odnosno funkcionalnosti mogu biti funkcije za pristup diskovnom podsustavu odnosno datotečnom sustavu, mrežnom podsustavu i slično.

*Slab* sustav prati ove *slab* cjeline koje se globalno nazivaju i *cache* memorije.
Pri tome se on brine kako bi alocirao dovoljno *slab* cjelina koje se sastoje od objekata odgovarajuće veličine, odnosno kako bi preraspodijelio one koje može rasporediti drugim kernel programima. Dakle *slab* sustav se brine kojem kernel programu odnosno funkcionalnosti treba dodijeliti i koliko *slab/cache* memorije, odnosno koju *slab* cjelinu, a koja je podijeljena na objekte određene veličine. U slučaju kada neke cjeline odnosno objekti stvarno više nisu potrebni, ne radi se brisanje *slab* objekata odnosno cjelina, već se one sâmo označavaju slobodnim. Potom ih sljedeći kernel program može alocirati, ako su one veličine koja mu odgovara, a o čemu se opet brine *slab allocator* mehanizam. Na kraju se s ovim mehanizmom dodatno ubrzao proces alociranja *slab* objekata, a u konačnici i RAM memorije. Informacije o alociranoj *slab* memoriji mogu se vidjeti:

1. U datoteci `/proc/slabinfo`
2. U datoteci `/proc/meminfo` red *Slab*, pri čemu se slijedećom naredbom prikazuje ukupno zauzeće *slab* memorije:

```
cat /proc/meminfo | grep -i slab
```

3. Ili u realnom vremenu, s naredbom `slabtop` koja se ponaša poput naredbe `top`



Pogledajte i poglavlje o `/proc` virtualnom datotečnom sustavu:
**13.9.1 Direktorij */proc*.**

Vezano za zauzeće RAM memorije Linux sustava, važno je znati kako je određeni dio memorije alociran kao *slab* memorija za posebne komponente kernela. Ove komponente mogu biti:

- Takozvana *disk (cache)* međumemorija i to kao: ***inode cache***, ***dentries***, ***filesystem cache*** *(ext3/4/xfs/…)…*
- Ostale vrste (*cache*) međumemorija: ***TCP*** ili ***UDP***, ***blkdev_requests***, ***kmalloc*** (proces za alociranje memorije za aplikacije) i drugi.

Ukupno zauzeće RAM memorije je podijeljeno u tri logičke cjeline unutar sustava virtualne memorije:

- Standardno alocirane memorije.
- *Slab* memorije kao predmemorije za razne objekte kernela.
- Diskovnih predmemorija: *buffer* i *cache* (tzv. *Page Cache*).

Pri tome je moguće, gledajući na zauzetu memoriju, primjerice s programima `free`, `top` i `atop` uočiti kako imamo veliko zauzeće RAM memorije, ali vrlo malo što se tiče pokrenutih vidljivih programa (procesa). U stvarnosti je zauzeće RAM memorije prilično veliko jer se razlika (slobodna RAM memorija) dinamički koristi za upotrebu kao brze međumemorije za neko među memoriranje podataka i/ili datoteka (pr. *disk cache*) te se prema potrebi oslobađa za aplikacije/programe kada im zatreba.
Pogledajmo naredbu `slabtop` i njene osnovne prekidače:

- `-o` – ispiši statistiku sâmo jednom i izađi iz programa.
- `-s` – sortiraj prema:
    - `a` – broju aktivnih objekata.
    - `b` – broju objekata *slab* memorije.
    - `c` – veličini *(cache)* međumemorije.
    - `s` – veličini objekata.
    - `u` – iskorištenju *(cache)* međumemorije.

**Slijedi primjer**:
Pogledajmo trenutno zauzeće *slab* (*cache*) memorije, poredano prema veličini alocirane *slab* memorije (`-s c`), upotrebom naredbe: `slabtop` (skratili smo ispis):

```
slabtop -o -s c

Active / Total Objects (% used)     : 1091395 / 1140789 (95.7%)
Active / Total Slabs (% used)       : 143800 / 143821 (100.0%)
Active / Total Caches (% used)      : 140 / 312 (44.9%)
Active / Total Size (% used)        : 559006.99K / 572778.23K (97.6%)
Minimum / Average / Maximum Object : 0.02K / 0.50K / 4096.00K
```

Slijedi nastavak ispisa prethodne naredbe

```
 OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
481551 467162  97%    0.55K 68793        7    275172K radix_tree_node
 44236  44231  99%    4.00K 44236        1    176944K ip4-frags
 14492  14418  99%    0.85K  3623        4     14492K ext3_inode_cache
 11360  11322  99%    1.00K  2840        4     11360K pid_namespace
 13385  13259  99%    0.72K  2677        5     10708K proc_inode_cache
 44625  44328  99%    0.22K  2625       17     10500K dentry
   156    156 100%   64.00K   156        1      9984K kern_node_cache
```

Pogledajmo opis nekih od polja vidljivih u ispisu:

- `OBJS` – označava broj objeka koji se trenutno mogu koristiti za pojedinu funkciju kernela (vidljivu pod `NAME`).
- `ACTIVE` – označava broj aktivnih objekata koje trenutno ta funkcija i koristi.
- `USE` – ovo je odnos između *OBJS* i *ACTIVE.*
- `OBJ SIZE` – ovo je veličina pojedinog objekta (ovisi o funkciji koja ih treba).
- `CACHE SIZE` – je količina RAM memorija koja je alocirana za pojedinu funkcionalnost kernela.
- `NAME` – je ime kernel funkcionalnosti koja koristi *slab/cache* memoriju.

U primjeru ispisa, vidljivo je kako je kernel funkcija/program `radix_tree_node` zauzeo 275.172kB (238 MB) RAM memorije za svoje operacije tj. za pohranu raznih objekata. Pri tome aktivno koristi 467.162 objekata veličine 0.55kB (560 *bajta*). Zanimljiva je i kernel unos `ext3_inode_cache` koji predstavlja *Inode* unose datotečnog sustava `ext3`, koji su u međumemoriji, a konkretno ih je u ovoj međumemoriji 14.492 što zauzima 14 MB RAM memorije.
Dok `dentry` unosi poboljšavaju brzinu pristupa direktorijima predmemorirajući najčešće korištene direktorije.

U slučaju kada bismo primjerice radili neke intenzivne operacije pristupanja ili pretraživanja velikog broja datoteka, njihovi *Inode i Dentry* unosi bi također bili dodani u ovu međumemoriju, kako bi ubuduće ubrzali ove operacije.

Memorije `proc_inode_cache` i `kernfs_node_cache` su specifične za virtualne datotečne sustave `/proc` i `kernfs`. Ovdje se pohranjuju i drugi kernel objekti koji nemaju veze s diskovnim ili datotečnim sustavom poput struktura vezanih za sâm sustav virtualne memorije, mrežni podsustav (pr.`ip4-frags`), sustav za upravljanje procesima (pr. `pid_namespace`) i drugo.

Ukupno zauzeće *slab* i drugih (*cache)* međumemorija je vidljivo i iz programa `vmstat`.
Pogledajmo statistiku zauzeća memorijskog sustava s programom odnosno naredbom `vmstat` na istom ovom računalu:

```
vmstat -S M

procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  0      0    352    141 113410    0    0    68     2    1    0  0  0 99  0
```

Koristili smo prekidač `-S M` kako bismo zauzeće vidjeli u MB.
Ono što se nalazi u vršnom stupcu *`memory`* odnosno u podstupcu `cache` je diskovna međumemorija. U našem slučaju se za `cache` memoriju koristi 110GB RAM memorije (113.410 MB). Podstupac `buff` se odnosi na *buffer* memoriju koja se koristi kao predmemorija za blokove podataka diskovnog sustava te za metapodatke poput objekata **VFS** sustava vezanih za diskovni podsustav odnosno koristi se kao predmemorija za diskovni podsustav.



**SLAB međumemorija je samo manjim djelom vidljiva kao "`buff`" i "`cache`" u ispisu naredbe `vmstat`.**



*Cache* memorija vidljiva u gornjoj statistici se koristi kao diskovna međumemorija (tzv. ***Page cache***). Pogledajte:
**12.3.3.5. Naredba *vmstat*.**
**12.6. Dodatne optimizacije sustava virtualne memorije.**
**14.1.2. Optimizacija *Page Cache* sloja.**

Za statistiku upotrebe cjelokupne Slab (cache) međumemorije (**VFS**+*svi kernel objekti*) možemo koristiti naredbu `slabinfo`:

```
slabinfo -T

Slabcache Totals
----------------------------------------------------------------------------
Slabcaches :              157   Aliases  :         154->69   Active:    114
Memory used:            63.0M   # Loss   :            1.7M   MRatio:     2%
# Objects  :           465.4K   # PartObj:            6.9K   ORatio:     1%
```

Ovaj ispis će nam dati detalje u zauzeću memorije od strane *Slab* (*cache*) međumemorija kernela.

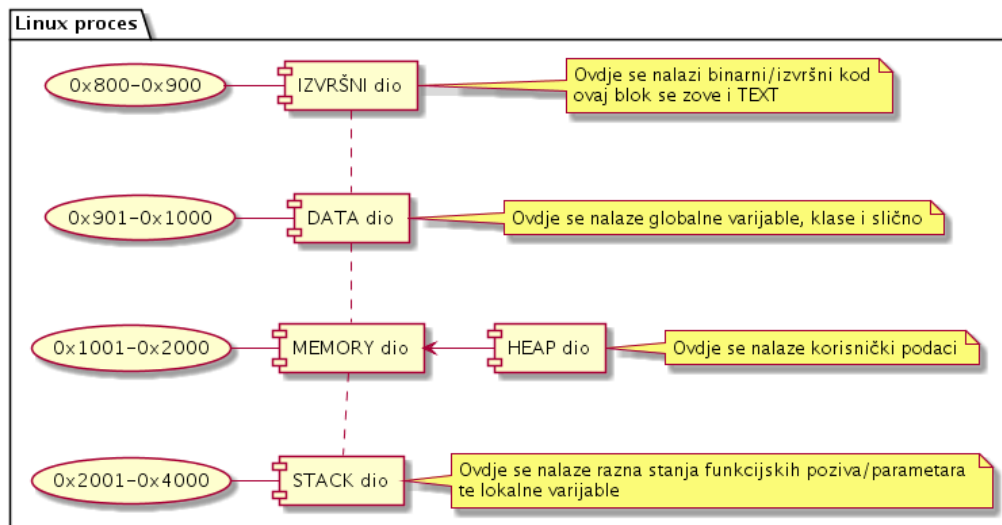
**Izvori informacija: (661),(662),(663),(664),(665),(1560),(1561),(1562),(1563),(K-5), `man slabtop`, `man vmstat`, `man slabinfo`,`man 5 proc`.**

# 12.4. Anatomija programa u memoriji

**Slijedi napredno poglavlje!**

U ovom poglavlju govorit ćemo o tome kako se programi učitavaju u (virtualnu) memoriju. Prije nego se program uopće krene izvršavati, njegovi dijelovi se prvo učitavaju u sustav virtualne memorije, a koju čine RAM i eventualno *swap* memorija. Kako se dijelovi programa učitavaju u područja virtualne memorije, zapisuju se i njihove memorijske lokacije odnosno adrese, u takozvanu **memory mapping tablicu** koja je specifična za svaki pokrenuti programa odnosno proces. Pod specifična mislimo na to kako ima oznaku njegovog pripadajućeg ***PID*** broja (od procesa). Konkretno se svi ti podaci zapisuju u *memory mapping* datoteku `/proc/PID/maps`. Dakle u ovoj tablici (datoteci) preslikavanja (mapiranja) regija memorije, čuvaju se točne memorijske lokacije odnosno adrese na kojima se nalaze dijelovi učitanog programa. Naime niti jedan program se ne učitava cijeli, u komadu, već u dijelovima i cjelinama od kojih svaka ima svoju namjenu. Pogledajmo kako izgleda jedan linux proces odnosno pokrenuti program, koji je učitan u memoriju. S lijeve strane su vidljive (*virtualne*) memorijske adrese na kojima se nalazi određeni dio programa, dok je u sredini naveden dio programa odnosno njegova logička cjelina (slika **102.**).

**Slika 102. Anatomija programa u memoriji.**




Kako se program izvršava, dohvaćaju se dijelovi sâmog programa, biblioteka ili drugih objekata iz sustava virtualne memorije, odnosno iz stranica virtualne memorije (*memory pages*). Stranice virtualne memorije za sada zamislite kao najmanje blokove memorije u koje se mogu zapisivati ili čitati podaci. Kako smo vidjeli na slici gore, svaki izvršni program se sastoji od nekoliko logičkih cjelina koje se učitavaju u različite blokove memorije. Dodatno, kako svaki program za rad koristi i razne biblioteke, i one se moraju učitati u memoriju kako bi ih naš program uopće mogao koristiti. Nadalje neke programe može koristiti više korisnika ili drugih programa istovremeno.

Zamislimo *web* preglednik (pr. ***Firefox***) koji je pokrenuo prvi korisnik, te krenuo sa *surfanjem webom*. Nakon toga, ako bilo koji drugi korisnik također pokrene isti web preglednik (*Firefox*), sustav će to prepoznati te će pokušati iskoristiti već učitani izvršni dio *Firefox* web preglednika, kao i biblioteke koje je on već učitao, ali u adresnom prostoru drugog korisnika. Dakle ovdje se radi o binarnim (izvršnim) datotekama kao i o bibliotekama, koje su statične i označene sâmo za čitanje *(read-only)* pa se one bez problema mogu i dijeliti s drugim programima/procesima i korisnicima naravno, ako je to dozvoljeno odnosno omogućeno za konkretan program. Stoga se za novog korisnika neće morati učitavati sve izvršne datoteke i biblioteke programa *Firefox* s diska, već će sustav za njega sâmo povezati (*mapirati)* one stranice memorije, koje su označene da se mogu dijeliti prema drugom korisniku (takozvane „*shareable pages*"). I potom mu dozvoliti pristup na njih, kako bi on mogao izvršiti programski (izvršni/binarni) kôd, koji se već nalazi učitan u memoriju.

Taj izvršni dio programa se tada dijeli u memoriji između više korisnika ili programa. To je **IZVRŠNI** dio na slici označen kao ***TEXT***. Naime ovdje smo govorili o izvršnom dijelu programa i njegovoj lokaciji u virtualnoj (*RAM*) memoriji te dijeljenju tog izvršnog dijela programa odnosno memorijskih adresa na kojima se taj binarni/izvršni dio programa i nalazi.
Ono što se ne može dijeliti, već se mora kopirati je dio koji je na slici označen kao ***DATA*** *dio* izvršnog programa, jer on sadrži globalne varijable, klase i slično, pa se mora kopirati za drugu instancu istog programa. Potom će kod otvaranja neke *web* stranice za drugog korisnika, biti kreirana nova *mapiranja* odnosno bit će kreirane nove stranice memorije koje će se koristiti za tu namjenu, a to je dio označen kao ***MEMORY*** *dio*. Još jedan dio memorije se kopira između više instanci istog programa, a to su stanja svih funkcijskih poziva, koja se snimaju u dio koji smo na slici označili kao ***STACK*** *dio*.
Adresiranje *STACK* djela svakog pokrenutog programa (*procesa*), možemo vidjeti unutar datoteke: `/proc/PID/stack` pri čemu je ***`PID`*** stvaran ***PID*** broj procesa koji gledamo.

Pogledajmo *STACK* dio memorije, za proces ***PID*** broja: **7945** (to je u našem slučaju `crond` servis):

```
cat /proc/7945/stack
[<ffffffff810b586b>] hrtimer_nanosleep+0xbb/0x180
[<ffffffff810b59ae>] SyS_nanosleep+0x7e/0x90
[<ffffffff81697649>] system_call_fastpath+0x16/0x1b
[<ffffffffffffffff>] 0xffffffffffffffff
```

Na prethodnom ispisu je vidljivo da naš proces u ***STACK*** dijelu ima tri vidljive programske nîti odnosno *thread*a, koja predstavljaju prva tri reda ispisa. Pri tome je vidljivo:

- U lijevom dijelu (stupcu) su memorijske adrese.
- U desnom dijelu (stupcu) su programske nîti odnosno funkcije koje se izvršavaju:
    - Primjerice u prvom redu je to funkcija: `hrtimer_nanosleep` – ona konkretno pauzira dalje izvršavanje programa na određeno vrijeme. ***Info:*** ***https://linux.die.net/man/2/nanosleep.***
    - U drugom redu je također funkcija za pauziranje izvršavanja programa: `SyS_nanosleep`

Za konkretan program iz primjera odnosno *`cron`* servis/*daemon*, ovo je normalno ponašanje jer se on pokreće tijekom pokretanja sustava te se pauzira, obično na 1 minutu, pa se ponovno pokreće i provjerava ima li novih *crontab* unosa, koje potom izvršava (ako ih ima), te se opet pauzira i tako u krug. Ako sâmo želimo vidjeti koja se trenutna funkcija izvršava za konkretan program, to možemo vidjeti očitavanjem vrijednosti posebne datoteke: `/proc/`**`PID`**`/wchan` pri čemu je ***`PID`*** direktorij, zapravo ***PID*** broj našeg procesa. U primjeru od gore bi to bilo čitanje sadržaja sljedeće datoteke:

```
cat /proc/7945/wchan

hrtimer_nanosleep
```

Vratimo se na priču o izvršavanju programa. Pri izvršavanju programa normalno dolazi do uspješnog dohvaćanja stranica memorije odnosno blokova memorije u koje su ti dijelovi programa pohranjeni, ako se podaci već nalaze u virtualnoj memoriji, odnosno u konačnici u RAM memoriji.
Međutim može doći i do neuspješnog dohvaćanja, koje ćemo ubrzo objasniti.

Naime sustav u nekim trenucima ovisno o iskorištenju RAM memorije ne mora učitavati programe u potpunosti u RAM memoriju ili može učitati sve dijelove određenog programa, a poslije osloboditi one dijelove programa koji se ne koriste često, te ih ponovno učitati tek kada budu zatraženi odnosno potrebni. Na ovaj način se može uštedjeti na iskorištenju RAM memorije, obično u situacijama kada ona postaje prepunjena. U toj graničnoj situaciji kernel će pokrenuti i mehanizam znan kao ***memory pressure***, odnosno u konačnici, ako je potrebno čak i aktivaciju *swap* particije. Pri tom procesu će se prvo pokušati osloboditi što više nekorištene RAM memorije, ali ako niti to ne pomogne, krenuti će se u proširenje RAM memorije upotrebom *swap* particije na disku, za čiju funkciju je zadužen linux servis/*daemon* `kswapd`.

## *Minor page fault*

Postoje slučajevi kada su nastupile neke promjene, zbog gore navedenih mehanizama ili zbog nekih drugih uzroka, a zbog kojih su se promijenile memorijske adrese odnosno njihova mapiranja. Točnije u bilo kojem slučaju kada se napravi neka promjena u *mapiranju* adresa pri kojemu *NIJE potreban* ponovni pristup disku kako bi se dohvatile datoteke/podaci, a drugi program/proces/korisnik se oslanja na poveznicu s tim mapiranjem, tada dolazi do stanja: ***`Minor page fault`***. Nakon ovog stanja će taj drugi program/proces dobiti osvježene adrese odnosno poveznice na njih, a potom normalno nastaviti s radom. Ovo je uobičajeno u radu i ne predstavlja neki poseban problem.

## *Copy on write*

Slična stvar se može dogoditi i s dijelom memorije u kojemu se nalazi logička cjelina programa koju smo na slici gore definirali kao ***DATA*** *dio*, u koji se obično pohranjuju varijable, klase i slični objekti, izvršnog programa. Naime u bilo kojem trenutku izvršni dio programa, na slici 102 vidljiv kao *IZVRŠNI dio* ili ***TEXT***, može od kernela zatražiti proširenje tog dijela, za primjerice 20MB. Linux kernel će odgovoriti programu, kako mu je alocirao dodatnih 20MB RAM memorije, ali u stvarnosti to neće napraviti. Kernel će u sustavu virtualne memorije, koju program jedino i vidi, zapravo alocirati traženih 20MB virtualne memorije i označiti ih kao ***copy on write***. To znači da sve dok ta memorija nije stvarno zauzeta nema potrebe da se ona stvarno alocira u stvarnoj RAM memoriji.
Ali u trenutku kada program stvarno i krene s korištenjem te memorije, kernel će alocirati dio stvarne RAM memorije za tu potrebu. Ova funkcionalnost se također stalno događa na svim modernim operativnim sustavima pa tako i na Linuxu. To se radi stoga kako bi se stvarna RAM memorija iskorištavala na optimalan način jer se stvarna odnosno fizička RAM memorija alocira samo onda i u trenutku kada je stvarno potreba, bez rezervacije unaprijed, koja se ne bi u potpunosti koristila, te bi stajala potencijalno neiskorištena.

## *Posebni slučajevi*

Moguće je tijekom inicijalizacije odnosno pokretanja programa, od operativnog sustava odmah zatražiti alociranje veće količine memorije odjednom. Naime u normalnom radu kada se program učitava u memoriju, sustav mu dodjeljuje memorije koliko mu je potrebno, a kako je opisano gore. Međutim visoko performantni programi koji intenzivno koriste memoriju, a ne žele da im sustav malo po malo alocira dodatnu memoriju kako im raste potreba za njom, mogu od sustava zatražiti rezervaciju određene količine memorije odmah, već od trenutka pokretanja sâmog programa.
Ne zaboravimo kako se memorija u konačnici alocira u stranicama od 4kB pa je tako svako alociranje novih stranica memorije u radu programa nešto što oduzima mali djelić vremena.
To je vrijeme za normalne programe zanemarivo, ali za one posebne namjene i poput onih koji stalno zahtijevaju novu količinu memorije i/ili ju stalno popunjavaju i prazne, nešto što može utjecati na performanse programa. Naime nije svejedno obrađuje li vaš program sâmo nekoliko određenih upita ili radnji ili njih na milijune istovremeno.

### *Major page fault*

Postoje i dodatni slučajevi, kada je došlo do:

- Objašnjenog mehanizma *memory pressure*, uslijed nedostatka RAM memorije pa je sustav krenuo u čišćenje *RAMa* i aktivaciju *swap*-a pa postoje određeni blokovi memorije koji su mapirani, ali podaci zapravo nisu učitani s diska u RAM memoriju.
- Ili kada neki program ili komponenta sustava pokuša čitati iz regije (RAM) memorije koja još nije dohvaćena s diska, u trenutku kada je alocirana i mapirana RAM memorija, a još svi podaci s diska nisu stigli biti učitani u RAM memoriju.
- Ili u nekom od slučajeva kada nisu učitani svi dijelovi programa u RAM memoriju, ali su blokovi memorije mapirani i pokazuju na te podatke, pa ih je potrebno i učitati s diska.

U svakom od tri navedena slučaja, bit će generirana poruka: `Major page fault` (***MAJFL***).
Ovo nije nužno veliki problem, ako ih nemamo često i puno.

Pogledajmo kako vidjeti statistike za sve pokrenute programe (procese) po pitanju ***Major page fault*** i ***Minor page fault***. To ćemo napraviti pomoću naredbe `ps` pomoću dodatnih opcija (`-eo`) i to (`min_flt` i `maj_flt`) te dodatnog filtriranja:

```
ps -eo min_flt,maj_flt,comm | grep -v 0 | (sed -u 1q; sort -rn)

MINFL   MAJFL   COMMAND
 17264    196   firewalld
  1833      2   systemd-udevd
  1363      6   systemd-logind
  1331     86   NetworkManager
  1218     37   sshd
   936      2   sftp-server
   914     32   rsyslogd
. . .
```

Dakle u ispisu vidimo poredane sve programe (procese) prema tome koliko imaju **`MINFL`** i **`MAJFL`** statistika.

Drugi način je pomoću naredbe `pidstat` (skratili smo ispis):

```
pidstat -r

10:20:46 AM UID   PID  minflt/s   majflt/s    VSZ     RSS   %MEM  Command
10:20:46 AM   0     1     12.36    0.20     105864   15500   0.85  systemd
10:20:46 AM   0   525      0.84    0.01      22272    9412   0.52  systemd-journal
10:20:46 AM   0    539     1.76    0.00      33972   11184   0.61  systemd-udevd
10:20:46 AM   0   565      0.07    0.00      18112    4392   0.24  auditd
10:20:46 AM   0   596     16.61    0.19     125008   40552   2.22  firewalld
. . .
```

Ovdje iste statistike vidimo kao `minflt/s` i `majflt/s`.



Pogledajte i sljedeća poglavlja:
**10.7.2.3.7. Naredba *pidstat* za CPU (ali i memoriju i disk prema potrebi).**
**12.6. Dodatne optimizacije sustava virtualne memorije**.
**12.6.1. Drugi granični slučajevi.**

**Izvori informacija: (250),(251),(252),(253),(K-4),(K-5), `man ps`,`man pidstat`,`man 5 proc`.**

### 12.4.1. Mehanizmi zaštite memorije

**Slijedi napredno poglavlje!**
U ovom poglavlju govorit ćemo o zaštitnim mehanizmima koje koristi linux, kako bi osigurao aplikacije odnosno procese od zlonamjernih manipulacija.



Stoga prvo dobro proučite i sustav virtualne memorije u poglavlju:
**12. Sustav virtualne memorije i memorijski menadžment.**

Tradicionalne računalne arhitekture koje su implementirale mehanizme zaštite memorije tipično imaju dva stanja prema kojima se mogu mapirati regije memorije:

- Regija memorije samo za čitanje (*R*).
- Ili regija memorije za pisanje (*W*).

Točni detalji o sposobnostima zaštite memorije ovise o arhitekturi CPU-a, ali ovi osnovni mehanizmi vrijede za većinu 32 bîtnih sustava koji koriste x86 arhitekturu procesora.
Međutim pored osnovnog ograničavanja određene regije memorije za čitanje ili pisanje, svaki sustav također mora implementirati i neke druge osnovne zaštite memorije kao što je mehanizam koji osigurava da različiti procesi ne mogu čitati ili pisati u memorijske regije (područja) drugog programa na koja nemaju prava.

Način implementacije ovog sustava je da svaki proces ima posebnu strukturu podataka dodijeljenih od strane kernela, koja se naziva tablica stranica memorije (engl *Page table*) s kojom radi i koju koristi kernel Linuxa.

Ova tablica stranica memorije sadrži poveznice regija virtualne memorije na fizičku memoriju ili na proširenje fizičke memorije (*swap*), ako se ona koristi. Za svaku stranicu memorije koja je definirana u ovoj tablici nalazi se i bît koji definira ima li se na tu stranicu pravo čitanja ili pisanja, a s kojim se ***MMU*** (komponenata *kontrolera memorije unutar CPU-a*) koristi za provođenje ovih prava, za regije memorije koje bi trebale biti namijenjene samo za čitanje ili pisanje. Zaštita memorije između procesa događa se tako što će kernel naložiti ***MMU***-u (*Memory Management Unit-u*) da za proces koji tek treba pokrenuti; dakle kod svakog *context switching-a* odnosno prebacivanja na aktivni proces (program), taj proces smije koristiti tablicu (*Page table*) koja je trenutno dostupna sâmo za njega. To znači da proces nikako ne može ući u stanje u kojem može koristi tablicu stranica memorije (*Page table)* drugog programa (procesa) i time doći do njegovih memorijskih adresa i sâmim time sadržaja memorije.

Vezano za označavanje regija memorije samo za čitanje, postoji i nekoliko različitih slučajeva upotrebe od kojih je vjerojatno najvažniji slučaj ovakve upotrebe (samo za čitanje) onaj u kojem se regija memorije koristi za takozvani tekstualni segment (***TEXT*** dio) programa. Ovaj tekstualni segment programa u memoriji je dio memorije procesa (pokrenutog programa) koja sadrži stvarni strojni programski odnosno izvršni kôd za taj proces. Razlog zbog kojeg je poželjno mapirati ovo područje odnosno regiju memorije sâmo za čitanje je to da nam ovaj mehanizam pomaže kod zaštite od napada kod kojeg se pokušava ubaciti zlonamjerni programski kôd u proces odnosno u regiju memorije u kojoj se nalazi izvršni dio programa. Naime ako napadač može ubaciti zlonamjerni programski kôd u područje memorije koje sadrži programski kôd procesa, onda može i izvršiti proizvoljni kôd s dozvolama sâmog procesa.

Dakle označavanjem ***TEXT*** djela odnosno regije memorije procesa sâmo za čitanje bez prava mijenjanja/zapisivanja osigurava nas od malicioznih ili nenamjernih promjena izvršnog programskog kôda sâmog procesa.
Izvor problema se zapravo nalazi na drugom mjestu; naime drugi dio odnosno regija memorije svakog procesa je ***DATA*** dio u koji se moraju moći zapisivati podaci. Ova regija memorije svakog procesa (programa) mora biti postavljena s pravima čitanja i zapisivanja. U ovu regiju memorije procesa se između ostalog zapisuju i informacije o tome koji dio programskog izvršnog kôda se trenutno izvršava.

Za x86 arhitekturu računala ovdje se zapisuje memorijska adresa (lokacija) za poziv funkcija unutar ***STACK*** regije memorije našeg (svakog) procesa, a stoga ***STACK*** regija memorije procesa mora biti mapirana s pravima čitanja (*write*). Stoga napadač koji na neki način može doći do ***STACK*** (*stôg*) dijela memorije konkretnog procesa, može promijeniti i adrese s koje se određena funkcija treba pozvati, tako da umjesto pozivanja legitimne funkcije pozove svoju zlonamjernu funkciju koju je upisao na memorijsku lokaciju unutar STACK memorije napadnutog procesa (programa).

Metoda s kojom je uopće moguće „*uletjeti*“ u ***STACK*** dio memorije procesa je poznata kako „*stack buffer overflow*“ odnosno općenito kao „*buffer overflow*“. Međutim osim zlonamjernog uzrokovanja ovog stanja, može se dogoditi i klasično prepunjavanje ove (***STACK***) memorije loše napisanim programom, što će obično rezultirati rušenjem programa.

## *No-Execute* (NX) bit - prva pomoć

***NX*** bît odnosno funkcionalnost, prisutna na svim 64-bîtnim x86 ali i nekim novijim 32-bîtnim sustavima, važna je mjera koja pomaže ublažavanju ovakvih napada. Sustavi koji implementiraju ovu funkciju imaju tri bîta dozvola za regije memorije: čitanje, pisanje i izvršavanje. Te dozvole rade točno kao i dozvole (ovlasti) na datotečnom sustavu: za čitanje (***R***), pisanje (***W***) i izvršavanje (***X***). Ono što se događa na sustavima koji koriste ***NX*** bît je to da su tekstualna područja (***TEXT***) memorije mapirana da budu označena za čitanje + izvršavanje, a podatkovna područja (***DATA***) mapirana da budu za čitanje + pisanje, ali ne i izvršavanje.

Za ***STACK*** dio bi tada bilo preporučivo da se onemogući izvršavanje, a omogući samo čitanje (***R***) i zapisivanje (***W***) jer se tada iz te regije memorije nikako neće moći pokrenuti nikakav zlonamjerno ubačeni programski kôd.
Za ovu funkcionalnost je potrebna i podrška od strane procesora (CPU), odnosno potrebno je da su vidljive dvije CPU zastavice (`nx` i `pae`).

Pogledajmo imamo li ove zastavice/funkcionalnosti unutar našeg CPU-a, sa sljedećim nîzom naredbi:

```
egrep -o '^flags|nx|pae' /proc/cpuinfo | grep -v flags| sort | uniq
```

Dodatno pogledajmo je li **NX** funkcionalnost aktivirana (sa sljedećim nîzom naredbi):

```
dmesg | grep --color '[NX|DX]*protection'
[    0.000000] NX (Execute Disable) protection: active
```

Ako ovdje imam status `active`, što znači kako je sve u redu.



*Kako bi ova funkcionalnost radila, potrebno ju je i uključiti tijekom kompiliranja programa odnosno potrebna je podrška u programskom jeziku u kojem izrađujemo naše programe (aplikacije).*

**Kako uopće možemo vidjeti ima li određeni program uključenu ovu podršku (*Executable Stack Protection*)?**
Koristit ćemo naredbu `readelf` s kojom možemo dobiti razne korisne informacije o binarnim izvršnim ***ELF*** datotekama.

Pogledajmo primjerice program ***ssh*** koji se nalazi u direktoriju: `/usr/bin/ssh`, gledano s naredbom `readelf`:

```
readelf -l /usr/bin/ssh

Elf file type is DYN (Shared object file)
Entry point 0xcd16
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP         0x0000000000000238 0x0000000000000238 0x0000000000000238
                 0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x000000000009cf3c 0x000000000009cf3c  R E    200000

  LOAD           0x000000000009d208 0x000000000029d208 0x000000000029d208
                 0x0000000000003068 0x0000000000006d18  RW     200000
  DYNAMIC        0x000000000009f120 0x000000000029f120 0x000000000029f120
                 0x00000000000002e0 0x00000000000002e0  RW     8
  NOTE           0x0000000000000254 0x0000000000000254 0x0000000000000254
                 0x0000000000000044 0x0000000000000044  R      4
  GNU_EH_FRAME   0x000000000008f330 0x000000000008f330 0x000000000008f330
                 0x0000000000001de4 0x0000000000001de4  R      4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     10
  GNU_RELRO      0x000000000009d208 0x000000000029d208 0x000000000029d208
                 0x0000000000002df8 0x0000000000002df8  R      1
```

U retku (`GNU_STACK`) vidimo kako u ***STACK*** regiji ovog programa imamo definirano kako se postavljaju samo prava `RW`, što znači kako se zabranjuje ***E*** (izvršavanje= Engl. *Execute*) što nam govori kako je ovaj mehanizam uključen.
Ipak i ovdje (kod upotrebe ***NX*** bîta) je moguće izvršiti neke vrste napada o kojima sada nećemo govoriti. Dodatno postoje i drugi mehanizmi zaštite, a koji ovise o mogućnostima programskog jezika.

## *Data Execution Prevention (DEP)*
Značajka *Data Execution Prevention* (**DEP**) sprječava aplikaciju ili servis u izvršavanju koda u neizvršnom memorijskom području. Hardverski nametnuti **DEP** radi u kombinaciji s **NX** (Never eXecute) bitom na kompatibilnim procesorima. *DEP* sprječava izvršavanje koda iz određenih područja memorije koja su označena kao neizvršna. Cilj je spriječiti napade koji pokušavaju izvršiti maliciozni kod iz područja kao što su *STACK* ili *HEAP*. Pri tome procesor i operativni sustav surađuju kako bi označili određene segmente memorije (npr. *STACK*, *HEAP*) kao neizvršne. Ako se pokušava izvršiti kod iz tih područja, memorije sustav generira poruku i terminira proces.

## *Address Space Layout Randomization* (ASLR)
*ASLR* je mehanizam koji je tehnički komplementaran ***NX*** bîtu, ali je mnogo snažniji kada je omogućen ***NX*** bît. ***ASLR*** nas može zaštititi od nekih vrsta *buffer overflow* napada, tako da se tijekom svakog učitavanja programa u memoriju, bazni dio programa, biblioteke koje program učitava, kao i *HEAP* i *STACK* dio programa učitavaju, svaki puta na druge lokacije (regije) memorije. Tako napadač nikada ne može predvidjeti na kojoj memorijskoj lokaciji se nalazi koji dio kojeg programa.
Dakle kod svakog učitavanja programa u RAM memoriju svi dijelovi programa se učitavaju na druge memorijske lokacije.



Pogledajte i poglavlje:
**11.2.3.4 Napredno o dijeljenim bibliotekama**.
← Pogledajte dio u kojemu vidimo u koje regije memorije se program učitava (`/proc/`***`PID`***`/maps`).

U Linuxu imamo posebnu *sysctl* varijablu: `kernel.randomize_va_space` odnosno pripadajuću `/proc` datoteku: `/proc/sys/kernel/randomize_va_space` u kojoj globalno definiramo, je li *ASLR* aktivan ili nije.

Pogledajmo koje vrijednosti možemo postaviti ovoj varijabli, te što one znače:

- `0` – znači da smo u potpunosti isključili *ASLR* (ne preporuča se isključiti ju).
- `1` – znači kako se „*randomizira*“ (odabiru se slučajne memorijske lokacije) za učitavanje ***STACK*** dijela, ***VDSO*** objekata, i dijelova memorije koje se dijele (*share memory*).
- `2` – znači kako se „*randomizira*“ (odabiru se slučajne memorijske lokacije) za učitavanje ***STACK*** dijela, ***VDSO*** objekata, i dijelova memorije koje se dijele (*share memory*), ali i ***DATA*** dijela programa.
  ***Ovo je standardno postavljena vrijednost koja donosi maksimalnu zaštitu.***

Provjerimo naš sustav:
```
sysctl kernel.randomize_va_space

kernel.randomize_va_space = 2
```
Vidimo kako je aktivna vrijednost `2` što znači kako je ***ASLR*** aktivan.

## Pointer Authentication Codes (PAC)
PAC koristi kriptografske potpise za provjeru autentičnosti pokazivača, čime se sprječava manipulacija pokazivačima i snižava mogućnost napada kao što su *control-flow hijacking*. *PAC* dodaje kriptografski potpis na pokazivače. Prilikom korištenja pokazivača, sustav provjerava potpis i potvrđuje autentičnost pokazivača. Ova značajka traži podršku u procesoru (za sada ju podržava **ARM** arhitektura procesora).

## Control Flow Integrity (CFI)
*CFI* provjerava je li tijek kontrole programa legitiman. To uključuje provjeru da li skokovi i pozivi funkcija idu na dozvoljene adrese. *CFI* osigurava da napadači ne mogu preusmjeriti tijek izvršavanja programa na proizvoljne memorijske lokacije koje bi mogle biti iskorištene za izvršavanje malicioznog koda.

**Konfiguriranje CFI na Linuxu**
Linux kernel mora biti konfiguriran s podrškom za *CFI*. To uključuje omogućavanje odgovarajućih opcija prilikom konfiguracije kernela. Provjerite da je kernel verzija koja se koristi podržava *CFI* (Linux 4.9 i noviji).
Dodajte ili provjerite jesu li sljedeće opcije omogućene u `config` datoteci kernela:
```
CONFIG_CFI_CLANG=y
CONFIG_CFI_PERMISSIVE=y  # ← Ovo je opcionalno
```
Dakle provjeru možemo napraviti sa:
```
grep -i CFI /boot/config-`uname -r`
```
← Potrebna je podrška prilikom kompiliranje programa to jest podrška za ***CFI*** (pr. **Clang** ili **LLVM**).

**Izvori informacija: (529),(530),(531),(532),(533),(534),(535),(K-4),(K-5)**,`man readelf`,`man sysctl`.

# 12.5. *Out of Memory* stanje rada i konfiguracija *OOM killera*

***Slijedi napredno poglavlje!***
U prijašnjim poglavljima smo već govorili o tome kako Linux kernel alocira memoriju na zahtjev aplikacija pokrenutih na sustavu. Budući da mnoge aplikacije rezerviraju svoju memoriju unaprijed, ali i često ne koriste svu tu raspoloživu memoriju, kernel je dizajniran s mogućnošću alociranja više memorije nego je stvarno postoji na sustavu (Engl. *Over-commit*), sve u svrhu učinkovitije upotrebe memorije i u konačnici cijelog sustava. Ovaj model prekomjernog rezerviranja memorije dopušta kernelu da dodijeli više memorije aplikacijama nego što je zapravo fizički dostupno na sustavu.

Definicija ponašanja prekomjerne rezervacije memorije definirana je u sljedećim *sysctl* varijablama (i u naravno pripadajućim ***/proc/*** datotekama):

- `vm.overcommit_memory = X` – ovo je osnovna postavka sustava s kojom se uključuje ili isključuje *memory overcommitment* funkcionalnost ili se postavlja u posebne načine rada. ***Standardno je sustav postavljen na vrijednost `0`***.
  Pri tome ova vrijednost koju smo označili kao **X** može biti i:
  - **0** – Ako imamo postavljenu ovu vrijednost tada će kernel pokušati procijeniti količinu slobodne memorije kada programi/aplikacije traže više memorije za sebe, sve do određene granice ukupne količine memorije koja se dopušta za upotrebu. Pri tome se u izračun uzima u obzir ukupna slobodna RAM memorija te dostupna *cache* memorija, slobodna *SWAP* memorija kao i informacija koliko se od svih navedenih memorija može osloboditi u datom trenutku. Procjena se radi posebnim heurističkim algoritmom. Ovo je standardna postavka sustava.
  - **1** – Ako imamo postavljenu ovu vrijednost, tada će kernel uvijek pretpostaviti da uvijek ima dovoljno slobodne memorije (što često nije optimalno ili najbolje rješenje).
  - **2** – Ako ipak imamo postavljenu ovu vrijednost, tada implicitno **isključujemo** *memory overcommitment* funkcionalnost odnosno isključujemo mogućnost prekomjerne rezervacije memorije.
    Samo i isključivo ako imamo postavljeno `vm.overcommit_memory = 2` tada imamo i dvije dodatne mogućnosti od kojih se mogu koristiti ili jedna ili druga (ako je jedna u upotrebi druga ima vrijednost *0*):
    - `vm.overcommit_ratio = X` – predstavlja vrijednost u postotku (%) koliko postotaka sustava virtualne memorije se uopće smije koristiti za programe/aplikacije, pri tome se misli na korisničke programe, a ne sistemske. Pri tome je virtualna memorija= RAM + SWAP memorija. Standardno je postavljeno na 50, dakle 50%. Naime ako ovdje imamo vrijednost 50, to znači da će ukupna količina memorije dostupna za korisničke aplikacije biti slijedeće veličine (dodajte svoje veličine RAM i SWAP memorije te izračunajte):
      $Dostupna\ overcommit\ memorija = \boldsymbol{Swap} + \boldsymbol{RAM} \times \left(\frac{\boldsymbol{overcommit_ratio}}{\boldsymbol{100}}\right)$
    - `vm.overcommit_kbytes = X` - ovo je ista postavka kao prethodna, ali u kojoj se vrijednost umjesto u postotku (%) definira u kB memorije.

U slučaju kada smo se recimo odlučili da ne želimo imati ***memory overcommitment*** jer imamo specifične poslužiteljske potrebe kao i posebne aplikacije (pr. ***In-memory*** baze podataka), napravimo sljedeće (s čime prebacujmo sustav u drugi način rada [**2**<sup></sup>gore]):

```
sysctl -w vm.overcommit_memory=2
```

Provjerimo na koliko je trenutno postavljen `overcommit_ratio`:

```
sysctl vm.overcommit_ratio
vm.overcommit_ratio = 50
```

Vidimo kako je sada samo 50% virtualne memorije dostupno za korisničke programe. Mi ćemo to povećati na 80%:

```
Sysctl -w vm.overcommit_ratio=80
```

Sada pogledajmo koliko memorije je postalo dostupno za korisničke programe (poslužitelj ima 54GB RAM= 57 509 656 kB):

```
cat /proc/meminfo | grep Commit

CommitLimit:     46007724 kB
Committed_AS:    44014588 kB
```

Dakle vidimo kako sada imamo maksimalno dostupno (`CommitLimit`): `46007724` kB = ~46GB za (korisničke) aplikacije i sustav. Ovdje vidimo i još jednu vrijednost (`Committed_AS`) koja postavlja granicu malo niže, u našem slučaju na ~44GB koliko mora biti dovoljno za korisničke aplikacije i kada traže maksimalno memorije koju u tom trenutku i ne koriste.
Ovdje imamo malu razliku u odnosu na (`CommitLimit`) jer sustav ostavlja nešto memorije za svaki slučaj, kako ne bi došlo do takozvanog kritičnog stanja nedostatka memorije odnosno ***OOM*** stanja o kojem ćemo kasnije govoriti.

Ako aplikacija (proces) stvarno počne i koristiti svu memoriju koja mu je dodijeljena, kernel mu normalno pruža te resurse (memoriju). Međutim kada prevelik broj aplikacija započne s korištenjem sve memorije koja im je dodijeljena, model pretjerivanja može postati problematičan i kernel mora početi ubijati procese kako bi sustav ostao u funkciji. Mehanizam koji kernel koristi za oporavak odnosno čišćenje memorije na sustavu naziva se ubojica prekomjerne memorije ili ***OOM*** *killer* (Engl. *Out-of-memory killer*). ***OOM Killer**** na Linuxu ima nekoliko opcija konfiguracije koje omogućuju određeni odabir ponašanja koje će sustav koristiti kada se suočava sa stanjem u kojem mora doći do ubijanja aplikacija koje su počele (nenormalno) zauzimati sve više (RAM) memorije.



Navedene postavke i mogućnosti ovise o sistemskom okruženju i aplikacijama koje su instalirane na njemu.



Osnovne *Poruke o procesima ubijenim od strane **OOM*** *će se upisati u log datoteku:* `/var/log/messages`, *a bit će vidljive kao:* `kernel: Out of Memory: Killed process XX score YY or sacrifice child`
*Pri tome oznaka **XX** predstavlja **PID** broj ubijenog procesa a tzv.* `score` *broj* **YY** *je **OOM težinski koeficijent** (o njemu kasnije).*

*Predlaže se da se testiranje i ugađanje sustava izvode u razvojnom okruženju prije nego što se izvrše izmjene na važnim proizvodnim (živim) sustavima!.*

U nekim okruženjima, kada na sustavu imamo primarno aktivan jedan kritični program (proces), koji ako počne zauzimati previše memorije, a osigurali smo se da se taj program uredno pokreće tijekom svakog pokretanja (ili restarta) sustava, ubijanje tog programa te ponovno pokretanje sustava je dobro rješenje. Pod uvjetom da će se sustav nakon ponovnog pokretanja (*restarta*) odnosno nakon ***OOM*** stanja brzo vratiti u operativno stanje navedenog programa bez administratorske intervencije.
Iako ovo nije optimalan pristup, logika koja stoji iza toga je ta, da ako je naša aplikacija nesposobna za rad zbog toga što ju je ubio *OOM killer*, tada će ponovno podizanje sustava vratiti aplikaciju, ako se ona pokreće sâma prilikom podizanja sustava.

Ako administrator mora ručno pokretati navedenu aplikaciju/program ili ručno raditi neke dodatne radnje prije njenog pokretanja, tada nam ovakva opcija rada *OOM killera* nije previše korisna.



***Out of Memory (OOM)*** stanje se odnosi na stanje u kojem više nema slobodne virtualne memorije, koja uključuje i ***RAM*** memoriju i ***SWAP*** memoriju. Ovakvo stanje će izazvati takozvanu ***paniku*** sustava u kojem sustav više neće raditi kako treba. Tada će se pokrenuti mehanizmi o kojima ovdje govorimo, kako bi se oslobodila memorija i osigurala stabilnost sustava.

Što će se dogoditi u trenutku kada sustav više nema slobodne memorije, ovisi o opcijama koje su konfigurirane na sustavu.

Ovdje imamo cijeli nîz opcija koje su definirane u sljedećim ***sysctl*** varijablama:

- S time da prvo imamo konfiguraciju sustava virtualne memorije:
  - `vm.panic_on_oom = X` - ovdje se definira kako će se sustav ponašati kada dođe do ***OOM*** stanja. Ovdje možemo postaviti sljedeće vrijednosti za **X**, pri čemu je standardno postavljena vrijednost nula (**0**).
    - `0` - sustav ne ulazi u agresivni ***panic*** način rada, već pokreće ***OOM killer*** proceduru, koja prvo ubija programe koji su počeli zauzimati previše memorije te nastavlja s radom. Ako sustav nema dovoljno memorije niti tada, ***OOM killer*** počinje ubijati i druge programe, a ako to ne pomogne, slijedi restart.
    - `1` - odmah se ulazi u ***panic*** način rada i restart sustava **osim**, ako je problematični proces pokrenut sa *mempolicy/cpusets* postavkama, tada se pokreće ***OOM killer***, ako to može osloboditi dovoljno memorije.
    - `2` - trenutno se ulazi u ***panic*** način rada u kojem će se sustav restartati nakon broja sekundi definiranih u *sysctl* varijabli: `kernel.panic`, koja se koristi i za prethodne dvije opcije u slučaju ulaska u ***panic***.
  - `vm.oom_dump_tasks = X` - omogućuje logiranje (spremanje) poruka o ubijenim aplikacijama koje je ***OOM killer*** ubio, pri čemu se spremaju podaci o ubijenoj aplikaciji/procesu: *pid*, *uid*, *tgid*, veličina virtualne memorije, ***rss*** veličina, *pgtables_bytes*, *swapents*, *oom_score_adj*, kao i ime sâmog procesa koji je ubijen. Ovo je od pomoći kod utvrđivanja razloga zašto je ***OOM*** ubio određeni proces. Ako je ova vrijednost **1**, tada je ova funkcionalnost uključena, a **0** ako je isključena. ***Standardno je ova postavka uključena (1)***.
  - `vm.oom_kill_allocating_task = X` - ako je ovo podešeno na nulu (**0**) ***OOM** killer* funkcionalnost skenira cijelu listu pokrenutih procesa (programa) te odabire program pomoću heuristike za odabir programa koji je najprikladniji za ubijanje. On obično odabire program koji će osloboditi veliku količinu memorije nakon što je program ubijen. Ako je ovo postavljeno na jedan (**1**), ***OOM** killer* jednostavno ubija program koji je uzrokovao ***OOM*** stanje što izbjegava dugotrajnije skeniranje programa koji bi mogli biti kandidati za ubijanje. **Standardno je ovdje postavljena vrijednost nula (0).**

- A potom slijedi dio ponašanja kernela oko ***panic*** načina rada:
  - `kernel.panic = X` - (`X`) ovo je vrijeme nakon koliko sekundi će sustav čekati nakon što se dogodi kritični ***OOM***, ako je kernel u ***panic*** načinu rada (`vm.panic_on_oom=1`) prije nego se sustav restarta.
  - `kernel.panic_on_io_nmi = X` - ako je postavljeno na `0` (***zadano***), sustav pokušava nastaviti s radom, ako kernel otkrije *IOCHK* (NMO) odnosno ***NMI*** (*Non Maskable Interrupt*) koji obično označava neispravljivu hardversku pogrešku. Ako je postavljeno na `1`, sustav kod ove detekcije ulazi u ***panic*** način rada.
  - `kernel.panic_on_oops = X` - ako je postavljeno na `0`, sustav pokušava nastaviti s radom, ako kernel naiđe na ***oops*** ili ***BUG*** uvjet. Ako je postavljeno na `1` (zadano) sustav odgađa nekoliko sekundi kako bi *kernel log* servis/daemon ***klogd***, imao vremena za snimanje log datoteke prije paničnog gašenja/restarta.
  - `kernel.panic_on_unrecovered_nmi = X` - ako je postavljeno na `0` (**zadano**) sustav pokušava nastaviti s radom ako kernel otkrije *NMI* signal koji obično pokazuje neispravljivu paritetnu pogrešku ili grešku na *ECC* memoriji. Ako je postavljeno na `1`, sustav ulazi u ***panic*** način rada, pri detekciji ove greške.
  - `kernel.softlockup_panic = X` - ako je postavljen na `0` (**zadano**), sustav pokušava nastaviti s radom, ako kernel otkrije pogrešku sa tzv. *soft lock* (blokiranjem) koja uzrokuje *NMI watchdog* proces da ažurira svoj vremenski zapis za više od dvije vrijednosti definirane u `kernel.watchdog_thresh`. Ako je postavljeno na `1` sustav ulazi u ***panic*** način rada, ako detektira ovakvu grešku.

Ovdje imamo i druge opcije, poput: `kernel.hung_task_panic`, `kernel.panic_on_stackoverflow`, `kernel.panic_on_warn`, `kernel.unknown_nmi_panic`, `kernel.hardlockup_panic`

Pogledajmo i dodatne optimizacije ***OOM killera***, na razini aplikacija/programa odnosno pokrenutih procesa.
Nakon što se aplikacija pokrene i postane proces sa svojim brojem procesa (***PID***), moguće je za konkretnu aplikaciju s konkretnim ***PID*** brojem postaviti težinsku vrijednost koju uzima u obzir *OOM killer* kod ubijanja aplikacija. Naime svakoj aplikaciji je moguće postaviti vrijednosti od -16 do +15 pri čemu +15 predstavlja aplikaciju koja će imati najveći prioritet pri ubijanju od strane *OOM killera*. Postoji i posebna vrijednost **-17** koja znači kako se ovaj proces NE SMIJE dirati od strane *OOM killera*.

Ako recimo imamo proces s ***PID*** brojem `1406`, tada se u direktoriju: `/proc/1406/` nalaze sljedeće datoteke:
- `oom_score` - ova datoteka sadrži trenutni *OOM* težinski koeficijent.
- `oom_adj` – u ovu datoteku možemo definirati promjenu ***OOM*** težinskog koeficijenta, za vrijednost od -16 do +15.

Težinsku vrijednost našeg procesa (s ***PID*** brojem `1406`) je sljedeća:
```
cat /proc/1406/oom_score

6
```

Trenutno je postavljena na `6`. Sada mu postavimo težinski koeficijent na **-17** što znači da isključujemo *OOM killer* za njega:
```
echo -17 /proc/1406/oom_adj
```

Sada mu se vrijednost mijenja na manji broj (ne linearno):
```
cat /proc/1406/oom_score
0
```
Vidimo da se spustila na `0` što je standardna vrijednost kada je proces isključen iz *OOM killer* mehanizma.



Za analizu zauzeća memorije, pogledajte poglavlje:
**12.3.3.4. Naredbe *ps* i *top* za memoriju** te pogavlja:
**12.3.3.5. Naredba *vmstat*.**
**12.3.3.6. Naredba** sar**.**

***OOM*** težinski koeficijent prema kojemu ***OOM killer*** ubija procese se izračunava automatski od strane sustava, prema sljedećim pravilima:
- Bilo koji proces dovoljno nesretan da koristi ***swap*** memoriju bit će odabran da bude ubijen prvi. Za ostale, početna veličina memorije postaje jedan od faktora *OOM* težinskog koeficijenta. Polovica veličine memorije svakog procesa *djeteta* dodaje se *OOM* težinskom koeficijentu roditeljskog procesa, ako ne dijele istu memoriju.
  Stoga su procesi koji su dijeljeni, glavni kandidati za ubijanje. Ako proces ima sâmo jedno dijete proces koje nenormalno zauzima memoriju, učinit će roditelja manje poželjnim od djeteta procesa.
  Naposljetku se primjenjuju sljedeće heuristike za odabir *OOM* važnosti procesa:
  - Ako proces ima ***nice*** vrijednost iznad 0, podvostručuje mu se *OOM* koeficijent.
  - Važni procesi kojima je vlasnik *root* i procesi koji upravljaju hardverom i sustavom imaju *OOM* koeficijent koji se dijeli s četiri (4).
  - Ako se uvjet zbog kojega je *OOM* pokrenut za proces koji se ne nalazi unutar *NUMA* fizičkog procesora na kojem se pokreće *OOM killer*, *OOM* koeficijent mu se dijeli s osam (8).
    Ostatak rezultata se množi s $2^N$ gdje je *N* vrijednost postavljena u `oom_adj`.

Zatim se odabire proces s najvišom *OOM* vrijednosti/ocjenom i ubijaju se njegova djeca procesi (pôd procesi).
Sâm proces će biti ubijen u *OOM* situaciji kada sâm proces nema djecu odnosno pôd procese.

**Kako promaći proces koji ima najveći *oom* težinski koeficijent pa će u slučaju OOM, prvi biti ubijen, uz ispis *PID* broja procesa i sâmog procesa (sve u jednom retku)**?

Za ovakvu pretragu, pokrenite sljedeći nîz naredbi (u jednom retku):
```
ps -p `grep "[0-9]" -rns /proc/*/oom_score | sort -n -t\: -k3 | tail -1| awk -F\/
'{print $3}'` -o pid,cmd
```



Za više detalja o ***swap*** funkcionalnosti i ***swap*** disku, pogledajte poglavlje:
**13.8.2.1. Malo više detalja o *swap* sustavu i particiji.**
Vezano za kernel **panic** stanje i što ono sve obuhvaća, pogledajte poglavlje:
**16. Kernel Dump/Crashdump/core dump.**

**Novi mehanizmi čišćenja memorije i novi OOM killer**
Nedostatak slobodnih stranica memorija za dodjelu u trenutku kada sustav ostaje bez slobodne memorije ne znači da se takve stranice još uvijek ne mogu dodijeliti. Naime ako nema stranica memorije koje bi zadovoljile zahtjev za dodjelu, kernel će izvršiti izravno vraćanje kako bi pokušao osloboditi nešto memorije. U nekim slučajevima, izravni povrat će biti uspješan; to se događa, na primjer, ako sustav pronađe stranice koje se mogu odmah prenamijeniti odnosno osloboditi i dodijeliti novom programu. U drugim slučajevima, oslobađanje stranica memorije zahtijeva njihovo zapisivanje prema diskovnom sustavu ili čišćenju dijelova privremene memorije (*buffers* i *cache*), ali sve to traje neko vrijeme. Jedan od problema su i slučajevi kada više aplikacija dijeli određene regije memorije ili sistemske biblioteke, što je uobičajeno. Problem može biti i u tome što ne postoje stvarne granice koliko dugo može biti potrebno da se stranice koje se "povrate" stvarno i oslobode, što može trajati iznimno dugo, iz više razloga. Primjerice određene stranice memorije su trenutno zaključane jer se čeka na operaciju njihovog zapisivanja na diskovni podsustav. Ovaj problem je riješen u novijim kernelima 4.3.x, tako da je uvedena nova heuristika u prepoznavanju stvarnog **OOM** stanja te uvođenja zaštitnog mehanizma koji definira gornju granicu od šesnaest pokušaja; nakon kojih kernel odustaje od potrage i oslobađanja stranica memorije te prelazi u **OOM** način rada.

Međutim kada sustav dođe do kritičnog zauzeća memorije, nastupit će **OOM** stanje i ***OOM killer*** mehanizam neće imati druge nego da počne s ubijanjem programa prema mehanizmima koje smo opisali. Barem bi to trebao biti slučaj, ako je ***OOM killer*** mehanizam u stanju osloboditi stranice memorije kada ga kernel pozove. Kao što smo vidjeli, nije tako teško stvoriti situaciju u kojoj ***OOM killer*** ne može napraviti očekivani napredak, obično zato što je ciljani proces blokiran jer je zaključan (pr. s I/O operacijama prema diskovnom podsustavu), a zatim takva **OOM** situacija sprječava otključavanje. Nadalje ako ***OOM killer*** proces ne može nastaviti s radom, obično ne može niti izaći iz svog stanja rada i stoga, ne može osloboditi memoriju. Kao rezultat, cijeli mehanizam *OOM killera* ne uspijeva odraditi svoj zadatak (u najgorem slučaju) ili će on trajati vrlo dugo (što je vjerojatniji scenarij).
Nadalje, poglavito kod poslužiteljskih sustava s većom količinom RAM memorije i velikim brojem pokrenutih programa, problem je u tome, što ovaj proces čišćenja memorije uzrokovan OOM stanjem može trajati, ponekad čak i satima. Ne zaboravimo da potencijalno svaki program (proces) ima otvorene desetke ili stotine datoteka i dijeljenih biblioteka te koristi ekstremno veliki broj stranica memorije. Stoga, kako bi se ublažili gore navedeni efekti, u novijim kernelima **4.6+,** uveden je kernel proces imena `oom_reaper`, koji rješava sve navedene nedostatke i probleme starijeg mehanizma čišćenja memorije i ***OOM*** stanja. Ovaj kernel proces (*thread*) pokreće se automatski s pokretanjem sustava.

**Izvori informacija: (478),(479),(480),(481),(482)(483),(484),(485),(486),(487),(488),(1140),(K-4),(K-5), `man ps`, `man 5 proc`,`man sysctl`.**

---

# 12.6. Dodatne optimizacije sustava virtualne memorije

***Slijedi napredno poglavlje!***
U ovoj cjelini proći ćemo neke od osnovnih dodatnih optimizacija koje je moguće napraviti unutar sustava virtualne memorije, a koje nismo spomenuli do sada. Kako bismo vidjeli opcije koje je moguće optimizirati na razini sustava virtualne memorije, možemo pokrenuti naredbu koja će nam dati popis svih opcija (*sysctl* varijabli) i njihovih vrijednosti:
```
sysctl -a | grep vm.
```
Odnosno sve je vidljivo i kao `/proc` datoteke:
```
ls -l /proc/sys/vm
```

Ovdje postoje desetci ***sysctl*** varijabli od kojih ćemo objasniti samo sljedeće:
- `vm.admin_reserve_kbytes` - ova varijabla definira količinu slobodne memorije na sustavu koja mora biti rezervirana za korisnike i korisničke aplikacije. Standardna vrijednost je 8192kb (8MB).
- `vm.compact_memory` - ova opcije nije standardno dostupna jer je za nju potrebno kompilirati kernel s uključenom opcijom `CONFIG_COMPACTION=y`. A ako je tada uključena (postavljena na **1**) sve zone memorije će biti kompaktne tako da će uvijek cijela slobodna memorija biti dostupna u slijedu (nîzu) odnosno u blokovima memorije koji se nalaze jedan iza drugog, koliko god je to moguće[(522)]. Važno je znati da se tada aktivira mehanizam koji prema određenom algoritmu memoriju mora konstantno optimizirati, kako bi bila dostupa na navedeni način. Ova procedura je slična *defragmentaciji* diska, samo što se radi s memorijom. Ova mogućnost stoga standardno nije omogućena jer od ovakvog načina rada ima korist samo ograničeni broj aplikacija i primjena, dok za druge primjene ovaj način rada uopće nije optimalan.
- `vm.dirty_background_bytes` – ova, kao i sljedeća opcija se odnose na količinu bajta memorije koja je još ostala slobodna, a nakon i njenog zauzeća će sustav pokrenuti proces čišćenja memorije. Ako je ova opcija aktivna, onda sljedeća nije. Obično se definira sljedeća opcija.
- `vm.dirty_background_ratio` - ova opcija je dakle obično aktivna (a ne prethodna) i u njoj se definira postotak slobodne memorije koji ako se pređe odnosno, ako sustavu ostane manje slobodne memorije nego je definirano ovdje, sustav će početi čistiti memoriju i to prvo one stranice memorije označene kao „*dirty*“. Standardno je ovdje definirano 10 (što označava 10%).

- `vm.dirty_bytes` - ova opcija se odnosi na *disk page cache* međumemoriju, a definira količinu memorije koja je označena za čišćenje (*dirty*) stranica memorije pri kojoj, a kada se dosegne ova granica, sâm proces koji je generirao operacije zapisivanja na disk (*disk write*) mora i sâm pokrenuti *writeback* proceduru. Ako je ova opcija aktivna sljedeća nije.

- `vm.dirty_ratio` - ova opcija se odnosi na *disk page cache* međumemoriju i ona isključuje djelovanje prethodne opcije.. Standardno je postavljena vrijednost 20 (%). To dodatno znači i da će se pri 20% zauzeća ukupne memorije sa stranicama memorije koje su označene kao kandidati za brisanje, odnosno kao „*dirty pages*“, a koje se koriste kao diskovna međumemorija, biti stvarno i zapisane na disk. U tom slučaju se zapravo prazni disk (*cache*) međumemorija tako da se svi podaci tada zapisuju na disk. Ako i kada dođe do ovog događaja, diskovni podsustav će se usporiti zbog intenzivnijih operacija zapisivanja, te će to vjerojatno biti vidljivo u statistikama diska kao ***I/O pause*** stanja. Postavljanje ove vrijednosti znači i ograničavanje veličine diskovne *međumemorije* (*disk page cache*). Važna opcija vezana za ovu je i `vm.dirty_background_ratio`, koja je opisana gore.



Pogledajte i poglavlje:
**14.1.2 Optimizacija Page Cache sloja.**

- `vm.min_free_kbytes` - varijabla koja se koristi za prisiljavanje Linux sustava da zadrži minimalni broj kilobajta RAM memorije slobodnim. Ako ju postavite na manje od 1024 KB, vaš će sustav postati sklon blokiranju pod većim opterećenjem. Međutim postavljenjem vrijednosti prvisooko učiniti će ustav skloniji **OOM** stanju. Standardno se postavlja na vrijednost između 68.000 i 100.000 kB (tj. 68-100MB).

- `vm.max_map_count` - ova opcija se odnosi na ograničavanje koliko sektora za mapiranje memorije pojedini program (proces) može koristiti. Ne zaboravimo kako programi alociraju memoriju u blokovima, koji mogu biti veličine 8kb, 16kB, 32kB, 64kB i tako dalje, a svaki blok se naravno u konačnici sastoji od stranica memorije veličine 4kB. Ovdje se radi o tome da možemo ograničiti broj tih blokova memorije, koliko ih se može alocirati po pojedinom procesu, što nam ne daje detalje o tome koja je to količina memorije jer kako smo rekli blokovi (engl. *Memory chunks*) mogu biti različite veličine, pa je i ukupna stvarna veličina alocirane memorije tada različita. Možemo reći da se ovdje radi o resursima u broju blokova memorije koji se mogu alocirati, a ne o MB/GB memorije koji su alocirani. Standardna vrijednost je 65535 što je za većinu programa dovoljno. Ovo eventualno može biti ograničavajuće za posebne *debuggere* na ekstremno velikim sustavima.

- `vm.memory_failure_early_kill` – u ovoj opciji koja može biti uključena (**1**) ili isključena (**0**), definira se kako sustav treba reagirati u slučaju kada se detektira greška u RAM memoriji, a koja je otkrivena od strane memorijskog kontrolera (hardverske komponente unutar procesora) i koju kernel ne može popraviti.

- Naime greške na RAM memoriji koje nisu mogle biti riješene na razini hardvera; ako recimo imamo ***ECC RAM*** memoriju, koja može podnijeti određene greške u RAM memoriji i pripada ovoj kategoriji. U nekim slučajevima poput onih u kojima za problematičnu stranicu memorije imamo važeću kopiju podataka na disku, kernel će rješavati ovaj problem transparentno bez utjecaja na bilo koju aplikaciju. Ali ako postoji problem kada nemamo kopije potrebnih podataka, kernel će ubiti proces koji se našao u problemu, sve u cilju kako bi spriječio bilo kakve dodatne probleme koji bi mogli uslijediti. Ova opcija je standardno postavljena na (**0**).

  Pri tome vrijednosti:
  - `0` (isključeno) znači kako će kernel isprazniti problematičnu stranicu RAM memorije i u slučaju da neki proces koristi tu stranicu memorije, kernel će ga ubiti. Ubijanje procesa se odrađuje preko sistemskog poziva (`BUS_MCEERR_AO`) pa proces može i reagirati na tu grešku, prije nego bude ubijen.
  - `1` (uključeno) znači kako će kernel ubiti sve procese koji imaju korumpiranu stranicu memorije, bez njene kopije na disku iz koje bi se podaci mogli restaurirati, i to čim se otkrije korupcija podataka.
    Važno je znati kako ovo nije podržano za nekoliko vrsta stranica memorije: kao što su one koje koriste interno dodijeljeni podaci kernela ili zamjenska privremena memorija (*swap*). Međutim ovaj mehanizam radi za većinu drugih stranica memorije.

- `vm.memory_failure_recovery` – u ovoj opciji definira se, a može biti uključena (**1**) ili isključena (**0**), kako sustav treba reagirati kada se otkrije greška u RAM memoriji, koja se može popraviti. Ovdje imamo dvije mogućnosti:
  - `0` (isključeno) znači kako će kernel odmah otići u ***panic*** načina rada.
  - `1` (uključeno) znači kako će kernel pokušati povratiti podatke (ovo je standardno postavljeno).

Dokumentacija kernela za sustav virtualne memorije je dostupna je na lokalnom računalu sa sljedećim nizom naredbi:

```
less /usr/share/doc/kernel-doc-`uname -r| awk -F- '{print
$1}'`/Documentation/sysctl/vm.txt
```



Detalje možemo vidjeti na poveznici:
**https://www.kernel.org/doc/Documentation/sysctl/vm.txt**
Odnosno detalji su vidljivi na izvoru informacija (157**).**

**Izvori informacija: (157),(491),(519),(520),(521),(522),(523),(524),** **`man sysctl`**,**`man 5 procfs`**.

### 12.6.1. Drugi granični slučajevi

U nekim posebnim slučajevima postoji mogućnost da program virtualno alocira dio memorije (vidljive kao `VIRT`) na način u kojem u nekim slučajevima može doći do prevelikog stvarnog zauzeća memorije iako je ona memorija koja je u stvarnoj upotrebi (`RES`) vidljiva kao drastično manja.



Ovdje se referenciramo na memoriju vidljivu naredbama **`top`** i **`ps`** opisanim u poglavlju:
**12.3.3.4. Naredbe *ps* i *top* za memoriju.**

Stoga pokrenimo naredbu `top` i pogledajmo naša prva dva procesa/programa (naziva: `Router-1` i `Router-2`):

```
top - 11:56:28 up 12 days,  2:23,  4 users,  load average: 0.79, 0.88, 0.76
Tasks: 419 total,   1 running, 418 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.4 us,  0.8 sy,  0.0 ni, 97.7 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 57508172 total,  9164096 free, 42760912 used,  5583164 buff/cache
KiB Swap: 16383996 total, 16327540 free,    56456 used. 14090416 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4588 root      20   0   11.4g   1.6g  14588 S  10.9  2.9 490:14.24 Router-1
 2511 root      20   0   12.3g 809716  14820 S   6.3  1.4 143:31.28 Router-2
```

Radi se o ***Java*** programima koji su pokrenuti s parametrom `-Xmx2048M` s kojim se oni ograničavaju da ne mogu koristiti više od 2GB RAM memorije, kao mehanizam zaštite sustava od **OOM** stanja koje može prouzročiti aplikacija bez ovakve zaštite. Stvarno iskorištenje memorije je vidljivo pod stupcem `RES` u kojem je sve unutar tih granica. Ovaj program je pokrenut i s opcijom `-Xms2048M` s kojom se nalaže sustavu da odmah alocira 2GB memorije za ovaj program u trenutku njegovog pokretanja, kako on u momentu kada mu stvarno i zatreba toliko memorije (2GB) ne bi morao čekati na sustav, što je važno kod visoko performantnih programa.



Pogledajte poglavlje:
**12.4. Anatomija programa u memoriji.** ← unutar navedenog poglavlja, pogledajte cjelinu: „**Posebni slučajevi**".

Dakle vidimo kako ovi programi (procesi) ne koriste više od 2GB rezidentne memorije (`RES`) jer smo to ograničili s gore navedenim prekidačem `-Xmx2048M`; što se zapravo odnosi na JAVA *HEAP* memoriju koja unosi ograničenje[(637)] unutar ovih granica. Međutim oba procesa su prijavila kako im je potrebno (`VIRT`) virtualnog dijela memorije; za prvi program: `11.4g` (11.4GB) a za drugi: `12.3g` (12.3GB) što nije nužno normalno odnosno standardno ponašanje programa u radu.
Naime konkretno se radi o programima koji su vršno opterećeni i optimizirani za rad s ekstremnim brojem konekcija.
Oni istovremeno zapisuju statistike u nekoliko log datoteka s informacijama o svakoj od desetaka tisuća ili više, otvorenih konekcija prema njima. Nadalje oni koriste, otvaraju i zatvaraju veliki broj programskih nîti unutar sâmog programa (procesa).

## VIRT statistika memorije

Ne zaboravimo kako `VIRT` označava virtualnu količinu memorije koju program (proces) može koristiti. Ona je zbroj memorije koju zapravo proces i koristi (`RES`) uz datoteke na disku koje je proces mapirao, a ovo se odnosi uglavnom na dijeljene (`SHR`) biblioteke i otvorene datoteke, ali i dijeljenu memoriju koja se koristi za komunikaciju između procesa, kao i određene međumemorije (*buffer/cache*) koje aplikacija može koristiti. Vezano za dijeljene biblioteke moramo znati kako se one u pravilu ne učitavaju u cijelosti u memoriju, već se učitavaju samo njihovi dijelovi s funkcijama koje stvarno i pozivamo i koristimo. Slično je i s otvorenim datotekama. U slučaju kada primjerice imamo dijeljenu biblioteku veličine 10MB, a od nje koristimo funkcije koje zauzimaju samo 1MB, u memoriju će se stvarno i učitati sâmo 1MB, što će biti vidljivo u `RES` statistici. Međutim cijela takva datoteka (po veličini) će se prikazivati u statistici koju vidimo pod `VIRT`. Druga vrsta datoteka koje ulaze u ovu statistiku su sve ostale datoteke koje ne moraju biti dijeljene već se njima sâmo pristupa jer se iz njih dohvaćaju ili zapisuju neki podaci. Stoga sve te datoteke mogu u punoj veličini završiti u `VIRT` statistici (u datom trenutku kada se koriste).

Dodatno u `VIRT` se upisuje i veličina *SWAP* memorije, ako se i ona koristi. Dakle `VIRT` statistika sadrži zbroj svih navedenih memorija uz dodatak kako nam ova veličina često govori koliko maksimalno memorije program može alocirati u datom trenutku. Sve navedeno možemo promatrati i na drugačiji način, te reći kako to ne znači da se sva ova memorija (`VIRT`) stvarno i koristi, kako smo i vidjeli, već koliko bi je program (proces) trenutno, maksimalno mogao koristiti, ako mu zatreba, uz dodatak upotrebe međumemorija koje mogu biti pripremljene, ali ne i u upotrebi (ili ne u cijelosti u upotrebi).

#### Drugi dio priče

Novije inačice *GNU C biblioteka* (*glibc*) koriste više memorijskih spremišta (Engl. *multiple memory pools*) koje sustav za dodjeljivanje memorije (***malloc***) može dodijeliti programu kao predmemoriju odnosno kao njegovo predmemorijsko spremište koje može, ali i ne mora stvarno biti iskorišteno. Vezano za *glibc*; on implementira, kako standardne ***C*** funkcije tako i *POSIX* funkcije, ali i sistemske pozive koje koriste svi programi na sustavu kako bi primjerice od sustava tražile resurse (CPU, RAM, DISK, …) i to tijekom cijelog životnog vijeka programa (procesa). Ovakav rad u kojem se za program dodjeljuje više memorijskih spremišta je posebno važan za programe koji koriste više programskih nîti u radu i traže više performanse, jer tada ovakvo korištenje više spremnika ove (pred) memorije može poboljšati ukupnu učinkovitost pokrenutog programa.



*Memorijska spremišta (engl.* ***memory pools****) nazivaju se* ***arene****, a implementirane su u* ***C*** *biblioteci* `arena.c`.

Međutim uz neke obrasce ovakve dodjele memorije ovo poboljšanje performansi može doći uz cijenu nešto veće stvarne potrošnje memorije programa odnosno aplikacije. To posebno može biti slučaj s programima koji koriste, stvaraju i uništavaju mnogo programskih nîti i dodjeljuju puno memorije tim programskim nîtima. Analogno tome, programi koji konstantno koriste veliki broj programskih nîti, odnosno koji stalno stvaraju i zatvaraju nove programske nîti, mogu zauzeti popriličnu količinu, prvo virtualnog dijela memorije (`VIRT`), a kasnije čak i stvarne memorije (`RES`). Standardno svi novi ***glibc*** (***>v.2.10***) za 64. bîtne sustave alociraju maksimalno do 64MB za svako ovo pojedino memorijsko međuspremište. Sustav standardno za aplikacije postavlja posebnu varijablu koja se čita i po kojoj se postavlja broj ovih memorijskih spremnika po svakom pojedinom programu odnosno procesu, na vrijednost: **`MALLOC_ARENA_MAX`**=***Broj CPU jezgri*** **× 8**



Druge varijable i opcije ***malloc***-a možete vidjeti na: http://man7.org/linux/man-pages/man3/mallopt.3.html

To u našem slučaju, za sustav s 32 CPU jezgre znači; `MALLOC_ARENA_MAX=32 x 8 =` **`256`** memorijskih spremišta.
Pošto radimo na 64 bîtnom sustavu to znači kako se za pojedini (ovakav) proces koji koristi puno programskih niti, može rezervirati do maksimalno 16GB *`VIRT`* memorije za memorijska spremišta*: 64 MB x 256 = 16 384 MB =* ***16 GB***.
Ponavljamo: to ne znači kako je sva ta memorija međuspremišta stvarno u upotrebi, već kako je sâmo rezerviran taj adresni prostor; stoga se ona i prikazuje u dijelu `VIRT` statistike memorije. Pošto ***Java*** standardno alocira memoriju preko ovog sustava, jasno je kako će dio memorije vidljiv kao `VIRT` biti znatno veći jer se u njegovim statistikama vidi ova memorija, a koja još i ne mora biti u stvarnoj upotrebi u datom trenutku.
Pogledajmo koliko programskih nîti koristi naš prvi ***Java*** proces ***PID*** broja `4588` s naredbom `ps` na sljedeći način:

```
ps -T -p 4588 | wc -l

126
```

Ovdje konstantno imamo `126` programskih nîti, što je priličan broj, koji ulazi u gore navedenu kategoriju.



Dalji koraci optimizacije su vrlo specifični za našu primjenu (ili druge slične njoj) te ih nemojte raditi prije nego dobro proučite vaš konkretan slučaj!.

Mi ćemo smanjiti maksimalni broj memorijskih spremišta na po dva, za svaku CPU jezgru. Dakle pošto imamo 32 CPU jezgre, bit će ih 64. S time smo dio memorije (`VIRT`) za memorijska spremišta smanjili na maksimalno: 64 x 64MB= 4096MB odnosno **4GB**. To znači kako moramo eksportirati sljedeću sistemsku varijablu:

```
export MALLOC_ARENA_MAX=64
```

I sada moramo restartati naše („*problematične*“) programe. I sada pogledajmo stanje njihovog rada (skraćeni ispis naredbe `top`):

```
  PID USER       PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
54314 root       20   0 7962264   1.0g  14508 S   4.3  1.9   2:12.86 Router-1
52150 root       20   0 8040860   1.1g  15572 S   3.3  2.1   1:50.86 Router-2
```

Vidimo kako smo dobili željeno; trenutno rezidentno zauzeće memorije (`RES`) je 1GB odnosno 1.1GB za drugi program, a zauzeće virtualnog dijela memorije (`VIRT`) je `7962264`KB odnosno 7.9GB za prvi program i `8040860`KB odnosno 8GB za drugi s čime smo očito smanjili zauzeće virtualnog dijela memorije (`VIRT`) kako smo i htjeli.
Ne zaboravimo kako se ovdje računaju `RES`, `SHR` i drugi dijelovi memorije. Varijable s kojima je program/proces pokrenut (poredane u nîzu, bez razmaka), vidimo u posebnoj datoteci: `/proc/PID/environ`, konkretno za ovaj proces (`54314`) u datoteci: `/proc/54314/environ`



*Ova optimizacija je nepotrebna za većinu standardnih odnosno „****normalnih“*** *programa!.*



Vezano za prebacivanje izvršavanja između procesa i/ili programskih nîti, pogledajte poglavlje:
**9.3.1. Context switching.**

**Izvori informacija: (631),(632),(633),(634),(635),(636)**,**`man ps`**,**`info bash export`**,**`man 7 libc`**.

## 12.6.2. Upotreba RAM diska

***Slijedi napredno poglavlje!***
Iako smo o inicijalnom RAM disku odnosno ***initrd*** ili ***initramfs*** tehnologijama njegove upotrebe u trenutku inicijalizacije sustava govorili u poglavlju: **10.7.1.2.1.**, sada ćemo govoriti o mogućnosti da sâmi kreiramo svoj RAM disk u trenutku kada nam to odgovara. Naime RAM disk koji se još naziva i RAM pogon predstavlja blok memorije sa slučajnim pristupom (primarna RAM memorija) koju softver računala odnosno sâm Linux promatra kao da se radi o diskovnom pogonu odnosno disku (bilo mehaničkom ili SSD).

Performanse RAM diska općenito su za red veličine brže od ostalih oblika medija za pohranu, kao što su SSD, tvrdi disk, a pogotovo od tračnih uređaja ili optičkih pogona odnosno diskova. Ovo povećanje performansi nastaje zbog više faktora, uključujući vrijeme pristupa, maksimalnu propusnost i u konačnici vrstu datotečnog sustava o kojoj ovdje nećemo govoriti. Vezano za vrijeme pristupa datoteci ono se uvelike smanjuje jer je RAM pogon poput SSD pogona (diska) pa nema pomjeranja glava za čitanje ili zapisivanje i drugih mehaničkih radnji.

Dakle RAM diskovi mogu pristupiti podacima sâmo s memorijskom adresom određene datoteke, bez pomicanja, poravnanja ili pozicioniranja glave za čitanje/zapisivanje i slično. Maksimalna propusnost RAM diska ograničena je brzinom RAM memorije, na što utječu brzine memorijske sabirnice i CPU-a računala. U konačnici definitivno možemo zaključiti kako je brzina RAM diska drastično veća čak i od najbržih NVMe diskova. Pod Linuxom imamo nekoliko tehnologija koje nam nude mogućnost korištenja RAM diska. Prva je upotrebom takozvanog virtualnog datotečnog sustava ***tmpfs*** koji se u pravilu koristi za snimanje podataka iz jednog programa u RAM memoriju te kasnije mogućnosti čitanja tih podataka iz drugog programa, što predstavlja najbržu moguću metodu razmjene podataka među programima (aplikacijama). Ovakva metoda razmjene podataka se koristi u određenim primjenama, kao što je primjerice kod upotrebe virtualizacije, posebnih baza podataka i slično.

Standardno Linux za ovu potrebu kreira posebnu *device* datoteku imena `/dev/shm` koja je zapravo RAM disk, standardne veličine polovine dostupne RAM memorije. Međutim ova posebna datoteka je u točki montiranja zapravo direktorij (mapa), unutar koje obično svaka aplikacija koja ju koristi, kreira svoju strukturu pôd direktorija (pôd mapa) i datoteka.

Je li ovaj RAM disk (`/dev/shm/`) u upotrebi, možemo provjeriti sa sljedećom naredbom:
```
mount | grep /dev/shm
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,seclabel)
```

Vidimo da imamo montiran ***tmpfs*** datotečni sustav u posebnu datoteku imena `/dev/shm`. Sustav tijekom pokretanja, u trenutku kada inicijalizira ovaj datotečni sustav i povezuje ga s posebnom datotekom `/dev/shm` određuje i njegovu veličinu.

Međutim mi možemo i mijenjati ovu veličinu u radu. Primjerice ako imamo poslužitelj sa 256GB RAM memorije i zbog posebne baze podataka ili ako poslužitelj primjerice koristimo za virtualizaciju (***KVM+QEMU***), za koju znamo da intenzivno koristi ovu posebnu `/dev/shm` datoteku, želimo povećati ovu veličinu na 32GB, što možemo postići na sljedeći način:
```
mount -o remount,size=32G /dev/shm
```

Ove promijene možemo snimiti trajno u datoteku `/etc/fstab` pa bi onda u njoj imali sljedeći unos:
```
none      /dev/shm        tmpfs    defaults,size=32G        0 0
```



***TMPFS*** kao *RAM* disk se koristi i za još jednu posebnu automatsku točku montiranja: `/run` koja je obično odredište simboličke veze direktorija `/var/run` u koji programi zapisuju svoje statusne informacije i slično (pr. ***PID*** datoteke) **(925)**.

Međutim možemo i sâmi kreirati dodatni ***tmpfs*** datotečni sustav i montirati ga u željeni direktorij. Ovdje treba biti vrlo oprezan jer je ekstremno važno ograničiti veličinu ovog datotečnog sustava koji bez ograničenja može *popuniti* cijelu RAM memoriju.
```
mkdir /mnt/mytmpfs
mount -t tmpfs -o size=10M tmpfs /mnt/mytmpfs
```
U navedenom primjeru kreirali smo točku montiranja `/mnt/mytmpfs` koja koristi ***tmpfs*** datotečni sustav koji je ograničen na 10MB. Pogledajmo sada kako to izgleda:
```
df -h /mnt/mytmpfs/
Filesystem      Size  Used Avail Use% Mounted on
tmpfs            10M     0   10M   0% /mnt/mytmpfs
```
Vidimo kako je naš novi ***tmpfs RAM*** disk veličine 10MB kako smo i željeli.
Važno je razumjeti i nekoliko činjenica o ***tmpfs*** datotečnom sustavu:

- Ovaj datotečni sustav može početi koristiti *SWAP* memoriju, ako mu ponestane prostora.
- Ovaj datotečni sustav zauzima samo onoliko (RAM) memorije koliko podataka i sadrži.
- Nakon što je montiran, a u nekom trenutku ga demontiramo i montiramo automatski (`mount -o remount`), ako mu primjerice mijenjamo veličinu, sadržaj (podaci) u njemu će ostati sačuvani (u pravilu).

## RAM Disk kao blok uređaj

Na novijim kernelima (3.10+) moguće je kreirati i poseban blok uređaj koji se veže za RAM memoriju pa na taj način možemo kreirati RAM disk željene veličine.
Prvo kreirajmo *blok* RAM disk veličine 500 MB:

```
modprobe brd rd_nr=1 rd_size=524288
```

Sada ćemo naš novi RAM disk vidjeti kao blok uređaj `/dev/ram0`

```
ls -al /dev/ram0
brw-rw----. 1 root disk 1, 0 Mar 30 12:41 /dev/ram0
```

U slučaju da RAM disk uređaj (**`/dev/ram0`**) ne postoji, možemo ga kreirati i sami:

```
mknod -m 660 /dev/ram0 b 1 0
```

I još mu moramo promijeniti ovlasti to jest vlasnika i grupu kojoj treba pripadati:

```
chown root:disk /dev/ram0
```

Potom ga možemo formatirati (ne moramo raditi particiju na njemu).
Mi ćemo ga formatirati sa *EXT4* datotečnim sustavom:

```
mkfs.ext4 /dev/ram0
```

Umjesto gornje naredbe, mogli smo uvesti i ograničenje veličine i na datotečni sustav; primjerice na 500MB

```
mkfs.ext4 /dev/ram0 500M
```

I zatim ga možemo montirati u direktorij (mapu) koji ćemo sada kreirati:

```
mkdir /mnt/ramdisk
mount /dev/ram0 /mnt/ramdisk
```

I od tog trenutka imamo poseban blok uređaj koji predstavlja RAM disk, koji možemo normalno koristiti (do restarta računala).



**Ne zaboravite da RAM disk gubi sve podatke nakon gašenja računala!**

Ako ga želimo trajno montirati (*mountati*), u `/etc/fstab` datoteku možemo dodati unos poput slijedećeg:

```
/dev/ram0        /mnt/ramdisk    ramfs   size=500M
```



Jedna vrsta RAM diska, konkretno inicijalni RAM disk odnosno ***initrd*** ili ***initramfs*** tehnologije koriste se u trenutku inicijalizacije operativnog sustava, nakon inicijalizacije kernela, u trenutku prije pokretanja ostatka sustava.



O ***initrd*** i ***initramfs*** smo govorili u poglavlju:
**10.7.1.2.1. initrd i/ili initramfs i inicijalizacija sustava.**

**Izvori informacija: (859),(860),(861),(862),(863),(864),(865),(924),(925),(K-12)**,`man modprobe`,`man mkfs`,`man 5 fstab`,`man mkdir`,`man mount`.

# 13. Diskovi, particije i RAID polja

U ovom poglavlju upoznat ćemo se s vrstama i namjeni diskova, diskovnim particijama i *RAID* poljima diskova.

## 13.1. Vrste tvrdih diskova i standardi

Osnovna podjela tvrdih diskova je prema njihovom sučelju za povezivanje s našim računalom ili poslužiteljem te sâmoj namjeni diskova. Ova podjela nam je važna zbog razumijevanja načina povezivanja tvrdog diska s Linuxom. Nemojmo zaboraviti kako *Linux* kao i *Unix* operativni sustavi pristupaju svim uređajima pa tako i tvrdim diskovima na vrlo niskoj razini. Prema tome, važno je prepoznati sučelje s kojim se tvrdi disk povezuje s računalom jer ga mi koristimo za povezivanje s Linuxom. Drugim riječima Linux koristi tvrdi disk prema njegovom sučelju, jer se povezuje direktno na sučelje tvrdog diska.

### 13.1.1. Podjela prema vrsti diskova

Osnovna podjela tvrdih diskova prema vrsti sučelja za povezivanje je vidljiva u tablici:

| **Ime sučelja** | **Opis** | **Broj nožica (pinova) na tvrdom disku** | **Broj žila u kabelu** | **Maksimalan broj diskova koji se može spojiti na jedan kabel/kanal** |
|---|---|---|---|---|
| (P)ATA | Paralelni ATA (engl. AT *Attachment*) | 40 (44 za 2.5” diskove) | 40, 44 ili 80 | 2 |
| SATA | Serijski ATA (engl. *Serial ATA*) | 7 | 7 | 1 |
| SCSI | Engl. *Small Computer System Interface* ili tzv. Paralelni SCSI | 50, 68 ili 80 | 50, 68 ili 80 | 15 |
| SAS | Serijski SCSI (engl. *Serial SCSI*) | 7 | 7 | 1 |

Pogledajmo i osnovne karakteristike raznih diskovnih sučelja i generacija/vrsta diskova koji se spajaju na njih, u drugoj tablici:

**ATA Standardi** (navodimo ih samo nekoliko):

| **Sučelje** | **Standard** | **Maksimalna brzina prijenosa podataka i novi standardi** |
|---|---|---|
| (P)ATA | ATA-1 | PIO-Mod 0: 3,3 MB/s; PIO 1: 5,2 MB/s; PIO 2: 8,3 MB/s.<br>Single Word DMA Mod 0: 2,1 MB/s, DMA single 1: 4,2 MB/s, DMA single 2: 8,3 MB/s.<br>Multi Word DMA Mod 0: 4,2 MB/s. |
| (P)ATA | ATA/ATAPI-6 | Uvodi se Ultra-DMA-100:100 MB/s.<br>Uvodi se 48.bitna LBA shema adresiranja diskova (**LBA-48**).<br>Uvodi se “*Automatic Acoustic Management*” (**AAM**). |
| (P)ATA | ATA/ATAPI-7 | **ATA** se sada službeno zove **PATA**, jer se pojavljuje **SATA** kao standard.<br>Uvodi se *UltraDMA* 6: 133 MB/s (ATA/133). |

**SATA Standardi**

| **Službeni naziv standarda** | **Neslužbeni naziv** | **Brzina prijenosa podataka** |
|---|---|---|
| Serial ATA 1,5 Gbps | SATA I | 1,2Gbps (150MB/s) |
| Serial ATA 3 Gbps, SATA Revision 2.x | SATA II, SATA 300 | 2,4Gbps (300MB/s) |
| Serial ATA 6 Gbps, SATA Revision 3.x | SATA III, SATA 600 | 4,8Gbps (600MB/s) |
| SATA Revision 3.2 | SATA 3.2 | 15,8Gbps (1.975 MB/s) |

**Postoji i noviji način povezivanja diskova pomoću takozvanog NVMe standarda, spajanjem na *PCI express* sabirnicu:**

| **Veza prema sabirnici** | **Brzina prijenosa podataka** |
|---|---|
| ***NVMe*** *preko* ***M.2*** *konektora za PCI express 3.0* ***×4*** *linka* | 3,9 GB/s (3.938 MB/s) |
| ***NVMe*** *preko* ***M.2*** *konektora za PCI express 4.0* ***×4*** *linka* | 7,9 GB/s (7.876 MB/s) |
| ***NVME*** *na PCI express sabirnici* | *Ovisno o generaciji PCI express sabirnice i njenoj širini* *(**x1, x2, x4, x8 ili x16**)*<br>*Pogledajte poglavlja:* ***10.1.1. Matična ploča, CPU, chipset i sabirnica(e).*** i **13.3.2. NVMe diskovi.** |



Za način rada **SSD** i **NVMe** diskova te njihov način povezivanja s računalom, pogledajte poglavlja:
**13.3. SSD diskovi.**
**13.3.2. NVMe diskovi.**

Postoji i takozvani *SCSI* standard za diskove i pripadajuće disk kontrolere. Pogledajmo ih u tablici.

**Pogledajmo i specifikacije SCSI i SAS standarda:**

| SCSI Standard | Brzina prijenosa podataka | Vrsta kabela (broj žila/pinova) |
|---|---|---|
| *SCSI (SCSI-1)* | **5 MB/s** | 50 |
| *Wide SCSI (SCSI-2)* | **10 MB/s** | 68 |
| *Fast SCSI (SCSI-2)* | **10 MB/s** | 50 |
| *Fast Wide SCSI (SCSI-2)* | **20 MB/s** | 68 |
| *Ultra SCSI (SCSI-3)* | **20 MB/s** | 68 |
| *Ultra Wide SCSI (SCSI-3)* | **40 MB/s** | 68 |
| *Ultra 2 SCSI* | **40 MB/s** | 50 |
| *Ultra 2 Wide SCSI* | **80 MB/s** | 68 ili 80 |
| *Ultra 3 - Ultra 160 SCSI* | **160 MB/s** | 68 ili 80 |
| *Ultra 4 - Ultra 320 SCSI* | **320 MB/s** | 68 ili 80 |
| *Ultra 5 - Ultra 640 SCSI* | **640 MB/s** | 68 ili 80 |

Koje su razlike i sličnosti navedenih sučelja i standarda:

- ***SCSI*** diskovi su se koristili (i još uvijek se koriste) uglavnom za poslužitelje. Postoje razne inačice brzina i sučelja. Sučelja su uglavnom s 50 nožica/vodova (starije varijante) te sa 68 ili 80 nožica (novije varijante). Na svaki *SCSI* kabel (kanal) je moguće spojiti do 15 diskova, zbog toga i česti naziv paralelni *SCSI*. **SCSI-3** standard uvodi modularnu arhitekturu s različitim slojevima za transport, fizičko sučelje i naredbe. SCSI-3 se stoga koristi i u *Fibre Channel Protocol* (**FCP**), koji omogućuje prenošenje SCSI naredbi preko ***Fibre Channel*** mreža u ***SAN*** sustavima. I ***iSCSI*** koristi SCSI-3 naredbe za ***SAN*** (*Storage Area Networks*) sustave koji koriste *Ethernet* mreže i IP protokol.
- ***SAS*** (serijski SCSI) - mijenja "stari" *SCSI* standard. *SAS* diskovi imaju isti konektor kao i *SATA* diskovi te se na jedan kabel spaja sâmo jedan disk, a odatle naziv serijski. *SAS* osim standardnog konektora (SFF-8492) koji je isti kao i *SATA* te podržava još nekoliko vrsta konektora: *SFF-8484, 8485, 8470, 8087, 8088*.

| SAS Standard | Brzina prijenosa podataka |
|---|---|
| *Serial Attached SCSI (SAS)* **SAS-1** | **300 MB/s** |
| *Serial Attached SCSI (SAS)* **SAS-2** | **600 MB/s** |
| *Serial Attached SCSI (SAS)* **SAS-3** | **1,2 GB/s (1.200 MB/s)** |
| *Serial Attached SCSI (SAS)* **SAS-4** | **2,4 GB/s (2.400 MB/s)** |

- ***ATA*** diskovi su se koristili za "obična" računala. Koristili su se kabeli sa 40, 44 ili 80 žila, ali konektori na sâmom disku su ili s 40 za 3.5" diskove ili s 44 pin-ova (nožica/konektora) za 2.5" diskove. Na jedan ATA kabel moguće je spojiti do dva diska, prema *Master* i *Slave* principu. Moguće je imati više ATA kanala za spajanje više ATA kabela na disk kontroler. Prioritet tijekom pokretanja sustava ima primarni kanal, potom sekundarni itd. Ovaj način povezivanja je paralelni prema logici rada.

- ***SATA*** je naslijedio odnosno zamijenio **ATA** standard. Na jedan kabel se sada može spojiti sâmo jedan disk. Kao što je rečeno, konektor je isti i za **SATA** i za **SAS** (serijski SCSI), ali to ne znači da ćete moći **SAS** disk spojiti na **SATA** kontroler i obratno, ako to nije podržano od strane sâmog disk kontrolera. **SATA** standardno preko **AHCI** sučelja (ako je uključeno u **BIOS**u) podržava dvije vrlo važne značajke: **NCQ** te spajanje i odspajanje diskova na živo.

- ***NVMe*** je najnoviji standard za spajanje diskova, obično preko **PCI express** sabirnice ili **M.2**. sučelja prema sabirnici, koje nam daje znatno manje kašnjenje, veću propusnost te druge napredne značajke (pogledajte poglavlje: **13.3.2.**).

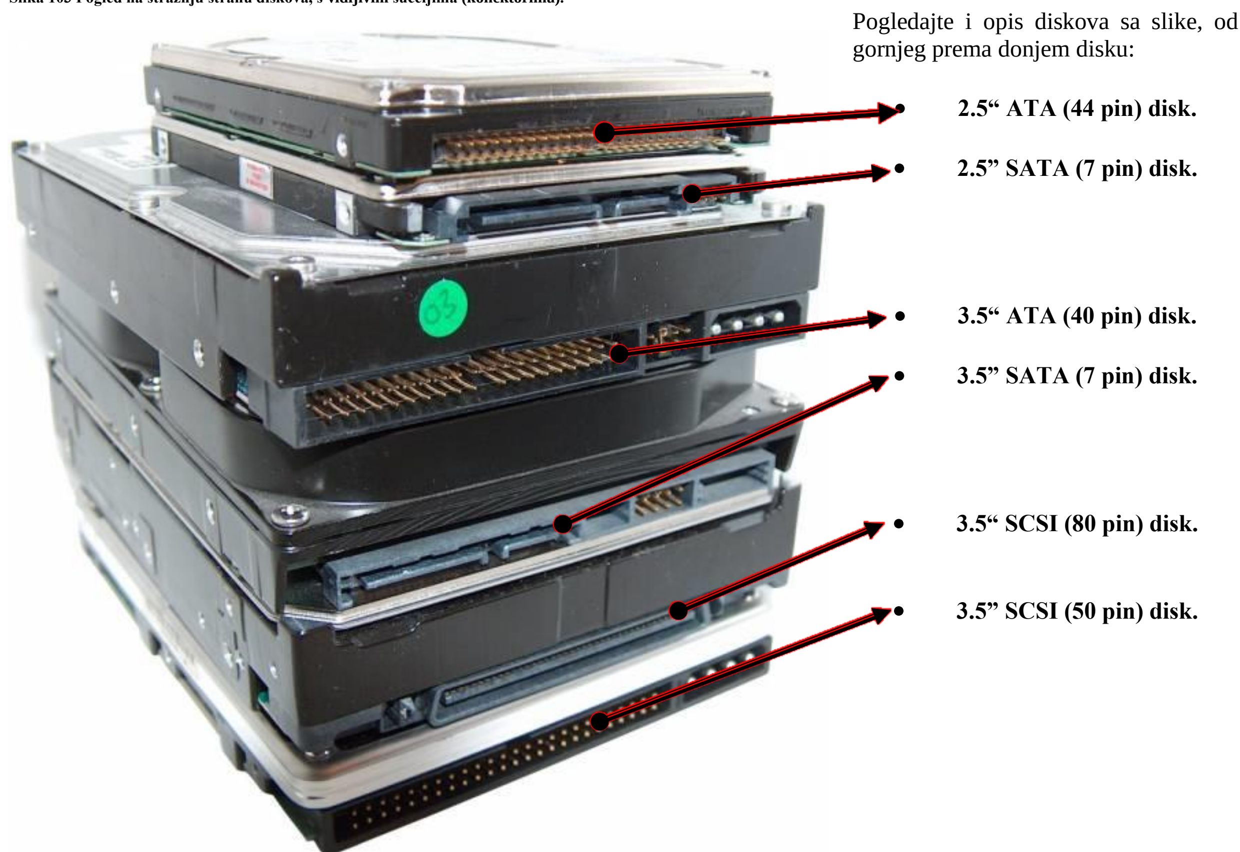
Način označavanja diskova prema Linuxu, ovisno o njihovom sučelju, vidljiv je u tablici:

| Vrsta diska/konektora | Naziv u Linuxu |
|---|---|
| **ATA** - primarni kanal - *master* disk | `/dev/hda` |
| **ATA** - primarni kanal - *slave* disk | `/dev/hdb` |
| **ATA** - sekundarni kanal - *master* disk | `/dev/hdc` |
| **ATA** - … …x. | `/dev/hd..x.` |
| **SATA** ili **SCSI** - prvi kanal ili disk | `/dev/sda` |
| **SATA** ili **SCSI** - drugi kanal ili disk | `/dev/sdb` |
| **SATA** ili **SCSI** - treći kanal ili disk | `/dev/sdc` |
| **SATA** ili **SCSI** - ..x. kanal ili disk | `/dev/sd..x.` |

Izgled utičnica (konektora) raznih vrsta diskova, sa stražnje strane diska, vidljiv je na slici 103 dolje.

**Slika 103 Pogled na stražnju stranu diskova, s vidljivim sučeljima (konektorima).**

Pogledajte i opis diskova sa slike, od gornjeg prema donjem disku:






**SCSI** naredbe (**SCSI-3**) se koriste i u ***SAN*** sustavima za pohranu, stoga pogledajte i poglavlja:
**13.12.1.1. iSCSI.**
**13.12.1.2. Fibre Channel (FC).**

**Izvori informacija: (994),(995),(996),(997),(998),(999).**

## 13.1.2. Podjela prema namjeni diskova

Diskove prema namjeni možemo u grubo podijeliti u dvije grupe:

- Diskove za profesionalnu upotrebu odnosno za poslužitelje ili namjenu 24/7 (365 dana godišnje, bez gašenja).
- Diskove za "kućnu" ili uredsku upotrebu.

Ovdje se prvenstveno radi o pouzdanosti, ali često i o brzini rada tvrdog diska. Važan parametar svakog diska vezan za brzinu rada je i broj okretaja samog diska, te često imamo sljedeće rotacijske brzine:

- 5.400 o/min [okretaja u minuti] (Engl. ***rpm*** (*Rounds Per Minute*))
- 7.200 o/min [okretaja u minuti]
- 10.000 o/min [okretaja u minuti]
- 15.000 o/min [okretaja u minuti]

Što su veće rotacijske brzine ploča diskova (okretaja/minuti) to je veće i zagrijavanje, ali i brzina prijenosa podataka. Usporedit ćemo tri tvrda diska tvrtke ***Western Digital***, oba sa SATA sučeljem, ali jedan za kućnu upotrebu (WD Black) i dva za profesionalnu primjenu: ***WD RE*** i ***WD SE (****u tablici dolje*):

| **Funkcionalnost / Model diska** | **WD Black (FZEX ili FAEX)** | **WD RE (FYYZ ili FSYZ)** | **WD SE (F9YZ)** |
|---|---|---|---|
| **MTBF (sati)** | Nedefinirano | 1.200.000 | 1.000.000 – 1.200.000 |
| **Load unload Cycles** | 300.000 | 600.000 | 300.000 |
| **Broj nepopravljivih grešaka prema bitovima pročitanih podataka** | Manje od 1 na $10^{14}$ bitova | Manje od 1 na $10^{15}$ bitova<br>Manje od 1 na $10^{16}$ za nove generacije (RE4) | Manje od 1 na $10^{15}$ bitova |
| **AFR %** | Nedefinirano | 0,73 | Nedefinirano |
| **TLER** | NE | DA | DA |
| **Zaštita od vibracija** | *Stable Trac, VCT* | *Stable Trac, VCT, RAAF* | *Stable Trac, Enhanced RAFF* |

Iz tablice s prethodne stranice je vidljivo sljedeće:

- **MTBF** ili engl. "*Mean time between failures*" je predviđeno vrijeme rada (u satima) diska prije kvarova ili grešaka u radu. Odnosno predviđen radni vijek sâmog diska.
- **Load unload Cycles** je broj pokretanja diska. Naime nakon svakog paljenja ili gašenja računala, disk se pokreće odnosno njegov *servo* motor "zavrti" ploče diska i vrti ih konstantnom brzinom. Ovdje se radi o tome koliko puta se disk može zaustaviti (ugasiti) i pokretati. Primjerice kada disk odlazi u stanje spavanja (Engl. *Sleep mode*) motor se zaustavlja i disk se gasi, a kod izlaska iz faze spavanja, disk se ponovno uključuje i "zavrti".
- **Broj nepopravljivih grešaka prema bitovima pročitanih podataka** je vrsta greške na koju će se naići s vremenom i koja se kao što i naziv kaže: neće moći popraviti. Naime svaki tvrdi disk s vremena na vrijeme nailazi na greške te ih popravlja u radu, a da mi toga nismo niti svjesni. Međutim također može doći i do pogrešaka koje elektronika diska ne može popraviti i to su upravo one greške koje su ovdje navedene. Vidljivo je da profesionalne serije diskova imaju 10 (deset) puta veću pouzdanost od "običnih" diskova (barem prema specifikacijama).
- **AFR** ili engl. "*Annualized failure rate*" daje nam vjerojatnost da će se disk pokvariti u toku cjelogodišnje stalne upotrebe. Vjerojatnost od 0,73 % za disk znači vjerojatnost od 0,73 % da se disk pokvari unutar jedne cijele godine rada (tj. 8.760 sati konstantnog rada).
- **TLER** ili engl. "*Time Limited Error Recovery*". Ovu opciju drugi proizvođači poput *Samsung*/*Hitachi*, nazivaju i ***CCTL*** (*Command Completion Time Limit*). Prvo se vratimo na osnove; kao što smo rekli: svaki tvrdi disk u pozadini stalno popravlja greške na koje povremeno nailazi. Kod običnih diskova to nije problem; ponekad možemo primijetiti da disk nešto radi, a da zapravo ne zapisuje ili ne čita neke nama vidljive podatke na disk. U tim trenucima naš disk vjerojatno u pozadini sâm popravlja određene greške, i to može potrajati neko vrijeme (i nekoliko minuta).
  U normalnom radu nam vrijeme koliko se greške popravljaju u pozadini nije previše važno, ali ako se disk koristi unutar nekog RAID kontrolera to postaje problem. Naime RAID kontroler očekuje da, ako neki disk ima greške, a koje sâm disk ne može popraviti u roku od par sekundi do maksimalno prosječno 8 sekundi, da disk prestane s procedurom popravljanja. Dakle ako disk ne popravi grešku za maksimalno 8 sekundi[*], on mora prestati s popravljanjem i prijaviti grešku RAID kontroleru koji će tada taj disk proglasiti neispravnim. Ako se ovo ne bi dogodilo cijelo RAID polje bi se zaustavilo s radom, sve dok se problematični disk ne popravi; ako se uopće uspije popraviti. Dakle ovo je vrlo važna funkcionalnost za diskove koje koristimo u RAID poljima.
- **Zaštita od vibracija** - standardni diskovi imaju standardne mehanizme zaštite od vibracija dok oni za profesionalnu primjenu moraju imati napredne mehanizme zaštite. **RAFF** tehnologija je jedna od njih (engl. *Rotary Acceleration Feed Forward*). *RAFF* tehnologija pomaže očuvanju performansi diska, ali i povećanju njegovog radnog vijeka. Naime ovdje je naglasak na stabilnost rada diska, koju mogu narušiti vibracije drugih (susjednih) diskova, ventilatora ili slično. Zamislimo poslužitelj koji ima više od dva diska od kojih svaki stvara svoje vibracije i prenosi ih na drugi disk, uz sve ventilatore unutar kućišta koji donose nove vibracije itd. Sve ove vanjske vibracije (vibracije izvan sâmog diska) su u neprekidnom radu 365 dana (ili više) vrlo problematične odnosno stvaraju probleme u radu diska.

***Sada postaje jasnije, zbog čega postoje diskovi za profesionalnu upotrebu.***

Kod **SSD** (engl. *Solid State Disk*) diskova, također postoji ista podjela prema standardnim i profesionalnim serijama diskova. Jedino su razlozi malo drugačiji. Prvi razlog je vrsta *flash* memorije, njena pouzdanosti i broja ciklusa zapisivanja te njene performanse. Drugi razlog je priručna brza RAM memorija (engl. *Cache*) na sâmom disku, uz koju se na profesionalnim SSD diskovima nalaze kondenzatori (učinak je poput *BBU* na RAID kontrolerima) za održavanje stanja memorije, do trenutka kada se svi podaci ne zapišu u *flash* memoriju diska.

## Stvarne statistike kvarova diskova

Pogledajmo statistike tvrtke *Google* na 100.000 diskova iz 2007 i tvrtke ***Backblaze*** na 100.000+ diskova. Svi navedeni diskovi su bili u upotrebi 24/7/365, odnosno *„non-stop"* minimalno cijelu godinu, bez gašenja. Dakle minimalno 8.760 sati konstantnog rada. Napomena:

- Statistike tvrtke *Google* koje spominjemo se odnose na razdoblje od 2005 do 2006 te se ovdje radi o starijim generacijama različitih diskova, koji su proizvedeni od 2001, nadalje radi se o "*Desktop*" i "*Server*" varijantama diskova (odnosno „*obični*" i profesionalni). Za dugoročnije statistike su korištene starije *Googleove* statistike.
- Statistike tvrtke *Backblaze* se odnose na razdoblje od 2013 do trećeg kvartala 2019.g te su u statistiku uključeni i nešto stariji, ali i najnoviji diskovi. Ovdje se u statistikama koriste također i "*Desktop*" i "*Server*" varijante diskova, a statistika je vidljiva prema točnom modelu diska.

**Ako sumiramo oba izvještaja, možemo zaključiti sljedeće**:

- Malo povišena radna temperatura diska ne utječe na njegovo kvarenje. Ne spominju se točne razlike, ali se pretpostavlja kako se radi o relativno malim povećanjima temperature. S druge strane snižavanje radne temperature dosta ispod preporučene, počinje povećavati broj pogrešaka.
- Prosjek kvarova prema mjesecima ili godinama upotrebe nam govori kako se ukupno pokvarilo (*Google* statistika do 2007.g.):
    - Oko 3% diskova već unutar prva tri mjeseca upotrebe.
    - Oko 2% diskova unutar prve godine upotrebe.
    - Oko 8% diskova unutar dvije godine upotrebe.
    - Oko 9% diskova unutar tri godine upotrebe.
    - Oko 6% diskova unutar četiri godine upotrebe.
    - Oko 7% diskova unutar pet godine upotrebe.
- Ukupno, prosjek kvarova diskova prema statistikama tvrtke *Google* za pet godina je 6.4%.
    - Statistika kvara diskova od (uključujući) 2013 do 2015.g., tvrtke *Backblaze* govori o prosječnom postotku kvarova diskova od 4.81% za sve tri godine statistike
    - Prosjek za više godina upotrebe je negdje oko 5,6 %. U svakom slučaju to je drastično više od specifikacije koje daju proizvođači kao "*AFR rate*" koji je u prosjeku između 0,5% i 0,8% godišnje. Ako pogledamo najpovoljniji postotak od 2% godišnje (to je AFR) to je odnos od 3 puta. Dakle tri puta (300%) je veća vjerojatnost da će se neki disk pokvariti u odnosu na ono što proizvođač izjavljuje u dokumentaciji.
- Nakon što se pojavi prva greška na disku tijekom skeniranja diska, vjerojatnost da će se disk pokvariti unutar 60 dana je **39 puta** veća nego na disku na kojemu se greške nisu pojavile. Dakle ako vam se pojavi i najmanja greška na disku, kupite novi i zamijenite stari disk i to hitno.
- Diskovi na kojima je počelo dolaziti do realokacije neispravnih sektora na nove pozicije (statistika vidljiva preko **S.M.A.R.T.** kao "*Reallocated Sector Count*") imaju **14 puta** veću vjerojatnost da će se pokvariti o odnosu na diskove koji nisu imali ovaj problem.
- Diskovi na kojima je počelo dolaziti do tzv.. "*Offline reallocation*" (ako imaju takvu statistiku) imaju **21 puta** veću vjerojatnost da će se pokvariti u odnosu na diskove koji nisu imali ove greške.
- Sveukupno, za sve diskove koji su imali bilo kakvu ***S.M.A.R.T. scan*** grešku, za njih je vjerojatnost da će se pokvariti unutar 60 dana veća **10 puta** od diskova koji nisu imali nikakve ***S.M.A.R.T scan*** greške.
- Diskovi koji su iz kategorije "Serverski" su se ipak pokazali nešto pouzdaniji od "običnih" diskova (ovisno o proizvođaču i točnom modelu).
- A sada i nešto što traži dublju analizu:
    - 56% diskova koji su se pokvarili nisu imali nikakve "*Strong*" S.M.A.R.T greške (očite, osnovne greške koje se prate preko **S.M.A.R.T.**).
    - U slučajevima kada su s pratile sve statistike iz **S.M.A.R.T**. odnosno kada su se redovito pokretali ***Full S.M.A.R.T*** testovi, 36% diskova koji nisu imali nikakve greške su se ipak pokva

**Izvori informacija:**

- Tvrtka ***Google*** (statistika na oko 100.000 diskova):
  http://static.googleusercontent.com/media/research.google.com/en//archive/disk_failures.pdf
- Tvrtka ***Backblaze*** (statistika na oko 50.000 diskova za 2015. godinu, do oko 211.000+ diskova za 2022. godinu):
  https://www.backblaze.com/blog/hard-drive-reliability-q3-2015/ → za 2015. godinu
  https://www.backblaze.com/blog/hard-drive-reliability-stats-q1-2016/ → za 2016. godinu
  https://www.backblaze.com/blog/hard-drive-stats-for-2017/ → za 2017. godinu
  https://www.backblaze.com/blog/hard-drive-stats-for-q1-2018/ → za 2018. godinu
  https://www.backblaze.com/blog/backblaze-hard-drive-stats-q1-2019/ → za 2019. godinu
  https://www.backblaze.com/blog/backblaze-hard-drive-stats-for-2020/ → za 2020. godinu.
  https://www.backblaze.com/blog/ssd-edition-2021-drive-stats-review/ → za SSD diskove u 2021. godini.
  https://www.backblaze.com/blog/backblaze-drive-stats-for-q1-2022/ → za mehaničke diskove u 2022. godini.

# 13.2. Geometrija diskova

S obzirom na činjenicu da operativni sustav, kao i sve programe i podatke držimo i spremamo na tvrdi disk, važno je razumjeti osnove rada tvrdog diska. Tvrde diskove nazivamo i mehanički odnosno rotacijski diskovi.
Svaki tvrdi disk sastoji se od nekoliko dijelova:

- Upravljačke elektronike.
- *Servo* motora (elektro motor koji radi na konstantnom broju okretaja).
- Ploča na koje se zapisuju podaci, a koje pogoni *servo* motor.
- Glava za čitanje i pisanje te kućišta diska.

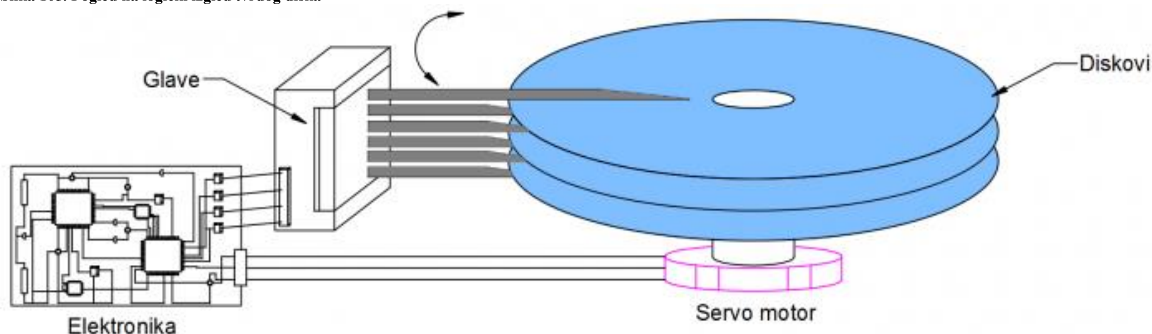
Fizički izgled tvrdog diska je vidljiv na slici **104**.

**Slika 104. Pogled na fizički tvrdi disk (otvoren, bez kućišta).**



Pojednostavljeno, logički, tvrdi disk izgleda kao na slici **105**.

**Slika 105. Pogled na logički izgled tvrdog diska**



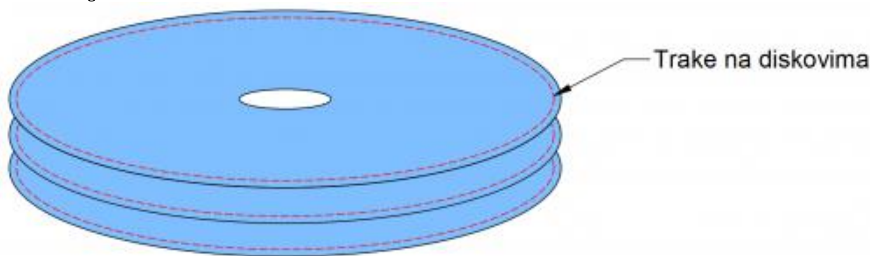

Autor slike: Hrvoje Varga

Tvrdi disk radi tako da upravljačka elektronika kontrolira rad *servo* motora, koji konstantnom brzinom rotira ploče (diskove) na koje se zapisuju podaci. Podatke na disk zapisuju i čitaju glave (engl. *head*) koje se nalaze sa svake strane ploče (diska).
Glave se pomiču odnosno pozicioniraju prema potrebi, na lokaciju (***traku/stazu***) s koje moraju čitati ili pisati podatke.
Podaci se čitaju i zapisuju s obje strane svake ploče (diska). Tablica prikazuje rotacijske brzine diskova, danas dostupnih na tržištu:

| Broj okretaja diska [o/min] |
|---|
| 5.400 |
| 7.200 |
| 10.000 |
| 15.000 |

Što je veća brzina rotacije diska i sve ostale komponente diska moraju raditi brže, a sâmim time brzina pristupa podacima kao i brzina prijenosa podataka postaje veća. Površina svake ploče (diska) je podijeljena u koncentrične krugove tj. u takozvane trake odnosno staze, kako je vidljivo na slici **106**.

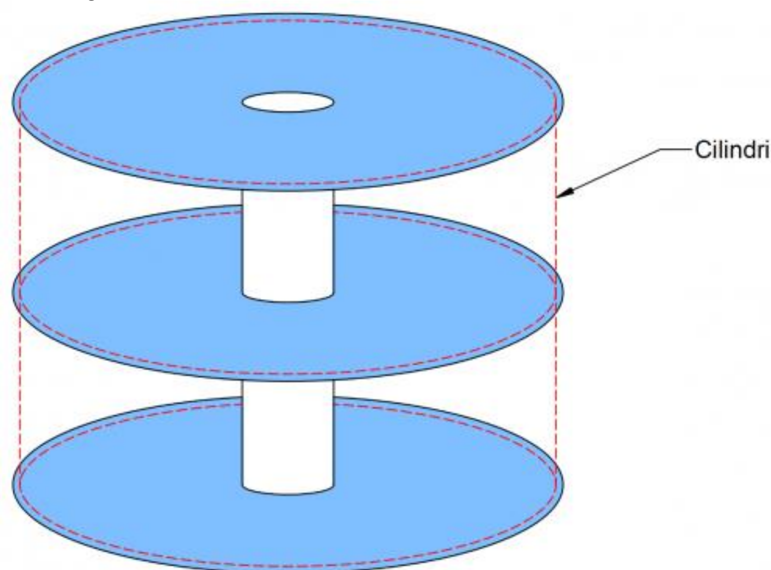
**Slika 106. Pogled na trake odnosno staze na disku.**




Autor slike: Hrvoje Varga

Više ***traka*** (*staza*) na istoj *vertikalnoj* poziciji, na svim pločama tj. diskovima se naziva cilindrom (slika **107** dolje)

**Slika 107. Pogled na cilindre na disku.**




Autor slike: Hrvoje Varga

**Kako BIOS prepoznaje tvrdi disk i njegov kapacitet**

Spajanjem tvrdog diska na disk kontroler (ATA/SATA/SCSI/SAS/…) na matičnoj ploči ili na PCI/PCI Express kartici, disk kontroler prijavljuje tvrdi disk *BIOS*-u računala. Važno je znati da svaki tvrdi disk dolazi s tvorničkim formatom na vrlo niskoj razini (engl. *Low Level Format*) koji služi za pravilno pozicioniranje glava na pozicije za pisanje/čitanje, prvo na pravu stazu, a potom za svaki sektor na disku.

Nakon što je tvrdi disk inicijalno vidljiv u *BIOSu* računala, prvi korak *BIOS-a* je preslikavanje fizičke i logičke podjele na tvrdom disku, sve do razine sektora, prema kojem se izračunava i nazivni kapacitet diska.

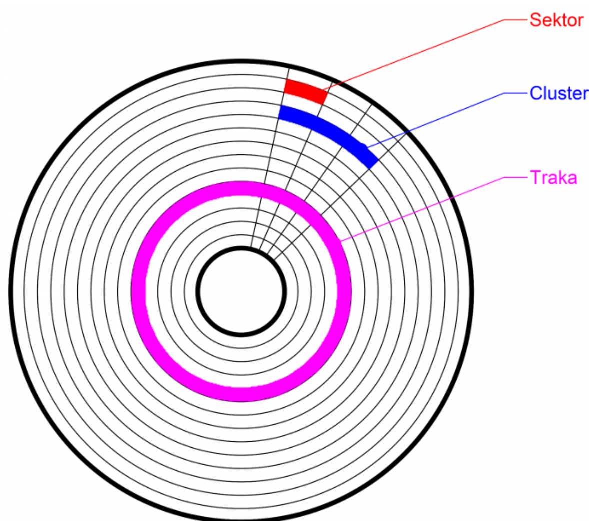
- Starije metode su koristile **CHS** (Engl. *Cilinder Head Sector*) preslikavanje (mapiranje) odnosno adresiranje tj. identifikaciju svakog individualnog sektora na disku prema njegovoj poziciji unutar trake (staze). Pri tome je traka (staza) bila određena prema (logičkoj) glavi (engl. *Head*) i broju cilindra. **CHS** zbog svog načina preslikavanja i ograničenosti koje donosi, može adresirati tvrde diskove do veličine od 504 MB.
- Novije metode koriste takozvanu **LBA** (Engl. *Logical block addressing*) metodu koja logički adresira sektore, odnosno preslikava *CHS* adrese u *LBA* adrese. Ranije inačice su imale ograničenje do maksimalne veličine tvrdog diska od 2,1GB a novije do 8,4 GB. Standard se proširivao pa je sljedeća granica bila 137 GB i dalje sve do 2TB granice. Najnovija **LBA** shema adresiranja je 48 bitna, pa podržava adresiranje do 128 *PB (Peta bajta)*.

Vezano za particioniranje, najnovije implementacije koje više ne koriste ***BIOS*** već ***UEFI*** (Engl. *Unified Extensible Firmware Interface*) kao noviju inačicu *BIOS*a, te ***GPT*** shemu particioniranja, koriste 64 bitno adresiranje pa je prema tome ograničenje, ako imamo sektor od 512 bajta pomnoženo sa 64 bîta = 9,4 ZB ($9{,}4 \times 10^{21}$ *bajta*), dakle ***9,4 Zeta bajta***.

***Svi moderni operativni sustavi, od trenutka pokretanja operativnog sustava, više se ne oslanjaju na BIOS tijekom pristupanja disku (čitanja ili zapisivanja). Dakle oni koriste direktnu metodu pristupa diskovima, bez posredovanja BIOSa.***

Pogledajmo logičku površinu jedne ploče diska, na slici **108.**

**Slika 108. Pogled na sektore, clustere i trake na disku.**




Područja za zapisivanje podataka se nalaze unutar ***traka***. Svaka ***traka (staza)*** je podijeljena na najmanje jedinice koje se nazivaju ***sektori***. Veličina sektora na većini tvrdih diskova je obično 512 bajta, dok je na najnovijim diskovima ova veličina 4096 bajta odnosno 4 kilo bajta (*kB*).
Podaci se u konačnici zapisuju u blokove od nekoliko *sektora* koji se nazivaju ***klasteri***. Veličina *klastera* podataka, koji je i najmanja jedinica za zapisivanje podataka, ovisi o datotečnom sustavu (engl. *File System*). Proces logičkog formiranja površine diska u *trake* i *sektore* se zove formatiranje. Različiti datotečni sustavi imaju različite moguće veličine *klastera* koje možemo odabrati i to samo u trenutku formatiranja particije.
Kao što smo rekli, podaci se zapisuju u *klastere* kao najmanje jedinice zapisa, jedan za drugim (koliko god je to moguće). Tijekom zapisivanja podataka na disk cilj je nîzove podataka koje zapisujemo, zapisati u *klastere* koji su u nîzu. U tom slučaju je i čitanje brže nego da su podaci razbacani na različitim dijelovima diska, što se može dogoditi kod brisanja nekih podataka koji se nalaze pozicionirani između drugih podataka odnosno datoteka, a nalaze se i na drugim stazama.
Naime kada se nakon brisanja ponovno zapisuju neki novi podaci, može se dogoditi da su upisani podaci dijelom zapisani na mjesta gdje su bili prethodno obrisani podaci, a drugim dijelom, ako su novi podaci veći od onih koji su prethodno obrisani, na prvu sljedeću slobodnu poziciju. Ovaj problem se zove *fragmentacija* i redovito se događa na nekim datotečnim sustavima koji nemaju zaštitni mehanizam protiv ***fragmentacije***.



*Datotečni sustavi: **FAT** i **NTFS** imaju problem s fragmentacijom jer nemaju zaštitni mehanizam protiv nje.*
*U slučaju fragmentacije podataka, čitanje s diska postaje s vremenom sve sporije i sporije.*



Vezano za shemu diskova i particioniranje, pogledajte sljedeća poglavlja:
**13.4. Logička shema diska.**
**13.6. Rad s particijama.**

**Vratimo se na klastere.**
Veličinom klastera je uvjetovana i veličina cijelog datotečnog sustava, a ovaj odnos je specifičan za svaki datotečni sustav.
Tablice pokazuju odnos maksimalne veličine particije uz pripadajuću veličinu klastera, za pojedini datotečni sustav ***(FAT16/FAT32/NTFS ili ext3 i ext4):***

**FAT16**

| Maksimalna veličina particije | Veličina klastera |
|---|---|
| 64 MB | 1 KB |
| 128 MB | 2 KB |
| 256 MB | 4 KB |
| 512 MB | 8 KB |
| 1 GB | 16 KB |
| 2 GB | 32 KB |
| 4 GB | 64 KB |

**FAT32**

| Maksimalna veličina particije | Veličina klastera (*Block size*) |
|---|---|
| 128 MB | 1 KB |
| 256 MB | 2 KB |
| 8 GB | 4 KB |
| 16 GB | 8 KB |
| 32 GB | 16 KB |

**NTFS**

| Maksimalna veličina particije | Veličina klastera |
|---|---|
| 16 TB | 4 KB |
| 32 TB | 8 KB |
| 64 TB | 16 KB |
| 128 TB | 32 KB |
| 256 TB | 64 KB |

Linux **ext3/ext4**

| Maksimalna veličina particije | Veličina klastera (*Block size*) |
|---|---|
| 4 TB | 1 KB |
| 8 TB | 2 KB |
| 16 TB | 4 KB |
| 32 TB | 8 KB |



Linux/UNIX datotečni sustavi veličinu klastera nazivaju veličinom bloka odnosno "***Block size***", sâmo u kontekstu datotečnog sustava. Jedan blok je i najmanja moguća jedinica za zapisivanje podataka na disk!.

**Primjer**: Ako nam je veličina sektora 512 bajta i koristimo datotečni sustav kojem je veličina klastera 4KB to znači kako se koristi 8 sektora za jedan klaster. Pri tome, ako trebamo zapisati datoteku veličine 4KB to znači da će se podaci zapisati točno u jedan klaster. Međutim, ako trebamo zapisati datoteku veličine 7KB to znači kako će se iskoristiti dva klastera od 4KB (ukupno 8KB), a ostatak od 7KB do 8KB u tom klasteru će ostati neiskorišten jer se podaci mogu zapisivati samo do razine klastera, bez obzira koliko će biti popunjen ili ne.

**Pogledajmo i tablicu s nekim od ograničenja često korištenih datotečnih sustava:**

| Datotečni sustav | Maksimalna veličina datoteke | Maksimalna veličina datotečnog sustava (*particije*) | Maksimalna duljina imena datoteke | Maksimalni broj datoteka |
|---|---|---|---|---|
| `exFAT` | **128 PB** | **128 PB** | **255** | **2.796.202** *[po direktoriju]* |
| `ext4` | **16 TB** | **1 EB** | **255** | **4.000.000.000** [definirano u trenutku formatiranja] |
| `FAT32` | **4 GB** | **2 TB** | **255** | **65.460** |
| `NTFS` | **8 PB** | **8 PB** | **255** | **4.294.967.295** |
| `ZFS` | **16 EB** | **$256x2^{50}$ [$2^{128}$ *bajta*]** | **255** | **$2^{48}$***[po direktoriju]* *Ukupno – neograničeno* |
| `XFS` | **8 EB** | **8 EB** | **255** | **$2^{64}$** |

**Izvori informacija: (313),(1000),(1001),(1002),(1003),(1004),(1005),** `man 5 filesystems`.

## 13.2.1. Drugi parametri rada mehaničkih diskova

Pogledajmo neke od drugih važnijih parametara rada mehaničkih diskova.

### 13.2.1.1. Vrijeme pristupa (Access time)

Na vrijeme pristupa disku, odnosno brzinu prijenosa podataka od i prema diska, utječe nekoliko parametara, od kojih ćemo spomenuti samo:

- Vrijeme traženja i pozicioniranja glava za čitanje i pisanje ***(Engl. Seek time)***.
- Rotacijska latencija odnosno kašnjenje zbog rotacije ploča diskova (***Engl. Rotational latency***).
- Brzina prijenosa podataka **(*Engl. Data transfer rate*)**.
- Utjecaj datotečnog sustava (***Engl. Effect of file system***).
- Broj istovremenih ulazno izlaznih operacija prema i od diska (***Engl. Input/output operations per second*** *(IOPS)*).

Vrijeme pristupa ili odziva mehaničkih diskova je vrijeme koje protekne prije nego možemo pristupiti podacima pohranjenim na pločama diska.
Ovdje se većinom radi o mehaničkim faktorima, poput rotacije diskova ili pomicanja glava za čitanje i pisanje kako bi se došlo do pozicije na disku na kojoj se nalaze traženi podaci.
Upoznajmo se s činjenicama o čemu se tu sve radi.

### 13.2.1.2. Vrijeme traženja (*Seek time*)

Vrijeme traženja (***Engl. Seek time***) se odnosi na vrijeme traženja kod rotirajućih ploča diska, koje je potrebno mehanici diska koja pomiče glave za čitanje/pisanje kako bi došle na pravu stazu (traku) odnosno poziciju na kojoj obaviti željeno čitanje ili zapisivanje. Prosječno vrijeme traženja odnosno pozicioniranja glave u odnosu na rotirajuće ploče, varira od 4ms za *enterprise* odnosno poslužiteljske diskove, pa čak do 12-15 ms za diskove za prijenosna računala.

Diskovi za stolna računala, odnosno "standardni" diskovi imaju vremena pristupa negdje između 8ms i 10ms.

### 13.2.1.3. Rotacijsko kašnjenje

Rotacijsko kašnjenje odnosno latencija odnose se na vrijeme koje je potrebno kada se glava već pozicionirala na pravu traku odnosno stazu, da se ploča diska koja se stalno rotira, zarotira na točnu poziciju do traženog sektora unutar trake/staze.
Dakle što se diskovi rotiraju većom brzinom prije će se unutar staze odnosno trake u kojoj se nalazi traženi sektor, doći do njega.

Tipična vremena čekanja odnosno kašnjenje koje je uzrokovano rotacijskim brzinama ploča diskova, vidljiva su u tablici:

| **Brzina rotacije diskova [*rpm* tj. o/min]** | **Prosječno kašnjenje [ms]** |
|---|---|
| 4.200 | 7,14 |
| 5.400 | 5,56 |
| 7.200 | 4,17 |
| 10.000 | 3 |
| 15.000 | 2 |

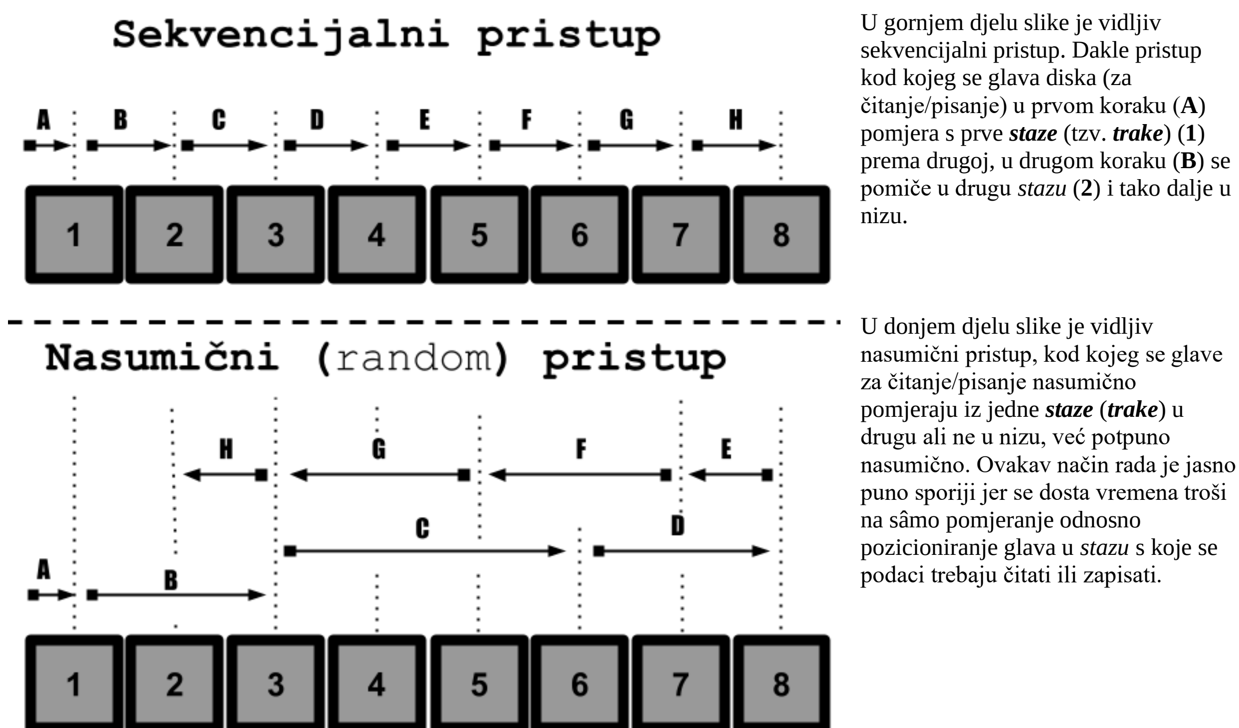
### 13.2.1.4. Brzina prijenosa podataka (*Data transfer rate*)

Brzina prijenosa podataka, koja se naziva i propusnost pokriva mnoge utjecaje, od internih poput pozicioniranja glava (engl. *Seek time*) te čekanja zbog sâme rotacije diska (*rotacijsko kašnjenje*), do prijenosa podataka prema elektronici diska i njenom memorijskom međuspremniku, do disk kontrolera, pa sve do zaprimanja podataka od strane kernela operativnog sustava. Efektivne brzine prijenosa podataka, ovisno o modelu i namjeni mehaničkog diska variraju između 50MB/s do 150MB/s ili nešto malo više. Brzine snimanja i čitanja također variraju. I na kraju sve ovisi kako se pristupa podacima:

- Sekvencijalno odnosno u nîzu, jedan za drugim.
- Nasumično (engl. *Random*) odnosno izvan nîza.

Pogledajmo kako logički izgledaju sekvencijalni i nasumični pristup podacima (slika 109.1).

**Slika 109.1. Sekvencijalni i nasumični pristup (disku).**




U gornjem djelu slike je vidljiv sekvencijalni pristup. Dakle pristup kod kojeg se glava diska (za čitanje/pisanje) u prvom koraku (**A**) pomjera s prve ***staze*** (tzv. ***trake***) (**1**) prema drugoj, u drugom koraku (**B**) se pomiče u drugu *stazu* (**2**) i tako dalje u nizu.

U donjem djelu slike je vidljiv nasumični pristup, kod kojeg se glave za čitanje/pisanje nasumično pomjeraju iz jedne ***staze*** (***trake***) u drugu ali ne u nizu, već potpuno nasumično. Ovakav način rada je jasno puno sporiji jer se dosta vremena troši na sâmo pomjeranje odnosno pozicioniranje glava u *stazu* s koje se podaci trebaju čitati ili zapisati.



Za optimizaciju pristupa podacima, pogledajte poglavlje: **13.2.1.7. Native Command Queuing (NCQ).**

### 13.2.1.5. Efekt datotečnog sustava

I na kraju, i sâm datotečni sustav može utjecati na konačnu brzinu prijenosa podataka, od i prema disku. Pogotovo kod datotečnih sustava kod kojih dolazi do porasta *fragmentacije* podataka. Ovakva *fragmentacija* može znatno utjecati na konačnu brzinu prijenosa podataka. Rješenje ovog problema, kod datotečnih sustava koji su podložni fragmentaciji je pokretanje defragmentacije periodički. Nisu svi datotečni sustavi podložni porastu fragmentacije već ju mogu držati unutar granica od samo par postotaka. Naime datotečni sustavi poput: ***ext4***, ***XFS***, ***Reiser4*** i ***ZFS*** imaju mehanizam za prevenciju fragmentacije.

Nadalje, osim *fragmentacije*, svaki datotečni sustav na malo drugačiji način zapisuje i pristupa datotekama i direktorijima. Osim toga datotečni sustavi osim samih datoteke, uz njih zapisuju i njihove ovlasti i atribute, a neki od njih i **ACL** liste. Primjerice upotreba ACL-a dodatno može usporiti diskovne operacije.



Za detalje oko ***ACL***-a i potencijalnog usporavanja sustava zbog njih, pogledate napomenu na kraju poglavlja:
**4.4.4. Niti to nije sve - Access control list (ACL).**

**Izvori informacija: (1597),(1659)**

### 13.2.1.6. IOPS (*Input/output operations per second*)

Ulazno izlazne operacije prema disku, koje se nazivaju i **IOPS** (Engl. *Input/output operations per second*) označavaju mjeru performansi diska, bilo mehaničkog ili SSD. Naime ovdje se radi o sposobnosti diska da odradi određeni broj operacija pristupanja podacima, bilo da se radi o operacijama čitanja ili zapisivanja, u jednoj sekundi.
Zamislimo dvije situacije u kojima se ukupno treba pročitati 10GB podataka:

1. Prva u kojoj je potrebno pročitati 10 datoteka, svaku veličine 1GB.
2. Druga u kojoj je potrebno pročitati 10.000 datoteka veličine 1MB.

**Dogodit će se sljedeće:**
**1.** Slučaj - učitat će se 10 datoteka vrlo brzo, recimo brzinom od 100 MB/s, pa će biti potrebno nekih 100 sekundi da se sve datoteke pročitaju. Pri tome imamo 10 IOPS operacija čitanja u sekundi.

**2.** Slučaj - učitavat će se 10.000 datoteka, što naravno naš disk neće moći podnijeti, jer čak i da imamo SAS disk koji radi na 15.000 okretaja, on nije u stanju odraditi više od 210 operacija čitanja u sekundi.
Zapravo će se odraditi maksimalno 210 operacija čitanja u sekundi, ali vrlo usporeno jer će se kod tog vršnog opterećenja glave diska morati stalno pomicati ne bi li očitavale datoteke, a koje su vjerojatno raspršene na drugim (različitim) stazama na disku. U konačnici će biti potrebno znatno više vremena da se pročita svih 10.000 datoteka, a brzina prijenosa podataka će postati mizerna: možda samo nekoliko MB/s.

Za mehaničke diskove u pravilu vrijede sljedeći brojevi ulazno izlaznih operacija (**IOPS**), ovisno o rotacijskoj brzini diskova:

| Vrsta diska/sučelje | Broj ulazno/izlaznih operacija (IOPS) |
|---|---|
| 5.400 o/min SATA | ~15 - 50 IOPS |
| 7.200 o/min SATA | ~75 - 100 IOPS |
| 10.000 o/min SATA | ~125 - 150 IOPS |
| 10.000 o/min SAS | ~140 IOPS |
| 15.000 o/min SAS | ~175 - 210 IOPS |



**SSD** i **NVMe** diskovi imaju drastično veći broj **IOPS** operacija jer nemaju pomičnih dijelova odnosno pomjeranja glava za čitanje/zapisivanje !



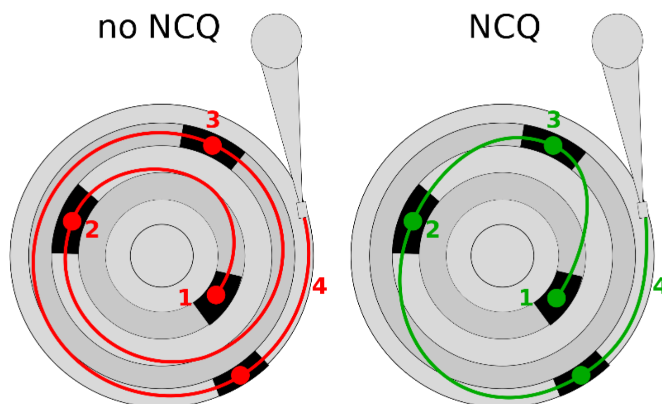
Za detalje oko **SSD** i **NVMe** diskova pogledajte poglavlja:
**13.3. SSD diskovi.**
**13.3.2. NVMe diskovi.**

**Izvori informacija: (1006).**

## 13.2.1.7. Native Command Queuing (NCQ)

*Native Command Queuing* (**NCQ**) je proširenje serijskog ATA (**SATA**) standarda koji omogućuje tvrdim diskovima internu optimizaciju redoslijeda izvršavanja primljenih naredbi za čitanje i zapisivanje. Optimizacija se ovdje svodi na odabir dohvaćanja podataka prvo s onih *staza* na disku, do kojih se dolazi s minimalnim pomjeranjem glave za čitanje/zapisivanje. Stoga ovakav način rada može smanjiti količinu nepotrebnog kretanja glave pogona diska, što rezultira povećanim performansama za radna opterećenja u kojima postoje višestruki istovremeni zahtjevi za čitanje ili zapisivanje na disk.

**Slika 109.2. Način čitanja/zapisivanja: s NCQ ili bez NCQ**




**Autor fotografije: https://en.wikipedia.org (helix84)**

**NCQ** načinu rada prethodila je paralelna ATA inačica zvana: *Tagged Command Queuing* (**TCQ**). Međutim pokušaj integracije **TCQ** u **(P)ATA** diskove je bio ograničen činjenicom da su **ATA** uređaji (diskovi) na sabirnici koristili **ISA** protokole za interakciju s operativnim sustavom. To je rezultiralo visokim opterećenjem CPU-a, a zanemarivim dobitkom na performansama. Stoga **TCQ** nikada nije znatnije prihvaćen na tržištu. **NCQ** se razlikuje od **TCQ**-a i po tome što je kod **NCQ**-a, svaka naredba za čitanje ili zapisivanje od jednake važnosti, a još važnije, **NCQ** preko sabirnice koristi **DMA** metodu rada, bez posredovanja CPU-a, dok **TCQ** prekida CPU (**IRQ**) tijekom upita s naredbama i tek tada koristi **DMA** pristup.

Da bi **NCQ** radio, on mora biti podržan i omogućen od strane **SATA** disk kontrolera, ali i na samom tvrdom disku. Dodatno je potreban i odgovarajući upravljački program za **SATA** disk kontroler. Osim toga potrebna je i podrška za **AHCI** sučelje preko kojeg se to sve odrađuje, odnosno preko kojeg se omogućuje operativnom sustavu upravljanje **SATA** diskovnim podsustavom i upotreba **NCQ-a**. Prva generacija tvrdih diskova s podrškom za **NCQ** se pojavila na tržištu 2004. godine.
Jedna manje poznata značajka **NCQ**-a je da on omogućuje operativnom sustavu da pri snimanju odredi želi li biti obaviješten kada podaci dođu do ploča diska ili kada dođu do međuspremnika diska (predmemorije na samom diskovnom pogonu [*disku*]). Ova značajka omogućuje jamčenje konzistentnosti podataka na disku, pogotovo kada se pristupa disku sa sistemskim pozivima kao što je ***fsync*** (primjerice izvršavanjem naredbe `sync`), ili u primjenama kod uporabe virtualizacije, pri odabiru metode pristupanja diskovnom podsustavu fizičkog računala, poput: **`Writeback`**, **`Directsync`** ili **`Writethrough`**.



Pogledajte cjelinu ***Disk kontroleri*** u poglavlju:
**27.1.2.3. Disk kontroleri.**

**NCQ** se također koristi i u novim **SSD** pogonima gdje ne postoje mehaničke komponente diska, ali ipak dolazi do određene latencije. Na primjer **SSD** pogoni mogu koristiti **NCQ** kako bi osigurali da pogon ima naredbe za obradu (za čitanje ili zapisivanje) dok je centralni procesor (CPU) zauzet obradom drugih zadataka odnosno programa (procesa).
**NCQ** također omogućuje **SSD** kontroleru istodobno prihvaćanje više zahtjeva za čitanjem ili zapisivanjem, ako unutarnja organizacija uređaja omogućuje takvu obradu. Standard *NVM Express* (**NVMe**) također podržava spremanje naredbi u redu čekanja upotrebom **NCQ**-a, ali u obliku optimiziranom za **SSD** pogone. Dodatno **NVMe** omogućuje više redova čekanja za pojedini diskovni kontroler i uređaj (pogon), omogućujući istovremeno mnogo veće dubine za svaki red čekanja, što više odgovara načinu na koji funkcionira **SSD** (i **NVMe**) hardver.
Međutim **NCQ** može ometati ***I/O scheduler*** operativnog sustava, zapravo smanjujući performanse sustava. To je uočeno u praksi na Linuxu, s upotrebom RAID-5 polja diskova. Naime unutar **NCQ**-a ne postoji mehanizam za određivanje bilo kakvih ograničenja za **I/O** sustav, kao primjerice koliko se puta zahtjev može zanemariti u korist drugih (pr. prioritetnijih) zahtjeva.
U teoriji, zahtjev u redu čekanja može se odgoditi za diskovni pogon, proizvoljno vrijeme, sve dok on poslužuje druge (možda i novije) zahtjeve. Štoviše, budući da algoritmi koji se koriste unutar *firmware*-a diskovnog pogona za rad **NCQ**-a nisu javno dostupni i poznati, to uvodi još jednu razinu nesigurnosti za performanse hardvera.
U svakom slučaju, i mi možemo napraviti određene promjene u Linuxu, po pitanju rada **NCQ**-a.
**NCQ** je standardno podržan u svim Linux kernelima, a ono što mi možemo učiniti je sljedeće:

**Isključiti ga, ako imamo dobar razlog (pr. testovima smo dokazali da nije dobar u kombinaciji s našim RAID poljem):**

```
echo 1 > /sys/block/sda/device/queue_depth
```

Za trajno permanentno i potpuno isključivanje **NCQ**-a (za cijeli Linux), možemo dodati sljedeću opciju u ***GRUB2 Boot loader***.

```
libata.force=noncq
```

Odnosno, ponovno ga možemo uključiti (što je standardno postavljeno) sa [***31*** *je preporučena vrijednost → pogledajte* (**1078**)]:

```
echo 31 > /sys/block/sda/device/queue_depth
```

**Izvori informacija: (1077),(1078), `man 5 sysfs` te sljedeća poglavlja u knjizi: 10.5.1, 10.6 i 10.8.**

## 13.3. SSD diskovi

*SSD* diskovi (Engl. *Solid state drive*) su uređaji za pohranu podataka koji se sve više koriste umjesto klasičnih mehaničkih diskova. Kod navedenih SSD *diskova,* ne postoje ploče odnosno diskovi koji se rotiraju, već se podaci spremaju na integrirane krugove (*čipove*). Sučelja na koja se može spojiti SSD disk su uglavnom **SATA** ili **SAS**, poput klasičnih mehaničkih diskova.
Standardno sučelje za spajanje SSD diskova je *SATA, preko* ***AHCI***, koje nosi neka ograničenja zbog svog naslijeđa, poput ograničenja na sâmo jedan nîz za naredbe (*command queue*) koji može prihvatiti sâmo 32 naredbe te sâmo jedan signal prekida. U novije vrijeme neke izvedbe SSD diskova spajaju se na **PCI Express** (*PCIe*) sabirnicu (**NVMe** standard) kao i na **M.2** utor. **M.2** utor je također direktno spojen na *PCIe* sabirnicu (iako postoje i varijante u kojima se koristi ***SATA***), stoga ima znatno veću propusnost i manje kašnjenje (latenciju) te nema posebne (odvojene) konektore za podatkovnu vezu i za napajanje. Dodatno, upotrebom ***NVMe*** se broj redova za naredbe povećao na 65.535, s time da se u svakom od njih može poslati do 65.535 naredbi.

Dizajn snažnijih ***NVMe*** diskova je obično takav da na ulazu imaju ekstremno brze RAM te *SLC flash* međumemorije. Dakle oni praktično prvo kao međumemoriju (*Buffer*) imaju ekstremno brzu RAM memoriju, a potom kao međumemoriju druge razine *SLC flash* memoriju, koja je višestruko brža od **MLC/TLC/QLC**, na koje se u konačnici podaci i zapisuju. Dodatno sve ovisi i o njihovom kontroleru i njegovim algoritmima upotrebe navedenih međumemorija, kao i njihovim veličinama. U ovakvom dizajnu, ovisno o veličinama RAM i SLC međumemorija, koje su najvažnije tijekom zapisivanja, ovisi i kolika će u konačnici biti brzina zapisivanja, u odnosu na količinu podataka koje zapisujemo.
Naime, ako zapisujemo više podataka nego što ih stane u navedene međumemorije, brzina zapisivanja će drastično pasti.

Vratimo se na klasične SSD diskove odnosno uređaje.
Glavni dijelovi *SSDa* su memorijski kontroler i *Flash* memorija te o njima najviše ovise performanse uređaja. Memorija je u *NOR* ili *NAND* izvedbi. U praksi se najviše koristi *NAND* vrsta. Četiri su najčešće izvedbe *NAND-a*: **SLC** (eng. *Single-level cell*), **MLC** (eng. *Multi-level cell*), **TLC** (eng. *Triple-level cell*) i **QLC**.

U usporedbi s klasičnim mehaničkim diskom, *SSD* je mnogo otporniji na fizičke šokove, manjih dimenzija je i mase, tiši, energetski učinkovitiji, ima kraće vrijeme pristupa memoriji i manju latenciju što ga čini bržim. Međutim, negativne strane *SSDa* su visoka cijena (GB/kuna) i kapacitetom manji memorijski prostor. Svaka od gore navedenih izvedbi ima svoje prednosti i mane:

- **SLC** – ova vrsta *flash* memorije pohranjuje podatke u zasebnim memorijskim ćelijama (*memory cell*) od kojih svaka može imati samo dva stanja 0 i 1. Dakle u svaku ćeliju se može pohraniti samo jedan bît podataka. Ova vrsta memorije ima iznimno veliku brzinu zapisivanja i čitanja, malu potrošnju energije te najduži vijek trajanja, odnosno najveći broj pisanja i brisanja. Zbog toga što svaka ćelija može pohraniti samo jedan bît podataka, ona je i najskuplja. Zbog visoke pouzdanosti i performansi, kao i cijene, koristi se samo za "*enterprise*" sustave ili kao međumemorija.
- **MLC** - ova vrsta memorije u jednu memorijsku ćeliju može pohraniti dva bîta podataka, pa je stoga i jeftinija, te većeg kapaciteta. Njenu nešto manju nepouzdanost, danas rješava memorijski kontroler, pomoću *ECC* mehanizama. Drugi njen problem je nešto veća sporost i manji vijek trajanja odnosno znatno manji broj zapisivanja u odnosu na *SLC* vrstu memorije.
- **TLC** - ova vrsta memorije u stanju je u jednoj ćeliji pohraniti tri bîta podataka, uz još manju brzinu, i još manji vijek trajanja, odnosno broj zapisivanja. Njena prednost je još veći kapacitet i znatno manja cijena.
- **QLC** - ova vrsta memorije u stanju je u jednoj ćeliji pohraniti četiri bîta podataka, što je povećanje kapaciteta za 33% u odnosu na *TLC*, ali uz još nešto manju brzinu i još kraći vijek trajanja, odnosno broj zapisivanja. Njena prednost je još veći kapacitet i još manja cijena.

Osim ovih karakteristika, postoji još jedna gruba podjela, a to su SSD uređaji za:
*Kućnu* ili *uredsku* upotrebu s jedne strane te upotrebu za poslužitelje odnosno takozvanu upotrebu: 365/7 („*non-stop*").

Naime svaki *SSD* uređaj (*drive*) između svog memorijskog kontrolera i *flash* memorije ima i malu količinu brze *RAM* memorije (*DRAM na slici*), koja se koristi kao brzi memorijski međuspremnik. Ovakav je dizajn kod svih *SSD* uređaja. Međutim u slučaju nestanka električne energije svi podaci koji su ostali u toj brzoj RAM memoriji, a nisu spremljeni u *flash* memoriju, ostaju izgubljeni.
Ovo nije neki poseban problem za kućnu ili uredsku upotrebu, ali za poslužitelje definitivno jeste.
Iako i oni mogu dijelom biti imuni sa zaštitnim mehanizmima, primjerice na razini datotečnog sustava.

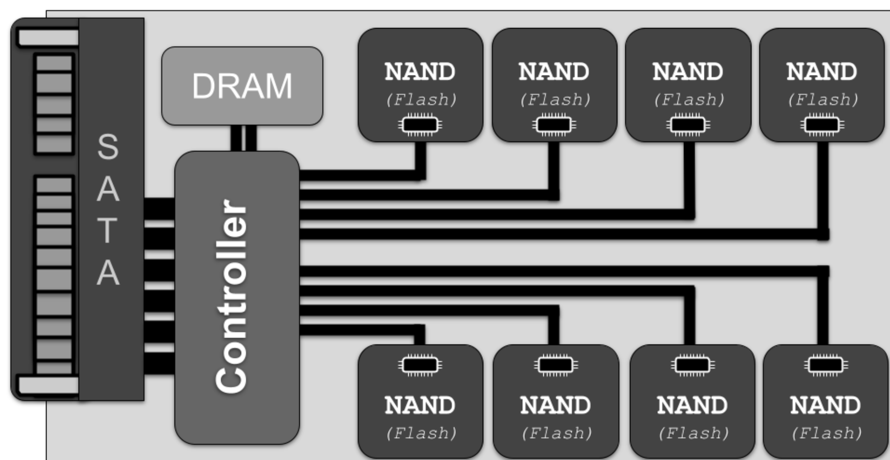
Ipak za poslužitelje postoji posebna vrsta *SSD* diskova koja imaju poseban kondenzator velikog kapaciteta uz ovu *DRAM* memoriju, a koji u slučaju nestanka električne energije čuva stanje ove RAM memorije, sve dok se računalo ponovno ne uključi/pokrene. Tada se svi podaci iz *DRAM* memorije sigurno zapisuju na flash memoriju (*NAND čip*).

Predstavnici ovakvih poslužiteljskih SSD diskova su primjerice:

- *Intel DC S3610 ili Intel DC S3500.*
- *Toshiba PX02SS* i drugi.

Pogledajmo i logičku shemu SSD diska, na slici 110.

Slika 110. Pogled na SSD disk: dizajn SSD diska.

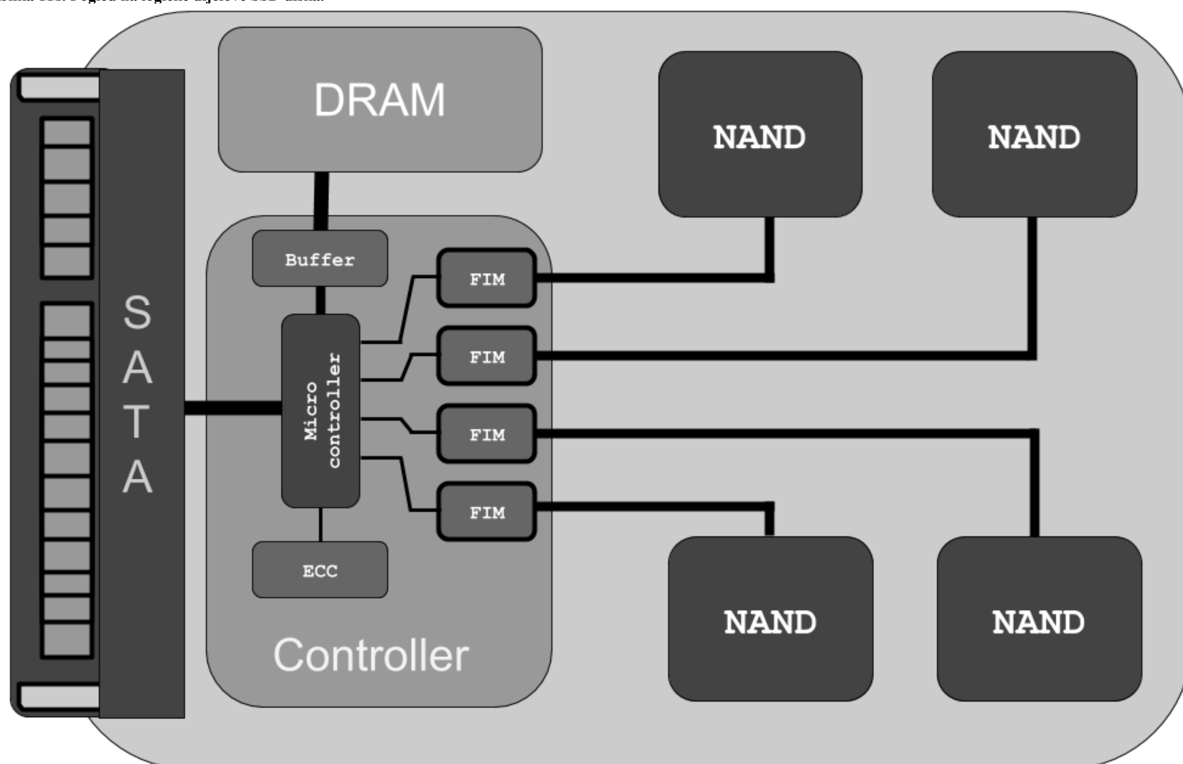



## 13.3.1. SSD detaljnije

***Slijedi napredno poglavlje (13.3.1.x)!.***

Pogledajmo malo detaljniju logičku shemu SSD diska. Na slici 111 su vidljive sljedeće logičke cjeline odnosno dijelovi.

Slika 111. Pogled na logičke dijelove SSD diska.




Na slici odnosno u dizajnu SSD diska vidimo sljedeće logičke cjeline:

- ***Micro controler*** na slici je mikroprocesor unutar *kontrolera* koji je zadužen za dohvaćanje (čitanje) i zapisivanja podataka na sâmu *NAND flash* memoriju, a koji u radu koristi pripadajuću mu vrlo brzu međumemoriju (*buffer*), kao i znatno veću i malo sporiju *DRAM* memoriju, a sve unutar sâmog SSD diska.
- ***Flash interface modules*** (*FIMs*) povezuje *mikrokontroler* unutar sâmog kontrolera s *NAND flash* memorijom. Dodatnim povećanjem broja *FIM* elemenata, pošto svaki od njih može istovremeno (paralelno) komunicirati s pripadajućom mu *NAND flash* memorijom, povećava se i ukupna brzina komunikacije s *NAND flash* memorijom.
- ***ECC*** (*Error Correction Code*) komponenta unutar *kontrolera*, zadužena je za sve operacije korekcije grešaka.
- ***Buffer*** je memorijski međuspremnik, za baratanje s podacima unutar kontrolera, prema i od *DRAM* memorije sâmog SSD uređaja.

Važno je razumjeti da *SSD kontroleri*, koji su zaduženi za rad cijelog SSD diska *(drivea)* nisu svi isti, niti brzinom niti funkcionalnostima (ili stabilnosti).

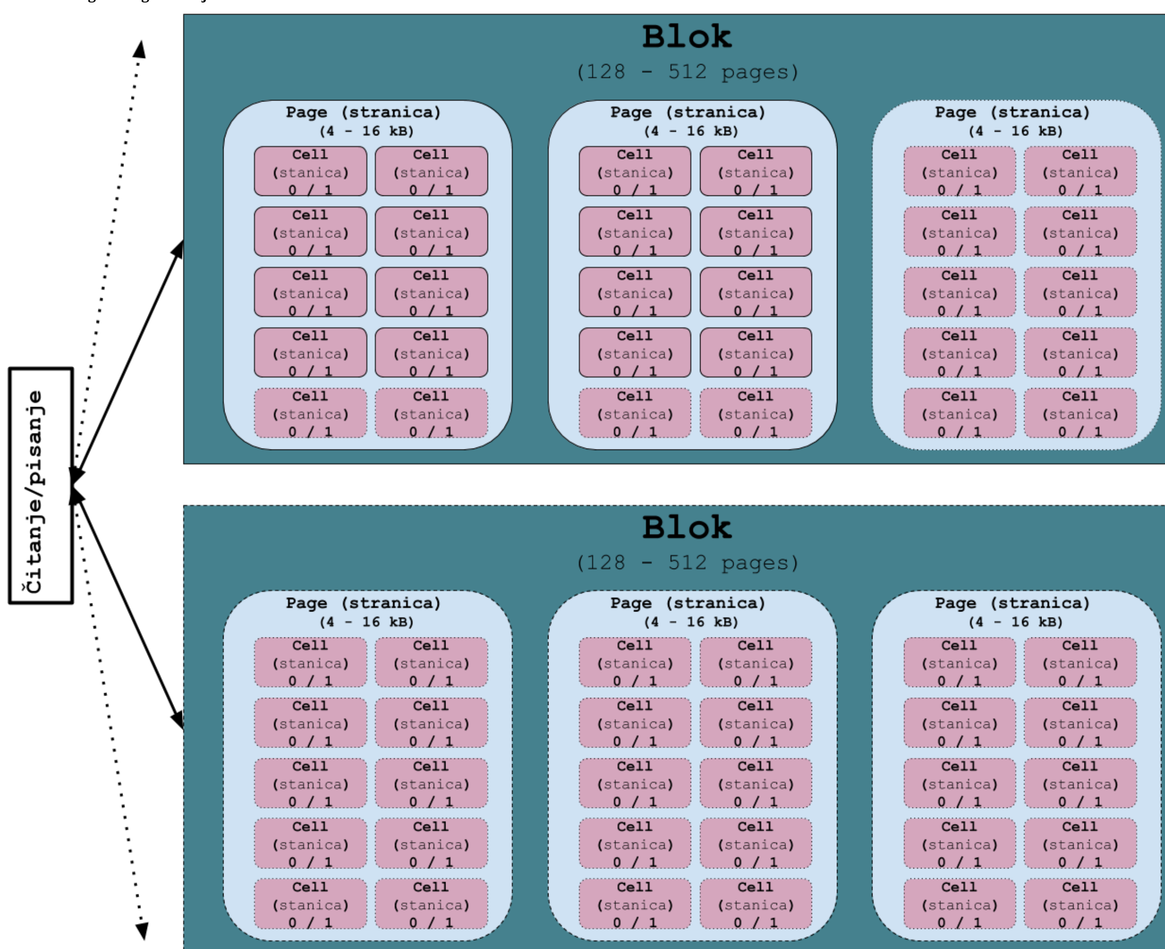
**Izvori informacija: (998),(999),(1007).**

### 13.3.1.1. Rad SSD diskova

Kod SSD diskova, podaci se spremaju u *flash* memoriju, i to u takozvane memorijske elemente odnosno takozvane ćelije (*memory cell*) o kojima smo već govorili. Memorijske ćelije su grupirane u stranice (*pages*) koje su obično veličine između 4kB i 16kB, slično kao stranice sustava virtualne memorije. Ove stranice se potom grupiraju u blokove.
Unutar ovih blokova se nalazi obično između 128 i 512 stranica. Dakle veličina bloka varira između 512kB i 8MB.
U prosjeku se ipak radi o 512 stranica od 4kB, što znači 2MB prostora za pohranu podataka unutar jednog bloka.
Međutim *flash* memorijske ćelije mogu biti zapisane sâmo, ako su prazne. Ako se pak na njima nalazi nešto zapisano, a mi to želimo prepisati, odnosno promijeniti ili zapisati nešto drugo, potrebno je prvo njihov sadržaj obrisati, prije sâme operacije zapisivanja. Dodatno operacija zapisivanja ne može se raditi na razini svake pojedine memorijske ćelije, već se može raditi samo i isključivo na razini cijelog bloka. Zbog ovog hardverskog ograničenja, zapisivanje na prazan (neiskorišten) prostor, se brže odrađuje, jer se ne mora prvo raditi brisanje. S obzirom na to kako je prije promjena na razini memorijskih ćelija, potrebno brisanje cijelog bloka sa svim memorijskim stranicama i ćelijama unutar tog bloka, jasno je kako bi se tada izgubili svi podaci koje ne mijenjamo, a nalaze se unutar tog bloka.
Stoga se prije operacije brisanja cijeli blok kopira u priručnu memoriju SSD diska. Potom se cijeli blok briše.
A tek zatim se zapisuje cijeli (novi) blok, koji sada sadrži naše promjene.
Ova operacija se zove: *read-erase-modify-write* ciklus u četiri koraka, a navedeni problem se naziva *write amplification*.

Pogledajmo i ono što bi mogli zvati geometrijom *SSD* diska, prema terminologiji za klasične tvrde diskove, na slici 112.

**Slika 112. Pogled na geometriju odnosno blokove SSD diska.**




### 13.3.1.2. Overprovisioning

Moderni *SSD* kontroleri vode brigu o tome, kako bi podatke uvijek zapisivali na blokove koji su već obrisani, koliko god je to moguće. Većina *SSD* diskova danas se izrađuju tako da im je interni kapacitet nešto veći od onoga koji je vidljiv operativnom sustavu. Njihov stvarni kapacitet je obično nešto veći (cca. *7%*), a kod poslužiteljski inačica može biti i znatno veći. Naime kontroler unutar diska se stalno brine o tome kako bi se zapisivanje radilo na blokove koji su već obrisani te briše blokove koji se mogu brisati. U tom procesu se koriste blokovi *viška*. S obzirom na to da diskovi većinom imaju nešto veći unutarnji kapacitet, na njemu zbog navedenog mehanizma, gotovo nikada ne može doći do pisanja po blokovima koji nisu prethodno obrisani. Ova tehnologija se zove ***overprovisioning***, a interni kapaciteti ovakvih diskova mogu biti nešto veći ili čak dvostruko veći od vanjskog (vidljivog) kapaciteta diska. Jedan od primjera su diskovi od recimo 120GB, kod kojih je interni kapacitet 128GB, pa se prema tome 8GB koriti kao *overprovisioning*. Drugi primjer bi bio disk kapaciteta 100GB, čiji je interni kapacitet 128GB itd. Dodatno, ova razlika kapaciteta koristi se i za zamjenu neispravnih blokova, spremanje *firmwarea* diska i slično.

### 13.3.1.3. TRIM funkcionalnost

**TRIM** funkcionalnost je definirana prema (S)ATA standardnu, dok se za SCSI standard (i za SAS) ona naziva **UNMAP**.
Ona omogućava operativnom sustavu da obavijesti SSD disk kada neki blokovi podataka više nisu u upotrebi te ih je moguće obrisati. Operativni sustav ih tada označava kao slobodne te šalje ***TRIM*** naredbu ***SSD*** disku koji ih tada može stvarno i obrisati. Ovime se ublažava problem SSD diskova s *read-erase-modify-write** ciklusom kod brisanja podataka, jer se dijelovi diska koji više nisu potrebni tada mogu odmah i obrisati, te se kod ponovnog zapisivanja na njih u budućnosti ne moraju prolaziti sva četiri navedena koraka odnosno cijeli ciklus brisanja*.
**TRIM** se koristi samo na SSD diskovima, jer kod tradicionalnih tvrdih diskova (HDD), proces brisanja podataka funkcionira drugačije, i nema koristi od slanja naredbe slične TRIM-u.

**DISCARD** je općenitiji termin koji operativni sustav koristi za brisanje podataka s blok uređaja (uključujući SSD-ove i neke druge medije). Iako je TRIM specifičan za SSD-ove, DISCARD naredba može raditi i s drugim vrstama uređaja, kao što su:

- Bilo koji diskovi uključujući mehaničke (HDD) čiji datotečni sustav podržava tzv. ***thin provisioning***, što se koristi u nekim specijaliziranim sustavima ili virtualizacijskim okruženjima gdje operativni sustav može obavijestiti sustav za pohranu da se neki blokovi više ne koriste. Ovaj princip koristi se za optimizaciju prostora u sustavima za pohranu s naprednim značajkama. Međutim, na mehaničkim diskovima koji nemaju podršku za ***thin provisioning*** ili slične funkcije, DISCARD nema stvarnu funkciju jer tradicionalni mehanički diskovi nemaju ograničenja u ciklusima brisanja/pisanja na isti način kao SSD-ovi. Vezano za datotečne sustave koji podržavaju DISCARD, to su:
  - **LVM2** (*Logical Volume Manager*) može koristiti DISCARD opciju kako bi informirao svoje SSD diskove da su određeni blokovi neiskorišteni. Kad LVM pošalje DISCARD, ako njegov diskovni uređaj podržava TRIM (npr. SSD disk), onda se zapravo izvršava TRIM naredba.
  - **ZFS** također podržava DISCARD mehanizme, iako je implementacija TRIM-a i DISCARD-a u ZFS-u nešto kompleksnija jer ZFS ima vlastiti način upravljanja prostorom. U novijim inačicama ZFS-a moguće je omogućiti TRIM putem `zpool trim` naredbe, dok za DISCARD podršku postoji opcija da ZFS proslijedi DISCARD naredbu svojim diskovnim uređajima (diskovima) ako ga podržavaju.
- **RAID** sustavi koji podržavaju SSD-ove ili ***thin provisioning*** mogu koristiti DISCARD naredbe za optimizaciju korištenja (brisanja) prostora na medijima za pohranu.



***TRIM funkcionalnost je upotrebljiva sâmo na operativnim sustavima koji ju podržavaju, pod uvjetom da je SSD disk u BIOS/UEFI konfiguriran u AHCI/SATA načinu rada te da SSD disk to podržava (što je očekivano)!***

***DISCARD*** je općeniti termin koji Linux koristi za brisanje podataka s blok uređaja (uključujući ***SSD***-ove, ali i neke druge medije/sustave). Iako je ***TRIM*** specifičan za SSD-ove, **DISCARD** značajka može raditi i s drugim vrstama uređaja poput ***RAID*** kontrolera, ali i datotečnih sustava ili naprednih sustava za pohranu (tzv. *volume menadžera)* poput ***LVM*** ili ***ZFS***,

## *Podrška za TRIM i DISCARD na razini diska*

***TRIM*** i ***DISCARD*** opcije na razini diska funkcioniraju samo i isključivo ako ih fizički disk podržava.
To možemo vidjeti upotrebom naredbe `lsblk` na sljedeći način:

```
lsblk --discard
NAME DISC-ALN DISC-GRAN DISC-MAX DISC-ZERO
sda         0      512B       2G         0
```

← U ovom slučaju u stupcima `DISC-GRAN` i `DISC-MAX` imamo vrijednosti veće od 0. što znači da imamo podršku za TRIM.
Drugi način provjere jest ručnim pokretanjem naredbe `fstrim` koja će ručno pokrenuti TRIM mehanizam na svim montiranim diskovima koji podržavaju TRIM (naravno podrška mora postojati i na razini datotečnog sustava):

```
fstrim -v
```

← U slučaju da neki od diskova ili datotečnih sustava na diskovima ne podržava TRIM, ovdje ćemo vidjeti upozorenje o tome.
Na novijim distribucijama Linuxa (pr. **Red Hat 8+**) preporuča se korištenje posebnog ***Systemd*** servisa (**`fstrim.timer`**) zaduženog za periodičko pokretanje *fstrim* mehanizma (zadano je jednom tjedno). Trajno omogućiti ovaj servis možemo sa:

```
systemctl enable fstrim.service
```

## *Podrška za TRIM i DISCARD na razini datotečnog sustava*

***TRIM*** i ***DISCARD*** opcije funkcioniraju različito ovisno o datotečnim sustavima, jer svaki od njih ima različit način upravljanja blokovima i oslobađanja prostora.
Neki datotečni sustavi bolje podržavaju ove funkcije, dok drugi mogu imati ograničenu podršku.
Pogledajmo popis ključnih datotečnih sustava koji podržavaju TRIM i DISCARD:

- **ext4** datotečni sustav:
  - **Podrška za TRIM - ext4** u Linuxu podržava TRIM putem opcije `discard` prilikom montiranja ili periodičnim pokretanjem naredbe `fstrim`. Preporučuje se koristiti `fstrim` radi performansi jer kontinuirani (online) TRIM može utjecati na brzinu zapisa.
  - **Vrsta TRIM-a** – može koristiti i "online" (uz opciju `discard` u `/etc/fstab`) i "*offline*" TRIM (putem naredbe `fstrim`).

- **XFS** datotečni sustav:
  - **Podrška za TRIM** – XFS ima dobru podršku za TRIM, posebno za velike diskove. Podržava i online (putem opcije `discard` prilikom montiranja) i offline TRIM (pokretanjem naredbe `fstrim`).
  - **Preporuka:** kao i kod **ext4**, `fstrim` je bolji izbor radi optimizacije performansi.

- **ZFS** datotečni sustav:
  - **Podrška za TRIM** – ZFS je dodao TRIM podršku u novijim inačicama. TRIM se može pokrenuti ručno ili automatski, a podržava *offline* i periodički TRIM putem naredbe `zpool trim`.

Pogledajmo i tablicu sa sažetkom:

| Datotečni sustav | Podrdka za TRIM | Online TRIM (opcija `discard`) | Offline TRIM (naredba `fstrim`) | Preporučena metoda |
|---|---|---|---|---|
| **ext4** | Da | Da | Da | Offline (`fstrim`) |
| **XFS** | Da | Da | Da | Offline (`fstrim`) |
| **Btrfs** | Da | Ne | Da | Offline (`fstrim`) |
| **F2FS** | Da | Da | Da | Online i Offline |
| **ZFS** | Da | Ne | Ne | Offline (`zpool trim`) |
| **FAT/exFAT** | Ne | Ne | Ne | - |

## *Virtualizacija (Hipervizor i virtualno računalo)*

***TRIM*** i/ili ***DISCARD*** se mogu koristiti i za virtualna računala koja na ***hipervizoru*** koriste ili **SSD** diskove ili tzv. *Thin provisioning* sustave (pr. ***LVM Thin,*** **ZFS** i slične) za pohranu diskovnih slika, kako bi njima naložila brisanje.
U KVM/QEMU virtualizacijskim okruženjima, TRIM i DISCARD opcije ovise o nekoliko faktora, uključujući format virtualnih diskova (**QCOW2** ili **RAW**), korištenje **VirtIO** diskova te konfiguraciju gostujućeg operacijskog sustava (virtualnog računala).

**VirtIO diskovi**
Za optimalne performanse i podršku za TRIM/DISCARD, preporuča se korištenje ***VirtIO*** upravljačkih programa (`virtio-blk` ili `virtio-scsi`). Pri tome `virtio-blk` podržava TRIM, ali `virtio-scsi` ima napredniju podršku za DISCARD naredbe, uključujući i bolju kompatibilnost s različitim datotečnim sustavima.

**Format diskova za virtualna računala**

- ***QCOW2*** podržava TRIM i DISCARD, ali samo ako je virtualni disk kreiran s opcijom `-o preallocation=off`. TRIM može osloboditi prostor unutar QCOW2 datoteke kada se koristi naredba `fstrim` na virtualnom računalu, čime se potom smanjuje veličina diska na hipervizoru. Naime ako koristimo QCOW2 format, preporučuje se da disk bude kreiran bez *prealokacije*, jer TRIM neće raditi ispravno ako je disk unaprijed alociran. Pogledajmo kako se kreira QCOW2 disk bez *prealokacije*:

```
qemu-img create -f qcow2 -o preallocation=off /putanja/do-diska/disk.qcow2 20G
```

- **RAW** – TRIM i DISCARD funkcioniraju na RAW formatima diskova, ali osloboditi prostor može biti teže jer je veličina RAW datoteke fiksna, čak i nakon brisanja podataka na virtualnom računalu (sustavu). Postoji mogućnost korištenja vanjskih alata za optimizaciju prostora na RAW diskovima.

**Kako TRIM/Discard radi s virtualnim diskovima?**

- **QCOW2 diskovi** – kada virtualni operativni sustav šalje TRIM/DISCARD naredbu (npr. kroz `fstrim`), disk označava neiskorištene blokove koje *hipervizor* može osloboditi, ali se veličina QCOW2 datoteke smanjuje samo ako postoji preostali prostor za oslobađanje. To se događa samo u slučajevima kada `preallocation` nije aktiviran.
- **RAW diskovi** – u slučaju RAW diskova, veličina datoteke na hipervizoru ne mijenja se automatski prilikom TRIM operacija. Iako operativni sustav unutar virtualnog računala može prepoznati slobodan prostor, RAW datoteka ostaje iste veličine.

**O čemu sve ovisi TRIM unutar virtualnog računala?**
TRIM značajka unutar virtualnog računala ovisi o sljedećem:

- **Datotečni sustav virtualnog računala** – kako bi TRIM radio, datotečni sustav unutar virtualnog računala mora podržavati TRIM, To primjerice podržavaju: **ext4**, **XFS**, **ZFS**.

- **KVM/QEMU** - novije inačice KVM-a i QEMU-a bolje podržavaju TRIM kroz *VirtIO-blk* i *VirtIO-scsi*, dok starije inačice mogu imati ograničenu podršku. To je potrebno i postaviti u opcijama virtualnog računala na strani hipervizora. Primjerice KVM/QEMU opcije: `discard=on` ili `discard=unmap` i `detect-zeroes=unmap`. Pri tome opcija `detect-zeroes=unmap` omogućuje da QEMU "otkrije" nulte blokove prilikom pisanja na disk i interpretira ih kao prazne, omogućujući automatsko oslobađanje prostora. Kada virtualno računalo piše nulte vrijednosti na disk, ta operacija se prepoznaje, a nulti blokovi se tretiraju kao "nepotrebni", oslobađajući prostor kroz DISCARD naredbu.
  Dok je opcija `discard=on` općenita opcija koja omogućava da virtualno računalo koristi bilo koji oblik DISCARD naredbe koji je podržan (može uključivati TRIM na SSD-u, UNMAP na SCSI uređaju, itd.). Dok `discard=unmap` šalje specifičnu opciju koja koristi UNMAP za upravljanje neiskorištenim blokovima, uglavnom na SCSI uređajima (za *VirtIO-scsi*).

- **Preporuke za performanse** - korištenje TRIM-a i DISCARD-a može smanjiti performanse, ako je uključen kontinuirano (npr. kroz `-o discard` prilikom montiranja).
  Preporuča se korištenje periodičnog `fstrim`.

### SSD Emulacija

SSD emulacija u virtualizaciji, na razini hipervizora, signalizira virtualiziranom operativnom sustavu (OS-u unutar virtualnog računala) da koristi specifične optimizacije za SSD diskove, iako fizički disk na hipervizoru (domaćinu) može biti i mehanički (što se naravno u ovom slučaju ne preporučuje). Virtualizirani OS tada prepoznaje disk kao SSD i prilagođava svoje I/O algoritme kako bi smanjio pristupne latencije i optimizirao performanse. Dakle kada je omogućena SSD emulacija, *hipervizor* (QEMU/KVM) prezentira disk kao SSD uređaj gostujućem OS-u. On (virtualizirani OS) tada koristi specifične algoritme za SSD diskove; kao što su optimizacije kod zapisivanja podataka i razmještanje podataka po disku, jer smatra da radi s SSD diskom.

**Zaključak:**

- ***DISCARD (TRIM/UNMAP)*** se koristi za oslobađanje neiskorištenog prostora na disku i smanjenje fragmentacije, što je važno za SSD-ove i QCOW2 slike diskova. Njegova funkcionalnost ovisi o podršci datotečnih sustava i hipervizorskog sustava odnosno fizičkog računala.
- ***SSD emulacija*** omogućuje gostujućem OS-u da koristi optimizacije specifične za SSD-ove, što može poboljšati performanse I/O operacija unutar virtualnog računala, čak i ako fizički disk nije SSD.

Pogledajmo ključne razlike između SSD emulacije i DISCARD/TRIM funkcija:

| Funkcionalnost | DISCARD (TRIM/UNMAP) | SSD Emulacija |
|---|---|---|
| **Svrha** | Oslobađanje neiskorištenog prostora na disku. | Signaliziranje gostujućem OS-u (VM) da koristi SSD optimizacije. |
| **Razina djelovanja** | Komunikacija između gostujućeg OS-a (VM-a) i *hipervizorovog* diska. | Promjena načina na koji gostujući OS (VM) vidi disk. |
| **Podržani formati** | VirtIO-blk,<br>VirtIO-scsi,<br>QCOW2,<br>RAW. | VirtIO-blk,<br>VirtIO-scsi. |
| **Utjecaj na performanse** | Povećanje performansi diska (povremeno smanjenje performansi kod TRIM-a u realnom vremenu). | Optimizira performanse kroz bolji I/O model. |
| **Implementacija** | Dodavanje opcija u QEMU konfiguraciju diska:<br>`discard=on` ili `discard=unmap` i `detect-zeroes=unmap` | Dodavanje SSD atributa u QEMU konfiguraciju diska. |
| **Povezanost s fizičkim diskom** | Direktno utječe na fizički disk *hipervizora* (SSD ili HDD) | Neovisno o fizičkom disku hipervizora (može biti SSD ili HDD). |



Za detalje oko virtualizacije i diskovnog podsustava pogledajte poglavlje:
**27.1.2.3. Disk kontroleri.**

**Izvori informacija: (1007),(1754),(1755),(1756),(1757),(K-8), `man lvmthin`, `man mount`, `man 5 fstab`, `man fstrim`,`man hdparm`,`man lsblk`,`man xfs`,`man ext4`, *Poglavlje: 27.1.2.3.***

## 13.3.1.4. Karakteristike SSD diskova

Pogledajmo osnovne karakteristike SSD diskova u tablici dolje:

| Parametar | Vrijednost |
|---|---|
| **Brzine zapisivanja SSD diskova** | **SAS** ili **SATA** sučelje: između 200MB/s i 500+MB/s |
| | **PCIe** brzine se kreću i preko 5.000 MB/s (pr. PCIe Gen4, x4) |
| **Brzine čitanja s SSD diskova** | **SAS** ili **SATA** sučelje: od 400+ MB/s |
| | **PCIe** sučelje: do 6 000 MB/s i više (pr. PCIe Gen4, x4) |
| **Broj ulazno izlaznih operacija prema disku** (IOPS) | **SAS** ili **SATA** sučelje: od 5.000 IOPS do 100.000 *IOPSa* |
| | **PCIe** sučelje: od 100.000 do 1.000.000 *IOPSa* |
| **Vrijeme traženja** (*Seek time*) | 0.08ms - 0.16ms |
| **MTBF** (*Mean Time Between Failure*) | 1.000.000 - 2.000.000 sati |

Kao što je i očekivano, brzine prijenosa podataka sa *SSD* diskova su znatno veće od mehaničkih diskova, jer u njima nema mehaničkih dijelova koji unose velika vremena čekanja: kao što su pomaci glave za čitanje ili pisanje, čekanje na rotaciju diska i slično.
Ovdje se radi sâmo o brzini elektronike. Iako i kod SSD diskova postoji vrijeme potrebno da se pronađu podaci, ovo vrijeme je višestruko manje, te iznosi između 0.08ms i 0.16 ms. Nadalje, broj ulazno izlaznih operacija prema disku (**IOPS**) je drastično veći jer je moguće paralelno čitati ili zapisivati veliki broj datoteka, odnosno čitati ili zapisivati na više memorijskih lokacija SSD diska istovremeno, što u konačnici uzrokuje bolje performanse.

**Izvori informacija: (1007),(K-8).**

### 13.3.1.5. Fragmentacija

Kod mehaničkih diskova, mehanička glava se mora pozicionirati na stazu unutar koje se nalazi sektor koji treba pročitati, te ako je datoteka koja se čita razlomljena na više pozicija na disku, odnosno došlo je do njene fragmentacije na više dijelova i sâmim time se nalazi na više staza, glava diska se mora pomjerati iz staze u stazu kako bi pročitala sve podatke.
Ovaj proces znatno usporava čitanje s diska.
Fragmentacija ne predstavlja problem kod SSD diskova jer nema glava koje se moraju pomjerati sa staze, tj. na potrebnu stazu, kako bi pročitale sve dijelove fragmentiranih datoteka. Kod SSD diskova čak i kada je fragmentacija prisutna, ona nije problem jer se blokovi podataka na njenoj *flash* memoriji pronalaze i čitaju ekstremno brzo, pa nema potrebe za defragmentacijom, čak niti kod upotrebe datotečnih sustava koji su podložni fragmentaciji.
Naime kod SSD diskova fragmentacija podataka postoji, ali utjecaj na performanse je potpuno drugačiji nego kod klasičnih tvrdih diskova (HDD). Razlike dolaze iz načina na koji SSD diskovi pohranjuju i pristupaju podacima.

**Kako se događa fragmentacija na SSD-u?**

- Logička fragmentacija – kada operativni sustav upisuje ili briše datoteke, podaci mogu biti pohranjeni u memorijskim blokovima koji nisu jedan pored drugoga, što je slična pojava kao kod HDD-ova. Iako se podaci mogu raspršiti, SSD diskovi imaju gotovo istu brzinu pristupa svakom memorijskom bloku, tako da ova vrsta fragmentacije ne utječe značajno na performanse.
- Unutarnji rad SSD-a – SSD diskovi koriste algoritme za izjednačavanje trošenja memorijskih ćelija (*wear leveling*) i napredne metode upravljanja prostorom, kao što su navedeni TRIM te *garbage collection*. Ovi algoritmi prebacuju podatke između ćelija kako bi ravnomjerno distribuirali zapisivanje po čitavoj memoriji, što može izgledati kao fragmentacija s logičke strane.

Zbog svega navedenog *fagmentacija* ne utječe na brzinu čitanja jer SSD-ovi mogu pristupiti bilo kojem bloku podataka bez mehaničkih ograničenja.
Međutim s vremenom, zbog mehanizama poput *garbage collectiona*, performanse pisanja se mogu pogoršati. Ako disk ima puno fragmentiranih (raspršenih) podataka, procesi oslobađanja i ponovnog organiziranja blokova mogu dovesti do povećanja latencije pri pisanju.
Tehnologije TRIM i *garbage collection* pomažu SSD-u u oslobađanju memorijskih ćelija koje sadrže nepotrebne (izbrisane) podatke. Ako TRIM nije pravilno konfiguriran ili podržan, disk će se s vremenom "ispuniti" fragmentiranim podacima i performanse će pasti. Nadalje, kada je SSD gotovo pun, performanse pisanja se mogu pogoršati jer disk mora provoditi više vremena reorganizirajući podatke.

**Izvori informacija: (1007),(1758),(1759),(K-8).**

### 13.3.1.6. Pristup i adresiranje prostora na SSD diskovima

Kao i kod mehaničkih (rotacijskih diskova) danas se za adresiranje prostora na disku, kao i za pristup podacima na disku ne koristi direktna shema adresiranja, kao što je za mehaničke diskove bila **CHS** shema; prema cilindrima, glavama i sektorima. Umjesto toga koristi se **LBA** shema (*Logical block addressing*) koja je posrednik između operativnog sustava i sâmog diska.
Naime kod mehaničkih diskova i dalje postoji fizička podjela i adresiranje prema cilindrima, glavama i sektorima jer se tako i zapisuju podaci na njih, ali se ta shema preslikava u novu **LBA** shemu, kojoj pristupa operativni sustav.

Isti slučaj je i sa SSD diskovima, koji nemaju cilindre, glave i sektore ali imaju adresiranje prema blokovima, stranicama i ćelijama, koje se opet preslikava u **LBA** adrese. Na ovaj način se sakrila interna struktura diska, te pojednostavio pristup.
Kao i kod mehaničkih diskova, nakon što se pokrene operativni sustav, on se (što vrijedi za sve moderne operativne sustave) više ne oslanja na *BIOS* za pristup disku.
Naime on nadalje koristi direktan pristup disku, bez posredovanja *BIOSa* za razlučivanje adresiranja blokova te stranica i ćelija SSD diska.

### 13.3.2. NVMe diskovi

**NVM Express** (**NVMe**) ili *Non-Volatile Memory Host Controller Interface Specification* (**NVMHCIS**) otvorena je specifikacija sučelja logičkog uređaja za pristup mediju za pohranu podataka na računalo, koji se obično povezuje putem *PCI Express* (**PCIe**) sabirnice. Akronim **NVM** označava vrstu memorije, koja je često **NAND** *flash* memorija koja dolazi u nekoliko fizičkih oblika, uključujući: **SSD** diskove (SSD), *PCI Express* (**PCIe**) ili **M.2** kartice.
Pri tome su **M.2** kartice odnosno format, nasljednika **mSATA** kartica.
Naziv **NVM** (*Non-Volatile Memory*) odnosi se na vrstu računalne memorije koja može zadržati pohranjene informacije (podatke) čak i nakon uklanjanja napajanja odnosno nakon gašenja računala. Nasuprot tome, primjerice **RAM** memorija treba stalno napajanje kako bi zadržala podatke, odnosno ona čuva podatke u svojoj memoriji, sve dok je računalo uključeno.

**NVM Express** je tako dizajniran da omogućuje hardveru i softveru računala da u potpunosti iskoriste višestruke razine paralelizma u operacijama čitanja ili zapisivanja.

*NVM Express* uređaji danas obično dolaze u dvije izvedbe:

- Kao standardne *PCI Express* (**PCIe**) kartice, kako je vidljivo na slici 112.1 (za ***Intel DC P3608*** karticu).
- Ili kao **M.2** varijanta, za spajanje u **M.2** utor, kako je vidljivo na slici 112.2 (za **Samsung 980 Pro M.2** karticu).

**Slika 112.1. Pogled na NVMe disk tvrtke Intel (DC P3608), koji se spaja na PCI Express sabirnicu (x8).**

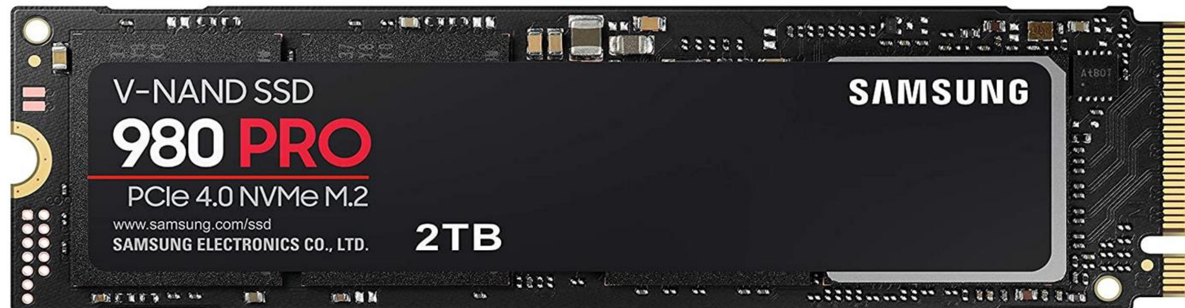



Karakteristike **1.6TB NVMe** diska:
**(Intel DC P3608)**

| | |
|---|---|
| **Brzina čitanja** | 5 GB/s |
| **Brzina zapisivanja** | 2 GB/s |
| **IOPS** – čitanje | 850.000 |
| **IOPS** – zapisivanje | 150.000 |
| **Sučelje** | PCI Express 3.0 x8 |
| **Latencija** | 20µs |

Kod dizajna ***NVMe*** diskova za poslužitelje, obično se na ulazu nalazi ekstremno brza RAM memorija. U nekim uređajima se koristi i dodatna ***SLC*** *flash* međumemorija.
Međutim u ovom uređaju se prvo kao međumemorija nalazi brza RAM memorija (konkretno **5GB** DDR3-1600), a u konačnici se sve snima na posebnu varijantu **MLC** memorije. Ovdje su vidljivi i kondenzatori koji napajaju RAM memoriju, a koriste se u slučaju nestanka električne energije (*Battery Backup Unit* [**BBU**]).

**Slika 112.2. Pogled na NVMe (Flash) disk tvrtke Samsung (980 Pro), koji se spaja na M.2 sučelje (vidljivo s desne strane diska).**




Povijesno gledano, većina SSD diskova koristila je sučelja kao što su **SATA**, **SAS** ili **Fiber Channel** za međusobno povezivanje s ostatkom računala. Pojavom SSD diskova i činjenicom kako su postali sve dostupniji na tržištu, SATA je postao najtipičniji način povezivanja SSD diskova na osobnim računalima.

Međutim, **SATA** je dizajniran prvenstveno za rad s mehaničkim tvrdim diskovima, te je postajao sve neadekvatniji za SSD-ove, koji su se s vremenom poboljšali u brzini prijenosa podataka i u mogućnosti paralelne obrade više zahtjeva za čitanjem ili zapisivanjem (tzv. **IOPS** operacije).

Kako smo već naučili *PCI Express **(PCIe)*** je sabirnica za spajanje raznih komponenti računala (grafička, mrežna i druge kartice), donekle i poput **SATA**; jer se obje koriste za povezivanje pojedinih komponenti s ostatkom računala.
Međutim **SATA** je disk kontroler, i on još eventualno treba i **AHCI** da bi tvrdi disk ili **SSD** disk mogli na napredan način komunicirati s računalnim sustavom (primjerice zbog **TRIM** i **NCQ** značajki).
Isto tako za spajanje ***NVMe*** diskova na ***PCIe*** sabirnicu, potreban je ***NVMe*** diskovni kontroler.

Povijesno gledano, zbog navedenih ograničenja **SATA** standarda došlo je do razvoja **NVMe** standarda koji se počeo razvijati u drugoj polovici 2009 godine. **NVMe** specifikacije razvila je radna skupina naziva. „*NVM Express Workgroup*", koja ima više od 90 tvrtki članova. Tijekom 2012 godine je napravljen prvi *chipset* (praktično **NVMe** *diskovni kontroler*), a već 2013 i prvi **NVMe** disk (*Samsung XS1715*). Kako se ovaj standard razvijao, od inačice do inačice, u njega su dodavane nove značajke.

## Usporedba s AHCI

*Advanced Host Controller Interface* (**AHCI**) i **SATA** imaju svoje prednosti, ali i veliku manu, jer su razvijani za mehaničke (rotacijske) diskove, a ne za **SSD** kategoriju diskova kojoj pripadaju i **NVMe** diskovi.
S druge strane, sučelje **NVMe** uređaja dizajnirano je od početka, iskorištavajući nisku latenciju i paralelizam ***PCI Express*** *SSD* diskova, te nadopunjujući paralelizam suvremenih (višejezgrenih) procesora, operativnih sustava i aplikacija.

Dakle osnovne prednosti **NVMe**-a nad **AHCI**-jem odnose se na njegovu sposobnost iskorištavanja paralelizma u hardveru i softveru računala, što se očituje razlikama u dubinama reda čekanja naredbi (pri zapisivanju i čitanju), učinkovitosti korištenja signala prekida (**IRQ**) i drugim optimizacijama, što rezultira mnogim poboljšanjima u performansama.

Pogledajmo usporedbu nekoliko osnovnih **AHCI** i **NVMe** značajki:

| | AHCI | NVMe |
|---|---|---|
| ***Maksimalna duljina reda čekanja i broj redova čekanja (Queue depth)*** | **1** red čekanja za naredbe (command queue)<br>**32** naredbe u redu čekanja | **65.535** redova čekanja za naredbe<br>**65.535** naredbi u redu čekanja |
| ***Uporaba signala prekida (IRQ)*** | **1** signal prekida (**IRQ**) | Do **2.048 MSI-X** signala prekida |
| ***Paralelizam i višenitnost*** | Zahtjeva blokiranje/zaključavanje (*synchronization lock*) za izvršavanje naredbi | Ne blokirajući |
| ***Prijenos podataka*** | ***Half-Duplex*** | **Full-Duplex** |

**Izvori informacija: (999),(1079),(1080), poglavlja u knjizi: 10.1.1, 10.8 i 13.3.1.**

## 13.3.2.1. NVMe diskovi i Linux

Mi koristimo kernel **4.20**, a za upotrebu naprednih postavki **NVMe** diskova, potrebno je instalirati dodatni softverski paket:

```
yum -y install nvme-cli
```

Nakon instalacije, dobit ćemo nekoliko programa za rad i optimizaciju *NVMe* diskova. *NVMe* diskovi za rad se oslanjaju na kernel module imena: `nvme`, i `nvme_core`, a podrška za ostalo je u sâmom kernelu. Pogledajmo s čim je kompiliran kernel:

```
grep -i nvme /boot/config-`uname -r`
```

Ovdje su najvažniji: `CONFIG_BLK_DEV_NVME` (blok uređaj), `CONFIG_NVME_CORE` (jezgra *NVMe sustava*) i drugi.

Provjerimo koje sve *NVMe* uređaje imamo na računalu, upotrebom naredbe `nvme` (skratili smo ispis):

```
nvme list

Node           SN                Model                Namespace Usage      Format   FW Rev
-----------  -------------  ------------------- --------- ----------  -------  ---------
/dev/nvme0n1 S4EVNF0M454811Z Samsung 970 EVOPlus    1     202GB/500GB 512B+0B 1B2QEXM7
```

Ovdje vidimo sam uređaj (`/dev/nvme0n1`). Inicijalno se ***NVMe*** diskovi vide kao `/dev/nvmeXnY` uređaji (**X**=oznaka diska, **Y**=*namespace*), iako je cijeli disk vidljiv i kao `/dev/nvmeX`. Primjerice prvi *NVMe* disk je vidljiv kao `/dev/nvme0`.
Potom vidimo njegov serijski broj (`S4EVNF0M454811Z`), model uređaja i druge informacije, poput revizije *firmware-a* (`1B2QEXM7`). *Nadogradnju firmware-a isto možemo odraditi s naredbom* `nvme`. *NVMe namespace* je konstrukt u *NVMe* tehnologiji koja sadrži korisničke podatke.

Naime *NVMe* kontroler može koristiti i više ovih prostora na disku (*NVMe namespace-a*). Pri tome svaki od ovih prostora sadrži svoj opseg logičkih blokova adresa, koje su izolirane od drugih prostora.
Međutim većina *NVMe* diskova danas koristi samo jedan prostor naziva, ali u virtualizaciji i za sigurnosne primjene, moguće je imati slučajeve korištenja više prostora naziva (*Namespaces*).

Za potrebe rada s *namespace-ovima*, mogu se koristiti naredbe `nvme create-ns` **`XY`** i `nvme delete-ns` **`XY`** (**XY** je ime *NVMe* uređaja).

Ako želimo više informacija o konkretnom *NVMe* uređaju (`/dev/nvme0n1`) možemo pokrenuti naredbu `nvme` ovako:

```
nvme id-ctrl -H /dev/nvme0n1
```

Ako želimo vidjeti stanje uređaja (temperatura, greške,…), preko *S.M.A.R.T.* sustava, tada za isti uređaj pokrenimo:

```
nvme smart-log /dev/nvme0n1
```

Moguće je primjerice napraviti i format niske razine, cijelog **NVMe** diska (**oprez jer ćete izgubiti sve podatke na njemu**):

```
nvme format /dev/nvme0n1
```

Pogledajmo kako je vidljiv *NVMe* disk `/dev/nvme0n1`, koji ima dvije particije, što ćemo vidjeti s naredbom `lsblk`:

```
lsblk /dev/nvme0n1
NAME          MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
nvme0n1       259:0     0 465.8G  0 disk
├─nvme0n1p1   259:1     0   460G  0 part
└─nvme0n1p2   259:2     0   5.8G  0 part
```

Dakle particija u oznaci diska dodaje `p`**`X`** (**X** je broj particije). Pogledajmo i kako se *NVMe* disk vidi s naredbom `lsscsi`:

```
lsscsi | grep nvme
[N:0:4:1]    disk    Samsung SSD 970 EVO Plus 500GB__1       /dev/nvme0n1
```

***Nadogradnja firmware-a NVMe diska ovisi o proizvođaču. Pri tome neki od proizvođača nude firmware datoteku, a neki ISO sliku s koje se pokreće sustav i automatska nadogradnja firmware-a, bez puno posredovanja korisnika.***
Nadogradnja *firmware*-a, ako imamo datoteku s *firmware*-om, u načelu se radi u nekoliko koraka, koje ovdje nećemo opisivati.

## *Pooling* i druge optimizacije

Od Linuxa kernela 4.20 došlo je do optimizacija **NVMe** upravljačkog programa kako bi se omogućio novi parametar koji upravlja dohvaćanjem podataka s diska. Ovdje se radi o takozvanom *Pooling* mehanizmu, čijom uporabom se ne bi trebali koristiti signali prekida (**IRQ**), slično kao kod *Pooling* mehanizma za rad mrežne kartice. To je dovelo i do pojave višestrukih redova za čekanje/izvršavanje (Engl. *poll queues*) koji su sada dostupni u inačici kernela 4.20 i novijih.
Dakle od kernela 4.20, upotrebu *poll queues* mehanizma radi se na razini kernel modula zaduženog za **NVMe**.

Pogledajmo kako povećati broj redova čekanja (za naredbe) na ukupno četiri:

```
modprobe -r nvme && modprobe nvme poll_queues=4
```

Odnosno za trajnu konfiguraciju, otvorite datoteku `/etc/modprobe.d/nvme.conf` i dodajte sljedeći redak:

```
options nvme poll_queues=4
```

Ovime smo povećali broj redova čekanja, pri čemu za svaki red čekanja može biti zadužena druga jezgra procesora.
Preporuka je kreirati do maksimalno onoliko redova čekanja koliko imamo dostupnih jezgri procesora. Međutim, najbolje je napraviti testiranje prije konačne odluke, jer samo povećanje nije uvijek najbolji odabir, ovisno o namjeni poslužitelja/računala. Unutar svakog reda čekanja moguće je imati do teoretski maksimalno 65.535 spremnika za naredbe u redu čekanja.

Nakon ovog koraka potrebno je rekreirati inicijalni RAM disk (*initrd*), sa sljedećom naredbom:

```
dracut –force
```

I potom restartati računalo. Zbog lakšeg pronalaska konfiguracijskih datoteka u */sys/* pseudo datotečnom sustavu instalirajmo:

```
yum -y install sysfsutils
```

Sada možemo jednostavno provjeriti status kernel modula `nvme`. To ćemo učiniti s novo instaliranom naredbom `systool`:

```
systool -vm nvme | grep poll_queues
    poll_queues          = "4"
```

Vidimo da imamo postavljenu vrijednost `4` što smo i očekivali. **Moguće su i mnoge druge optimizacije NVMe sustava!.**

**Datotečni sustavi i NVMe diskovi**
Iako većina datotečnih sustava uredno radi sa ***NVMe*** (i **SSD**) diskovima, oni nisu previše optimizirani za ovu relativno novu vrstu diskova. Stoga je u novije vrijeme razvijen datotečni sustav koji je posebno dizajniran za ***NVMe*** i **SSD** diskove, a zove se ***SSDFS***. Njegove trenutne prednosti (3.2023.g.) po pitanu rada s *NVMe* i *SSD* diskovima su:

- Povećava životni vijek **NVMe/SSD** diskova:
  - Smanjivanjem I/O operacija zapisivanja.
  - Smanjuje *read-erase-modify-write* ciklus (tzv. *amplification* faktor).
- Osim toga on podržava: *deduplikaciju*, izradu snimke stanja u vremenu (*snapshot*), napredne Linux atribute i druge opcije iskorištavajući specifičnosti ***NVMe/SSD*** diskova.

Osim **SSDFS** datotečnog sustava na Linuxu je u razvoju i **F2FS** (*Flash-Friendly File-System*) datotečni sustav prilagođen za SSD i NVMe diskove.

**Izvori informacija: (1081),(1082),(1083),(1084),(1448), `man systool`, `man dracut`, `man modprobe`, `man nvme`, `man lsscsi`, `man lsblk`, `man yum`.**

# 13.4. Logička shema diska

Prema logičkoj shemi diska, svaki tvrdi (ili ***SSD***) disk na sâmom početku diska, u prvom sektoru (koji je veličine 512 bajta) ima tzv. *Master Boot Record* (***MBR***). *MBR* je poseban zapis u prvom sektoru diska s kojega, tijekom pokretanja računala, *BIOS* računala počinje čitati podatke. U *Master Boot Record*u (***MBR***) se nalazi mali program koji pokreće ostatak sustava te čita particijsku tablicu diska. U nastavku provjerava koja od particija je označena kao aktivna i u konačnici s nje nastavlja proces pokretanja operativnog sustava. Ovaj mali program se naziva ***boot loader***.



Za osnove rada ***boot loadera*** i proces pokretanja računala, pogledajte poglavlje:
**12.1.2. Boot proces detaljnije.**

***Particija*** predstavlja logičku cjelinu tvrdog diska koju operativni sustav može koristiti. *Particija* je i najmanja logička cjelina na koju se može instalirati datotečni sustav. Dakle datotečni sustavi poput: ***ext2***/***3***/***4***, ***XFS***, ***ZFS*** i drugih se mogu instalirati sâmo na *particiju* na disku. Prema tome, sve dok disk nije podijeljen na *particije* (minimalno jednu), na njega se ne može instalirati datotečni sustav. Definicija svih *particija* na disku se nalazi u particijskoj tablici (engl. *partition table*), koja se nalazi u **MBR** zapisu (kod upotrebe *MBR* sheme particioniranja). *Particioniranje* diska odnosno logičko rezanje diska (engl. *slicing*) jest stvaranje jedne ili više regija na tvrdom disku, SSD ili nekom drugom mediju za pohranu podataka, tako da operativni sustav može zasebno upravljati informacijama u svakoj ovakvoj regiji. Drugim riječima, ove regije diska se nazivaju particije.
To je obično prvi korak pripreme novo ugrađenog diska, prije nego što smo ga uopće doveli u stanje da na njemu kreiramo strukturu direktorija (mapa) i datoteka. Disk pohranjuje informacije o lokacijama i veličinama particija na području poznatom kao particijska tablica koju operativni sustav čita prije bilo kojeg drugog dijela diska. Svaka se particija pojavljuje u operativnom sustavu kao zasebni logički disk, a koji koristi dio stvarnog (fizičkog) diska. Administratori sustava koriste razne programe za rad s particijama: za stvaranje, promjenu veličine, brisanje kao i druge manipulacije s njima.
Dalje u tekstu, govorit ćemo o takozvanom *PC stilu* odnosno vrsti particija, s kojima ćete se najčešće i susretati.
I dalje ćemo govoriti o **PC stilu** particioniranja odnosno o **MBR** vrsti, zbog lakšeg razumijevanja, a koja predstavlja stariji standard (potječe iz 1983.g). Vrlo sličan je princip i s novijim standardom koji se zove *GUID Partition Table* (**GPT**).

Važno je znati kako **MBR** ima nekoliko ograničenja:

- Unutar *MBR* zapisa nalaze se popisane particije kao i početni i završni sektor svake particije. Ovaj prostor koji služi za adresiranje svake particije je 32-bîtni broj. Dakle maksimalna veličina particija prema *MBR* shemi je, prema tome $\mathbf{2^{32}}$ za 512 *bajta* (veličina sektora) = 2 TB. Detaljnije, radi se o adresiranju koje se zove **LBA** (*Logical Block Addressing*). To uglavnom znači da ne možemo imati particiju veću od **2TB**. Netko se može zapitati: što ako imamo noviji tvrdi disk, kojemu veličina sektora nije 512 *bajta* nego 4096 bajta (što je slučaj sa svim najnovijim (velikim) diskovima). Odgovor je: ***MBR*** podržava samo 512 *bajtne* sektore.
- *MBR* podržava maksimalno 4 primarne particije. Veći broj particija je moguće imati, ako na mjestu primarne particije napravimo takozvanu proširenu odnosno ***extended*** particiju, unutar koje je moguće kreirati više logičkih particija.

*GUID Partition Table* (**GPT)** je noviji sustav, koji za sada nema ograničenja. Pogledajmo neke njegove karakteristike:

- Maksimalni broj particija je toliki da praktično nećete imati ograničenja na njihov broj (do 128 primarnih particija).
- Maksimalna veličina pojedine particije je sada 64 *bîtni* broj. Dakle za 512 *bajtne* sektore je to $2^{64}$ *bajta*, za adresiranje tj. 9.4 **ZB** (*Zetta bajta*). A za 4096 *bajtne* sektore (tj. 4KB) je to $2^{72}$ *bajta* za adresiranje tj. 19 **YB** (*Yotta bajta*).
  To znači kako limitirajući faktor postaje datotečni sustav.
- Veličina sektora može biti i 512 *bajta* i 4096 *bajta (4KB).*
- *GPT* zapis se snima na prvi i na zadnji sektor na disku (za svaki slučaj).
- *GPT* zapis sadrži i *CRC* (*Checksum*) odnosno provjerni zbroj; zbog vlastite konzistencije.

S **MBR** particijama možemo raditi sa svim programima za particioniranje, dok sa **GPT**, samo s onima novijima koji podržavaju **GPT** vrstu particija. Pod *Linux*om, za rad s **MBR** možemo koristiti sljedeće programe:

- `fdisk`, `cfdisk` i slične.

Dok za rad s **GPT** moramo koristiti novije generacije programa, poput:

- `parted` (*GNU Parted*), `gdisk` (*GPT fdisk*), `cgdisk` (*frontend*/sučelje za program `gdisk`) i druge.

Bez obzira koristimo li ***MBR*** ili ***GPT*** shemu particioniranja, svaki novi disk je potrebno pripremiti za korištenje, što podrazumijeva:

1. ***Particioniranje*** odnosno kreiranje minimalno jedne particije na disku.
2. ***Formatiranje*** odnosno instaliranje datotečnog sustava, na određenu particiju.

***Particija*** predstavlja logičku cjelinu tvrdog diska koju operativni sustav može koristiti.
Particija je najmanja logička cjelina na koju se može instalirati datotečni sustav.



Datotečni sustav (***NTFS***, ***ext2***/***3***/***4***, ***XFS***, ***ZFS*** i sl.) se može instalirati samo na *particiju* na disku, pa prema tome, sve dok disk nije podijeljen na *particije* (minimalno jednu), na njega se ne može instalirati datotečni sustav (niti operativni sustav). Definicija svih *particija* na disku se nalazi u particijskoj tablici (engl. *partition table*), koja se nalazi u **MBR** zapisu, zapisanom u prvom sektoru na disku. Prvi sektor na disku se nalazi u prvih **512** bajta na disku.



***Svaki disk mora imati minimalno jednu particiju!***
***Ne zaboravite da se datotečni sustav može instalirati samo i isključivo na particiju diska!***



***Formatiranje*** particije diska se odnosi na instalaciju datotečnog sustava na odabranu *particiju*. Nakon što smo kreirali *particiju* na disku, potrebno je instalirati datotečni sustav ovisno o operativnom sustavu na kojem to radimo.
Neki od datotečnih sustava su:

- **NTFS** - za *Windows* operativne sustave, a prepoznaje ih i Linux.
- **Ext2**, **ext3**, **ext4**, **XFS**, **ZFS, Btrfs** - za Linux operativne sustave.

Tek kada smo instalirali datotečni sustav na odabranu *particiju*, disk je spreman za upotrebu.

**Konačni kapacitet diska**
Kupnjom novog diska i spajanjem istoga na računalo, primijetiti ćete odstupanje kapaciteta diska od onoga deklariranog od strane proizvođača. Naime proizvođači diskova kapacitet označavaju dodavanjem metričkog prefiksa kako je i za očekivati i to za dekadski brojevni sustav. Pa je tako:

- 1kB = 1.000 *bajta*
- 1MB = 1.000.000 *bajta* = 1.000 kB
- 1GB = 1.000 MB
- 1TB = 1.000 GB

Sve ovo je u redu za klasične mjerne jedinice; primjerice 1kg = 1.000 g. što je točno, ali računala koriste bazu *2* odnosno binarni brojevni sustav. Stoga su odnosi malo drugačiji, pa je tako:

- 1 *kilo bajt* (**kB**) = 1024 *bajta (**b**).*
- 1 *mega bajt (**MB**)* = 1024 *kilo bajta (**kB**).*
- 1 *giga bajt* (***GB***) = 1024 *mega bajta (**MB**).*
- 1 *tera bajt (**TB**)* = 1024 *giga bajta (**GB**).*

To znači kako je disk s tvorničkom deklaracijom proizvođača, s kapacitetom 1TB nešto manji, kada se iz metričkog dekadskog sustava kapacitet preračuna u binarni. Tako je 100GB u metričkom dekadskom sustavu jednako: 100GB x 1.000 x 1.000 x 1.000 = 100.000.000.000 *bajta*. Isti kapacitet računalo, jer koristi binarni sustav, vidjelo bi kao: 100GB x 1024 x 1024 x 1024 = 107.374.182.400 *bajta*. To u praksi znači kako je deklarirani kapacitet diska od strane proizvođača nešto manji, zbog pretvorbe u binarni sustav.

Pogledajmo koliki je stvarno dostupni kapacitet diskova, zbog pretvorbe dekadski u binarno, u tablici dolje:

| **Dekadski sustav (oznaka proizvođača)** | **Binarni sustav (kako to vidi računalo)** |
|---|---|
| 100 GB | 93,13 GiB |
| 500 GB | 465,66 GiB |
| 1 TB | 931,32 GiB |
| 2 TB | 1.862,65 GiB |
| 3 TB | 2.794 GiB |
| 4 TB | 3.725 GiB |

Oznaka za binarno preračunati kapacitet je malo slovo **i** koje se dodaje nakon **metričkog prefiksa** te tako dobivam **binarne prefikse**. Prema *binarnim prefiksima* **GB** postaje **GiB** (*gibibajt*), **TB** postaje **TiB** (*tebibajt*) i tako dalje.

Pogledajmo i odnos između metričkih i binarnih prefiksa u sljedećoj tablici:

| Metrički dekadski prefiks | Vrijednost | Binarni prefiks | Vrijednost |
|---|---|---|---|
| **k** (kilo) | 1.000 | **ki** (kibi) | 1.024 |
| **M** (mega) | 1.000.000 | **Mi** (mebi) | 1.048.576 |
| **G** (giga) | 1.000.000.000 | **Gi** (gibi) | 1.073.741.824 |
| **T** (tera) | 1.000.000.000.000 | **Ti** (tebi) | 1.099.511.627.776 |
| **P** (peta) | 1.000.000.000.000.000 | **Pi** (pebi) | 1.125.899.906.842.624 |

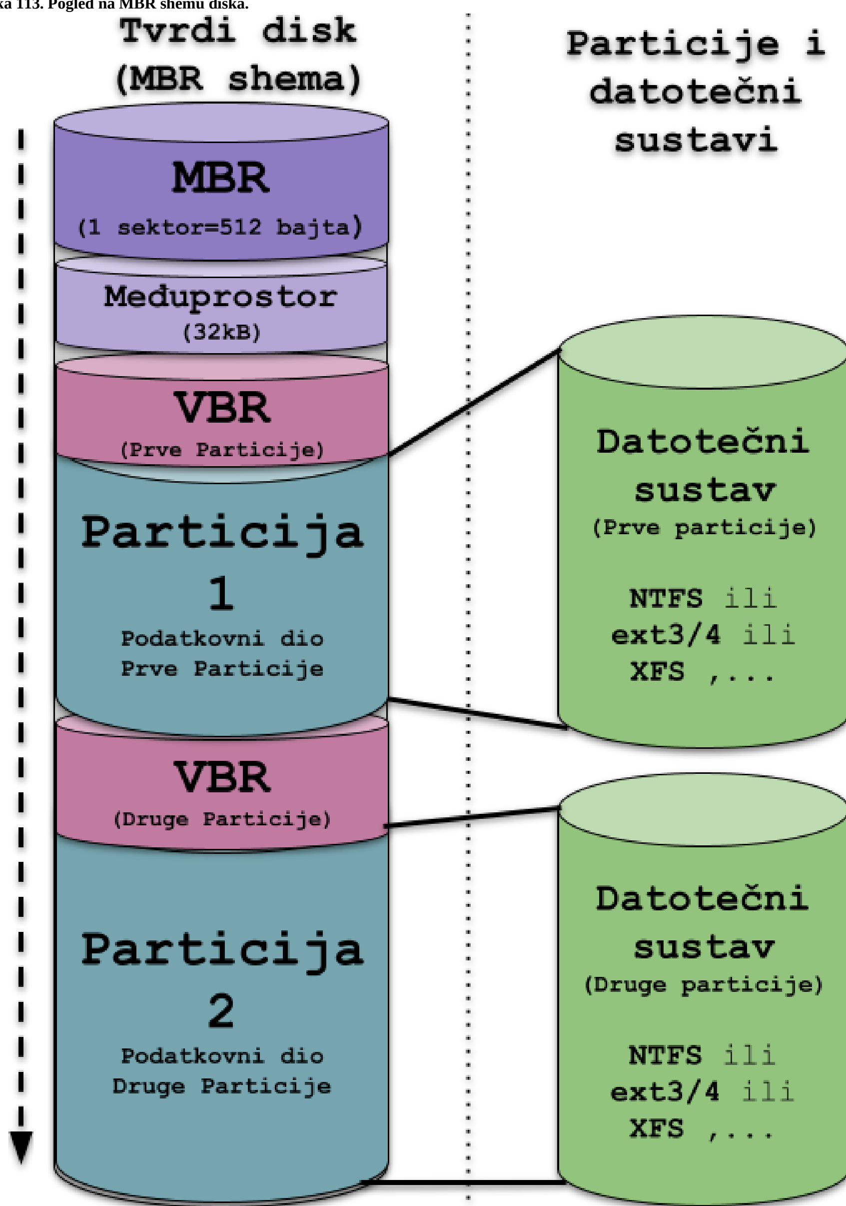
Dodatno, kada se disk *particionira* te *formatira*, gubi se još malo kapaciteta uglavnom ovisno o datotečnom sustavu.
U konačnici sâma iskoristivost kapaciteta pohranjenih podataka ovisi i o veličini *klastera*, broju i veličini datoteka i slično.

Bez obzira imamo li jedan disk ili se radi o RAID polju diskova (koje je ionako vidljivo kao jedan disk), procedura je ista:

1. ***Particioniranje*** diska na particije.
2. ***Formatiranje*** tih particija sa željenim datotečnim sustavom; primjerice sa: ***NTFS***, ***ext2***,***ext3***,***ext4***, ***XFS***, ***ZFS***, … .

Kako izgleda ***MBR*** shema diska, ako gledamo logičku površinu jednog diska s dvije particije, pogledajmo na slici 113.

**Slika 113. Pogled na MBR shemu diska.**




Kako je vidljivo na slici, svaki disk na prvom sektoru odnosno na početku diska, ima *Master Boot Record* (**MBR**) zapis u kojemu se nalazi popis svih particija i njihove lokacije, odnosno tu se nalazi ***particijska tablica*** te takozvani ***boot loader***. Unutar *boot loadera* nalazi se tzv. ***boot kôd*** za pokretanje prve faze *boot loadera*; pr. *GRUB2* (***GRUB2 stage 1***).

Između **MBR** i **VBR** nalazi se prazan prostor od 32kB u koji se često instalira međufaza *boot loadera*. Za ***GRUB2*** se tu instalira ***stage 1.5 faza***.

Ako gledamo svaku particiju zasebno, svaka particija na svom početku ima *Boot Record* (**BR**) koji se nekada naziva i *Volume Boot Record* (**VBR**) zapis, u kojem se osim ostalih polja (vrsta medija, naziv particije, vrsta datotečnog sustava i sl.) ponekad nalazi i druga faza ***boot loadera*** odnosno tzv. ***second stage loader***, za učitavanje operativnog sustava, ako ga imamo na toj particiji. To nije slučaj ako se koristi ***GRUB2***.

Ovakva struktura se često koristi i kada imamo potrebu za instalacijom dva (ili više) operativnih sustava, svakoga na svojoj particiji.

Particija s koje se pokreće operativni sustav mora biti označena kao ***boot (bootabilna)***, što znači kako ima postavljen tzv. „*boot flag*“ odnosno zastavicu za pokretanje operativnog sustava. U slučaju upotrebe *GRUB2 boot loadera*, prva faza *boot loadera* se nalazi u **MBR** zapisu, njegova međufaza (tzv. *faza 1.5*.) se nalazi pohranjena u prostoru između **MBR** i **VBR**, a druga faza tj. *second stage boot loader* za operativni sustav se nalazi pohranjena unutar `/boot/grub2/` direktorija na datotečnom sustavu prve particije.

U slučajevima kada nemamo potrebu za instaliranjem operativnog sustava na određenu *particiju* to jest kod primjena u kojima želimo razgraničiti i razdijeliti disk na više particija na kojima ćemo imati različite podatke.
Kao primjerice kada na prvoj particiji želimo imati operativni sustav, a na drugoj sâmo podatke i/ili programe.
Ili će na obje particije biti podaci, ali za različite namjene. U svakom slučaju, nakon izrade prve *particije* diska, kreiraju se dva dodatna segmenta diska nakon **MBR** zapisa i međuprostora, a to su: **VBR** dio koji opisuje sâmu *particiju* te (prazna) *particija*.
Praznu *particiju* potom moramo formatirati sa željenim datotečnim sustavom: ***NTFS***, ***ext2/3/4, XFS, ZFS***, …., koji će kasnije sadržavati sve naše podatke odnosno strukturu direktorija i datoteka.



Vezano za **boot loader** (konkretno **GRUB2**) pogledajte poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**
**11.1.1.5.1. Rad s GRUB-om prilikom pokretanja sustava (*rescue*).**

Što se tiče detalja oko procesa pokretanja računala i rada **boot loadera**, pogledajte i poglavlje:
**12.1. Što se događa kod pokretanja računala (*boot process*).**
**12.1.2. Boot proces detaljnije.**

**Vratimo se na MBR**.
Detaljnije, *Master Boot Record* (**MBR**) sadrži informacije o particijama i njihovim pozicijama: na kojoj lokaciji počinje i završava koja particija. Osim toga u *MBR* se nalazi i prva faza *boot loadera* (tzv. ***first-stage boot loader***) koji se učitava i pokreće tijekom pokretanja sustava (nakon inicijalizacije i pokretanja BIOS-a).
Unutar **MBR** zapisa se nalazi *boot loader* (pr. **GRUB2**) koji pomoću particijske tablice može doći do pozicije **VBR** zapisa unutar kojeg može biti zapisana druga faza *boot loadera*. Međutim to nije slučaj kod **GRUB2**, ali jest kod nekih drugih *boot loadera*.
Druga opcija jest da prva faza *boot loadera* iz **MBR**, može doći do međuprostora između **MBR** i **VBR** na kojem se kod *GRUB2* nalazi međufaza *boot loadera* (tzv. faza 1.5) koja može direktno pristupiti datotečnom sustavu na particiji i nastaviti s izvršavanjem *faze 2 GRUB2 boot loadera* iz direktorija `/boot/grub2/`.
Sve druge particije također počinju sa svojim **VBR** zapisom u kojem se nalazi definicija *particije*, iza kojeg slijedi sâma *particija*.

**Slika 114. Pogled na MBR shemu particioniranja tvrdog diska.**

**Tvrdi disk (MBR shema) i GRUB2 boot loader**

| MBR | Međuprostor | VBR (BR) prve particije | Prva particija | VBR (BR) druge particije | Druga particija |
|---|---|---|---|---|---|
| BOOT kôd (GRUB2 faza 1) | BOOT kôd (GRUB2 faza 1.5) | BOOT kôd (GRUB2 ga ne koristi) | Datotečni sustav (pr. NTFS, EXT4, XFS ,...) | BOOT kôd (GRUB2 ga ne koristi) | Datotečni sustav (pr. NTFS, EXT4, XFS ,...) |
| Particijska tablica | | Naziv particije<br>Vrsta medija<br>Vrsta datotečnog sustava<br>... | Stablo direktorija<br>/<br>/boot/grub2/ ← GRUB2 faza 2<br>/bin<br>... | Naziv particije<br>Vrsta medija<br>Vrsta datotečnog sustava<br>... | Stablo direktorija<br>/<br>/bin<br>/home<br>... |

**Ograničenja MBR-a**
Korištenjem **MBR** sheme particioniranja diska, ograničeno je na diskove čija je maksimalna veličina particije do 2TB.
To znači da tvrdi diskovi s većim particijama od 2 TB ne mogu koristiti **MBR** shemu particioniranja.
Stoga se u novije vrijeme više ne koristi ***MBR*** shema particioniranja, nego novija *GUID Partition Table* (***GPT***) shema koja nadilazi ovo ograničenje. Mi ćemo se u daljnjem tekstu bazirati na **MBR** shemi, zbog lakšeg razumijevanja ove materije.

***Podsjetimo se za MBR: aktivna može biti samo jedna od maksimalno četiri (4) primarne particije na disku.***

**Što kada želimo imati "*Multiboot*" sustav odnosno dva operativna sustava (OS-a) instalirana na jednom disku?**
U slučajevima kada želimo instalirati dva ili više operativnih sustava, te imati mogućnost odabira koji operativni sustav želimo pokretati; npr. instalirali smo dva ili više Linuxa ili Linux i Windows OS ili neki drugi *OS*. U ovakvom scenariju potreban nam je također *boot loader* koji je instaliran u ***MBR***, ali koji ima mogućnost pokretanja operativnog sustava sa željene particije, na kojoj se ili unutar **BR** zapisa nalazi *boot loader* druge razine odnosno onaj od sâmog operativnog sustava koji će potom i učitati *OS* ili možemo imati scenarij s *GRUB2* i međufazom (faza 1.5.) u zapisu na disku između **MBR** i **VBR**.

**Primjer kada imamo instalirane dvije distribucije Linuxa, svaku na svojoj particiji.**
Prvo smo instalirali *Linux Centos* Linux (particija **1**) i pri tome smo *boot loader* instalirali na "*root*" particiju tj. na *Boot Record* od particije **1**. (u primjeru je to `/dev/sdb1`). Nakon toga smo instalirali *Slackware Linux* na drugu particiju diska (u primjeru je to particija `/dev/sdb2`), ali smo na kraju instalacije rekli da želimo *boot loader* instalirati na ***MBR*** od cijelog diska (`/dev/sdb`). Tako će *boot loader* instaliran u ***MBR*** biti zadužen za prvu fazu pokretanja računala. Osim toga, on će morati biti svjestan i drugog Linuxa i njegovog *boot loadera* druge razine, koji je instaliran unutar druge particije diska (konkretno kod nas je to particija `/dev/sdb1`). Dakle imamo particiju **1**: ***Linux CentOS*** i particiju **2**: **Linux Slackware.**

Ove particije, gledane to jest pokrenute iz programa `cfdisk` izgledaju ovako (*o ovoj naredbi nešto kasnije*):
**cfdisk /dev/sdb**
```
                              Disk Drive: /dev/sdb
                        Size: 1073741824 bytes, 1073 MB
              Heads: 139   Sectors per Track: 8   Cylinders: 1885
    Name         Flags       Part Type  FS Type          [Label]        Size (MB)
 ------------------------------------------------------------------------------
    sdb1         Boot         Primary   Linux ext3         CentOS         511.85
    sdb2         Boot         Primary   Linux ext3         Slackware      561.91*
```
U navedenom primjeru s dva operativna sustava, imamo dvije particije od kojih je svaka označena kao `Boot` particija (vidljivo pod stupcem `Flags`) i sada nije bitno koja je od njih označena kao aktivna jer će se *Boot Loader* na **MBR** brinuti o tome s koje particije će pokretati koji operativni sustav. Odnosno bit će nam ponuđen odabir željenog operativnog sustava.
Pokretanjem računala, dobiti ćemo izbornik iz **MBR** *boot loadera* koji će nam nuditi mogućnost pokretanja operativnog sustava, s opcijama (u ovom slučaju):

- **1** (***CentOS Linux***).
- **2** (***Slackware Linux***).

Odnosno izbornik će imat iizbor između particije **1** (`/dev/sdb1`) ili particije **2** (`/dev/sdb2`) s kojih će se željeni operativni sustav moći pokrenuti tijekom pokretanja (uključivanja) računala.

← Konkretan slučaj s **CentOS** i **Slackware** Linuxom je malo kompleksniji jer *CentOS* koristi ***GRUB2*** dok *Slackware* Linux koristi **LILO (*LInux LOader*)** *boot loader*, koji rade malo drugačije, ali sada nećemo komplicirati.

Vratimo se na logički izgled diska. Prijašnja slika 114. prikazuje osnovni izgled diska s dvije primarne particije.



***Ne zaboravimo da originalna MBR shema particioniranja PC računala, dozvoljava sâmo četiri primarne particije!***

Pošto je to s vremenom postalo premalo, uvedene su takozvane ***extended*** odnosno proširene ili logičke particije.
Uvođenjem proširenih particija zapravo je omogućeno unutar svake od primarnih particija kreiranje do 15 pôd particija odnosno logičkih particija. U daljem tekstu ćemo se zbog jednostavnosti držati primarnih particija.
Za početak ćemo se bazirati na osnovnim Unix/Linux datotečnim sustavima, koji se nalaze na primarnoj particiji.



***Unix/Linux sustavi i diskove i particije vide prvo kao posebne datoteke (u*** `/dev/` ***direktoriju), a koje se nakon formatiranja povezuju (montiraju/mountaju) u neki postojeći direktorij.***

***Za razliku od Windows OS-a koji formatirane particije odmah povezuje sa slovima koja označavaju diskove, poput:*** `C`, `D`, `E`, ***…***

***Aktivna particija**** je ona koja sadrži operativni sustav koji računalo pokušava učitati tijekom pokretanja sustava.

**Napomena za tvrde diskove i njihove oznake pod Linuxom:**

- `/dev/sda` označava prvi SATA ili SAS ili SCSI tvrdi disk, a pri tome:
  - `/dev/sda1` označava prvu particiju prvog SATA tvrdog diska.
  - `/dev/sda2` označava drugu particiju prvog SATA tvrdog diska.
- `/dev/sdb` označava drugi SATA ili SAS ili SCSI tvrdi disk, a pri tome:
  - `/dev/sdb1` označava prvu particiju drugog SATA tvrdog diska.
  - `/dev/sdb2` označava drugu particiju drugog SATA tvrdog diska.
- `/dev/sdc` označava treći SATA ili SAS ili SCSI tvrdi disk…

Pogledajmo primjer kreiranja jedne particije na trećem SATA disku (`/dev/sdc`) koji smo spojili na računalo, a disk je veličine 2GB.



Za primjere i detalje oko particioniranja diskova pogledajte poglavlje:
**13.6.1. Kreiranje particija.**

Pokrenut ćemo program za particioniranje diskova: `cfdisk`, za željeni disk, na sljedeći način:

```
cfdisk /dev/sdc
                         Disk Drive: /dev/sdc
                   Size: 2147483648 bytes, 2147 MB
         Heads: 255    Sectors per Track: 63   Cylinders: 261

   Name        Flags     Part Type FS Type         [Label]      Size (MB)
 --------------------------------------------------------------------
                        Pri/Log  Free Space                    2147.4*

[   Help    ]  [   New     ]  [  Print    ]  [   Quit    ]  [  Units    ]  [  Write    ]
```

Odaberimo (s kursorskim tipkama ←↑↓→): `[New]` te stisnimo *ENTER*. Sada pošto želimo primarnu particiju, odabrat ćemo `[Primary]` te potvrditi s *ENTER*.

Sada će nas program za particioniranje pitati za veličinu prve particije. Pošto od cijelog diska od 2GB želimo napraviti samo jednu particiju, odabrat ćemo cijeli (ukupni) kapacitet diska za particiju:

```
Size (in MB): 2147.48
```

Potvrdimo to s ***ENTER***.

I sada smo dobili predloženo kreiranje prve particije (`/dev/sdc1`) na našem disku (`/dev/sdc`):

```
                          Disk Drive: /dev/sdc
                    Size: 2147483648 bytes, 2147 MB
         Heads: 255   Sectors per Track: 63   Cylinders: 261

   Name        Flags     Part Type FS Type         [Label]       Size (MB)
 --------------------------------------------------------------------
 sdc1                    Primary   Linux                          2147.4*

[Bootable] [Delete] [Help] [Maximize] [Print] [Quit]  [Type]  [Units] [Write]
```

Potom je potrebno to sve i potvrditi odnosno stvarno i snimiti na disk.

To ćemo napraviti tako da odaberemo: `[Write]` te potvrdimo s *ENTER*, a potom dodatno potvrdimo, tako što ćemo upisati: `yes` i ponovno potvrditi s *ENTER*.

Tek sada je particija kreirana i pripremljena da se može formatirati s datotečnim sustavom koji je standardno označen pod stupcem: `FS Type`. U našem slučaju je to `Linux` što podrazumijeva: ***ext2, ext3*** ili ***ext4*** datotečne sustave.

Sada možemo izaći iz programa `cfdisk` odabirom: `[Quit]` te *ENTER*. I tek sada možemo formatirati ovu particiju s datotečnim sustavom. Mi ćemo se odlučiti za ***ext4*** datotečni sustav.
Pogledajmo kako to sada izgleda, iz Linux *bash* ljuske, pomoću naredbe `lsblk`:

```
lsblk /dev/sdc

NAME    MAJ:MIN RM SIZE RO  TYPE  MOUNTPOINT
sdc       8:16   0   2G  0  disk
└─sdc1    8:17   0   2G  0  part
```

Dakle vidimo da disk `/dev/sdc` ima samo jednu odnosno prvu particiju: `/dev/sdc1`.

Ovu particiju ćemo formatirati s naredbom `mkfs.ext4` sa ***ext4*** datotečnim sustavom, na sljedeći način:

```
mkfs.ext4 /dev/sdc1
```

Uslijedit će poruke o formatiranju particije.
Kada sve završi možemo ponovno sve provjeriti s naredbom `lsblk` ovako:

```
lsblk -f /dev/sdc

NAME    FSTYPE LABEL UUID                                 MOUNTPOINT
sdc
└─sdc1  ext4         0cf6f404-9705-48e3-9c96-a40bf216d0bf
```

Vidimo kako je `FSTYPE` to jest vrsta datotečnog sustava upravo `ext4` s kojim smo i formatirali particiju `/dev/sdc1`



U ispisu je vidljiva **UUID** oznaka particije koja je njen jedinstveni identifikator, a za detalje o ovoj oznaci i čemu ona služi, proučite poglavlje:
**13.6.2.3. Universally unique identifier (UUID) oznake.**



*Još nam ostaje i zadnji korak, a to je montiranje (**mountanje**) particije u direktorij (mapu,) koje ćemo objasniti nešto kasnije. O cijeloj ovoj proceduri nešto detaljnije u jednom od sljedećih poglavlja.*



Za primjere i razloge montiranja formatiranih particija, proučite poglavlje:
**13.7. Kako se montiraju particije u direktorije i zašto.**

**Izvori informacija: (312),(313),(314),(315),(666),(667), `man fdisk`,`man cfdisk`,`man gdisk`,`man cgdisk`, `man parted`,`man mkfs`, `man lsblk`,`man 7 boot`.**

## 13.4.1. Napredno: MBR Detaljnije

***Slijedi napredno poglavlje!***
Prema **MBR** shemi particioniranja diska, prvih 512 bajta na svakom disku sadrži takozvani *Master Boot Record* (***MBR***) zapis u kojem se nalazi ***boot loader***, koji sadrži i particijsku tablicu s popisanim svim particijama koje se nalaze na ostatku diska.

Čisto za probu, pomoću naredbe `dd` prekopirat ćemo **MBR** zapis u datoteku na disku.
Nemojte pobrkati oznaku diska (recimo `/dev/sda`) i particiju; primjerice prvu `/dev/sda1` i drugu particija na istom disku: `/dev/sda2`. Dakle oznaka diska nema redni broj, kao što ga ima particija.
Dakle kopiramo prvih **512** bajta s našeg diska `/dev/sda` u datoteku: `/root/MBR.bin`, na sljedeći način:

```
dd if=/dev/sda of=/root/MBR.bin bs=512 count=1
```

U ovom procesu smo dobili datoteku imena: `MBR.bin` unutar `/root/` direktorija. Popis particija iz MBR-a možemo vidjeti s naredbom `file` koja prepoznaje sadržaj datoteka i daje nam informacije o njima (*ispis ovisi o inačici naredbe* `file`).

Pogledajmo ovu našu datoteku s naredbom `file` na sljedeći način:

```
file /root/MBR.bin
/root/MBR.bin: x86 boot sector;
partition 1: ID=0x83, active, starthead 1, startsector 16, 2000048 sectors;
partition 2: ID=0x82, starthead 21, startsector 2000064, 2194240 sectors, code offset 0x0
```

Istu, ali detaljniju statistiku možemo vidjeti i s naredbama `fdisk` ili `gdisk`, ovako:

```
fdisk -l /root/mbr.bin
```

Odnosno isto možemo postići i s naredbom:

```
gdisk -l /root/mbr.bin
```

U ispisu gornje naredbe vidimo sljedeće informacije o datoteci (`MBR.bin`) koja je zapravo ***MBR*** zapis (*boot loader*):

- Particija **1** (`partition 1`) – ovdje su podaci o prvoj particiji (ne zaboravimo da je ovo zapravo *MBR* zapis):
  - Vrsta particije je **`0x83`** što označava Linux vrstu particije.
  - Ona je aktivna (`active`) što znači kako se s nje može pokretati (*boot-ati*) sustav.
  - Vidimo i na kojem sektoru počinje (`startsector`) i na kojemu završava (`sectors`) particija.
- Particija **2** (`partition 2`) – ovdje su podaci o drugoj particiji (ne zaboravimo da je ovo zapravo *MBR* zapis):
  - Vrsta particije je **`0x82`** što označava Linux **swap** vrstu particije.
  - Vidimo i na kojem sektoru počinje (`startsector`) i na kojemu završava (`sectors`) particija.

Sve podržane vrste* particija pod Linuxom možete vidjeti u tablici dolje.



***Vrsta* particije se mijenja u trenutku kreiranja particije, prije njenog formatiranja!***

| Oznaka (ID) | Opis | Oznaka (ID) | Opis | Oznaka (ID) | Opis | Oznaka (ID) | Opis |
|---|---|---|---|---|---|---|---|
| 01 | FAT12 | 39 | Plan 9 | 83 | Linux | C6 | DRDOS/sec (FAT-16) |
| 02 | XENIX root | 3C | PartitionMagic recov | 84 | OS/2 hidden C: drive | C7 | Syrinx |
| 03 | XENIX usr | 40 | Venix 80286 | 85 | Linux extended | DA | Non-FS data |
| 04 | FAT16 -32M | 41 | PPC PReP Boot | 86 | NTFS volume set | DB | CP/M / CTOS / … |
| 05 | Extended | 42 | SFS | 87 | NTFS volume set | DE | Dell Utility |
| 06 | FAT16 | 4D | QNX4.x | 88 | Linux plaintext | DF | BootIt |
| 07 | HPFS/NTFS | 4E | QNX4.x 2nd part | 8E | Linux LVM | E1 | DOS access |
| 08 | AIX | 4F | QNX4.x 3rd part | 93 | Amoeba | E3 | DOS R/O |
| 09 | AIX bootable | 50 | OnTrack DM | 94 | Amoeba BBT | E4 | SpeedStor |
| 0A | OS/2 Boot Manager | 51 | OnTrack DM6 Aux1 | 9F | BSD/OS | EB | BeOS FS |
| 0B | W95 FAT32 | 52 | CP/M | A0 | IBM Thinkpad Hib. | EE | GPT |
| 0C | W95 FAT32 (LBA) | 53 | OnTrack DM6 | A5 | FreeBSD | EF | EFI (FAT-12/16/32) |
| 0E | W95 FAT16 (LBA) | 54 | OnTrackDM6 | A6 | OpenBSD | F0 | Linux/PA-RISC boot |
| 0F | W95 Extd (LBA) | 55 | EZ-Drive | A7 | NeXTSTEP | F1 | SpeedStor |

| Oznaka (ID) | Opis | Oznaka (ID) | Opis | Oznaka (ID) | Opis | Oznaka (ID) | Opis |
|---|---|---|---|---|---|---|---|
| 10 | OPUS | 56 | Golden Bow | A8 | Darwin UFS | F4 | SpeedStor |
| 11 | Hidden FAT12 | 5C | Priam Edisk | A9 | NetBSD | F2 | DOS secondary |
| 12 | Compaq diagnostics | 61 | SpeedStor | AB | Darwin boot | FB | VMware VMFS |
| 14 | Hidden FAT16 <32M | 63 | GNU HURD or SysV | AF | HFS / HFS+ | FC | VMware VMKCORE |
| 16 | Hidden FAT16 | 64 | Novell Netware 286 | B7 | BSDI fs | FD | Linux raid autodetect |
| 17 | Hidden HPFS/NTFS | 65 | Novell Netware 386 | B8 | BSDI swap | FE | LANstep |
| 18 | AST SmartSleep | 70 | DiskSecure Multi-Boot | BB | Boot Wizard hidden | FF | BBT |
| 1B | Hidden W95 FAT32 | 75 | PC/IX | BE | Solaris boot | | |
| 1C | Hidden W95 FAT32 (LBA) | 80 | Old Minix | BF | Solaris | | |
| 1E | Hidden W95 FAT16 (LBA) | 81 | Minix / old Linux | C1 | DRDOS (FAT-12) | | |
| 24 | NEC DOS | 82 | Linux swap / Solaris | C4 | DRDOS (FAT-16) | | |



Vezano za detalje oko **MBR** sheme particioniranja te **MBR** i **VBR** zapise pogledajte poglavlje:
**12.1.2. Boot proces detaljnije.**

Vratimo se na detalje ***MBR*** zapisa odnosno pogledajmo od čega se on sastoji.
Naime ovdje postoje: tradicionalna i novija vrsta MBR zapisa (koja se najčešće i koristi).

| Tradicionalni MBR | | Novija vrsta MBR | |
|---|---|---|---|
| **Opis** | **Veličina (bajta)** | **Opis** | **Veličina (bajta)** |
| **Boot kod** (*bootstrap*) | **446** | **Boot kôd** (*bootstrap*) **Prvi dio** | **21** |
| | | **Vremenska oznaka** (*timestamp*) | **6** |
| | | **Boot kôd** (*bootstrap*) **Drugi dio** | **216** |
| | | **Identifikacijska oznaka diska** | **4** |
| | | **Dodatna oznaka** (*često je prazna, ali ovisi o OS-u*) | **2** |
| **Particijski unos particije br.1** (*PTE*) | **16** | **Particijski unos particije br.1** (*PTE*) | **16** |
| **Particijski unos particije br.2** (*PTE*) | **16** | **Particijski unos particije br.2** (*PTE*) | **16** |
| **Particijski unos particije br.3** (*PTE*) | **16** | **Particijski unos particije br.3** (*PTE*) | **16** |
| **Particijski unos particije br.4** (*PTE*) | **16** | **Particijski unos particije br.4** (*PTE*) | **16** |
| **MBR potpis** (*Boot signature*) | **2** | **MBR potpis** (*Boot signature*) | **2** |
| **Ukupno** | **512** | **Ukupno** | **512** |

Sada pogledajmo samo osnovne elemente zapisa zaduženog za svaku pojedinu particiju, zvanog *Partition Table Entries (**PTE**)*:

- **Prvi i zadnji sektor** – definira prvi i zadnji sektor particije (početak i kraj).
- **Broj sektora** – definira veličinu particije u broju sektora.
- **Tip particije** – označava tip datotečnog sustava koji će biti instaliran na particiju (tijekom formatiranja), kako je vidljivo u tablici s prethodne stranice (primjerice: `82`-Linux Swap, `83`-Linux, …).
- **Status particije** – označava status particije, koji može biti: `0x80`-aktivna (*boot*) ili `0x00`-neaktivna.

Dio zapisa s identifikacijskom oznakom diska odnosi na cijeli tvrdi disk (ne na particiju). Ovaj identifikator diska je broj koji se nasumično generira kada se prvi puta kreira i pohranjuje *MBR* zapis. Ova oznaka diska nije **UUID** oznaka koja se kreira tijekom formatiranja diska. Ovaj identifikator može se vidjeti (pr. za disk `/dev/sda`) s naredbom `gdisk`:

```
gdisk -l /dev/sda |  grep "Disk identifier"
Disk identifier (GUID): 74A4F0A5-A5F8-403B-AA52-0F7BBC553021
```

**Izvori informacija: (666),(667),(668),(1637), `man dd`,`man gdisk`,`man 7 boot`, `man file` te poglavlje u knjizi: 12.1.**

# 13.5. Blok uređaji (*block device*)

Prije svega, moramo objasniti kako se uopće zapisuju podaci na neki disk uređaj odnosno medij. Disk uređaji kao što su: tvrdi disk, *SSD* disk i drugi, uređaji su na koje se podaci zapisuju i iz kojih se čitaju u blokovima podataka. Pri tome su ti blokovi iste veličine. Ovakve uređaje stoga zovemo blok uređajima (engl. *block device*) upravo zbog načina na koji se podaci na njih zapisuju i s kojih se čitaju. Blok uređaj zapravo predstavlja *virtualni* uređaj za zapisivanje podataka u blokovima odnosno većoj, ali fiksnoj količini podataka odjednom, pri čemu je moguć pristup bilo kojem pojedinačnom bloku podataka, neovisno o pristupanju bilo kojem drugom bloku podataka. Uređaji koji koriste blok metodu pristupa su:

- Tvrdi diskovi: USB disk, mehanički odnosno *rotacijski* disk, *SSD* ili *NVMe* disk.
- Određeni mrežni protokoli za dijeljenje diskova preko mreže (***SAN*** sustavi), poput: *iSCSI*, *DRBD*, …
- CD/DVD optički i drugi.

Sada pogledajmo naš prvi SATA tvrdi disk, koji u Linuxu predstavlja posebna datoteka: `/dev/sda`

```
ls -al /dev/sda

brw-rw---- 1 root disk 8, 0 Apr 8 15:57 /dev/sda
```

Prvo slovo u ovlastima (`b`) ove datoteke koja predstavlja naš cijeli tvrdi disk je `b` što znači kako se radi o blok uređaju. Dakle u radu s diskovima i particijama se podrazumijeva da radimo s blok uređajima. Pogledajmo ovaj disk s naredbom `lsblk`:

```
lsblk

NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   12G  0 disk
```

Vidimo da disk `/dev/sda` nema particije te da je veličine 12GB. Osim toga vidimo njegove *Major* (`8`) i *Minor* (`0`) oznake.



Ostale vrste datoteka (i direktorija) možete vidjeti u poglavlju:
**4.2 Direktoriji (mape) i datoteke.**
Pogledajte i poglavlja o diskovnom sustavu, uređajima i načinu njihove detekcije i aktivacije te o naredbi **lsblk**:
**11.1.2. Uređaji (*devices*) ukratko.**
**13.10.1. Naredba *lsblk*.**
**14. Diskovni (I/O) podsustav.**

**Izvori informacija: (K-5), `man lsblk` te poglavlje u knjizi: 13.10.1.**

## 13.5.1. Rad s blok uređajima i naredba dd

Naredba odnosno program `dd` je uslužni program naredbenog retka inicijalno razvijen za Unix, tijekom 1974. godine, a s vremenom se počeo razvijati i za druge operativne sustave poput *FreeBSD* Unixa, *MacOS*-a i Linuxa.
Njegova je primarna svrha konverzija i kopiranje datoteka. Moramo biti svjesni da su na Unixu (pa i Linuxu), upravljački programi uređaja odnosno hardvera (kao što su primjerice tvrdi diskovi, SSD, USB i drugi diskovi), kao i posebne datoteke (kao što su `/dev/zero` i `/dev/random`) zapravo vidljivi u datotečnom sustavu kao datoteke.
Stoga naredba `dd` također može čitati i/ili pisati iz/u ove datoteke, pod uvjetom da je ta funkcija implementirana u njihovom odgovarajućem upravljačkom programu. Kao rezultat toga, `dd` se može koristiti za zadatke kao što je izrada sigurnosne kopija *MBR* zapisa na disku, kopiranja diskova ili particija i slično. Program `dd` također može izvršiti pretvorbe na podacima dok se kopiraju, uključujući zamjenu redoslijeda bajtova ili pretvorbu u i iz ASCII i EBCDIC i slično.

**Blokovi**
Blok je jedinica mjere broja bajtova koji se čitaju, pišu ili pretvaraju odjednom. Srećom opcije naredbenog retka mogu odrediti i drugačiju veličinu bloka za unos/čitanje (`ibs`) u usporedbi s izlazom/pisanjem (`obs`), iako će opcija veličine bloka (`bs`) nadjačati i `ibs` i `obs`. Zadana vrijednost za ulaznu i izlaznu veličinu bloka je 512 bajtova: zbog tradicionalne veličina bloka kod diskova i POSIX-om propisane veličina "bloka".
Postoji i opcija brojanja (`count`) za broj kopiranja, kao i opcija za broj preskakanja za čitanje (`skip`) i broj traženja za pisanje (`seek`). Na operacije pretvorbe također utječe i takozvana veličina bloka pretvorbe (`cbs`).

Za sve navedene operande moguće je definirati i sufikse prema **SI** ili binarnom prefiksu, poput:

- **kB** – x 1000
- **K** – x 1024
- **MB** – 1000 x 1000
- **M** - 1024 x 1024
- **GB** - 1000 x 1000 x 1000
- **G** – 1024 x 1024 x 1024, i tako dalje.

**Slijede primjeri**
**Kopiranje diskova i particija**
**1.** Pogledajmo kako kopirati (klonirati) disk `/dev/sda` u disk `/dev/sdb`:

```
dd if=/dev/sda of=/dev/sdb bs=16k conv=noerror
```

← Opcija `bs=16k` je definirala da je veličina bloka 16 kB što je optimalna vrijednost za mehaničke diskove koji imaju velićinu sektora od 4kB, pa je veličina bloka od 4 sektora (4x4kB) dobra opcija. Dok za *SSD* diskove ta veličina može biti dosta veća; primjerice: od 4kB do 128kB ili više. Kod *NVMe* diskova ove vrijednosti mogu biti još veće: od 64kB do 256kB, a neki NVMe diskovi imaju još veće veličine blokova (od 1MB na više) pa je kod njih dobro povećavati vrijednost od 1MB na više. U svakom slučaju dobro je prvo napraviti inicijalne testove povećanja ove vrijednosti do one veličine s kojom dobivamo najveću brzinu prijenosa podataka.
Opcija `conv=noerror` znači da iako se slučajno naiđe na greške pri čitanju s određenih blokova diska, da se nastavi dalje.

**2.** Kopiranje (kloniranje) druge particije `/dev/sda2` u drugu particiju na drugom disku `/dev/sdb2`:

```
dd if=/dev/sda2 of=/dev/sdb2 bs=16k conv=noerror
```

← Opcije su iste kao i iz prethodnog primjera.

**3.** Kopiranje (kloniranje) cijelog diska `/dev/sda` u diskovnu sliku (datoteku na dugom disku, u direktoriju `/BKP/`):

```
dd if=/dev/sda of=/BKP/disk-sda-bkp.img bs=16k conv=noerror
```

← Opcije su također iste kao i prethodne. Važno je da ako je `/dev/sda` sistemski disk, da je `/BKP` montirani direktorij s drugog diska.

## Proces kopiranja podataka

Naredba `dd` prvo čita podatke s izvornog diska u međuspremnik operativnog sustava (*RAM*).
Ovaj međuspremnik je privremeno spremište koje operativni sustav koristi za brzi pristup često korištenim podacima. Međutim, kada koristimo `conv=sync`, podaci se ne zadržavaju u međuspremniku te se nastavlja proces čitanja (*sinkroni* rad) i direktonog zapisivanja na odredišni uređaj (disk). Tada, kada koristimo `conv=sync`, `dd` tada šalje poseban zahtjev operativnom sustavu koji zahtijeva da se podaci iz međuspremnika odmah zapišu na odredišni disk, što negativno utječe na performanse, ali u slučaju primjerice nestanka napajanja osiguravaju pouzdano kopiranje podataka (jer se praktično ne koristi međumemorija). Negativna strana upotrebe `conv=sync` je prilično sporiji *sinkroni* način kopiranja podataka.

**4. Kopiranje i brisanje sektora na disku**
Moguće je i primjerice kopirati prvi sektor na disku odnosno takozvani **MBR** zapis, u datoteku udirektoriju `/DATA/`
Mi ćemo to napraviti na disku `/dev/sda`:

```
dd if=/dev/sda of=/DATA/MBR.bin bs=512 count=1
```

← Ovime smo ***MBR*** zapis na disku, koji unutar sebe sadrži i particijsku tablicu cijelog diska kopirali u datoteku `/DATA/MBR.bin`.

**5.** Moguće je i obrisati primjerice prvi sektor na disku odnosno takozvani ***MBR*** zapis.
Ovime će vam disk biti nedostupan, stoga to nemojte raditi na disku na kojem su vam važni podaci.
Mi ćemo to napraviti na disku `/dev/sda`:

```
dd if=/dev/zero of=/dev/sda bs=512 count=1
```

← Ovdje s generatorom nula (`/dev/zero`) popunjavamo prvih (`count=1`) 512 bajta (`bs=512`) na disku.
Odnosno brišemo ***MBR*** zapis na disku, koji unutar sebe sadrži i particijsku tablicu cijelog diska.
**Nemojte ovo raditi!**

**6.** U ovom koraku ćemo vratiti malo prije (u koraku **4.**) pohranjeni ***MBR*** zapis, na odredišni disk `/dev/sda`:

```
dd if=/DATA/MBR.bin of=/dev/sda bs=512 count=1
```

← Ovdje smo sadržaj postojeće datoteke (`/DATA/MBR.bin`) kopirali u prvih (`count=1`) 512 bajta (`bs=512`) na disku.
Odnosno kopirali smo kopiju *MBR* zapisa iz datoteke kreirane u koraku **4.** na disk odnosno u njegov prvi sektor od 512 bajta.

**Testiranje brzine diska**
**7.** U ovom primjeru testiramo brzinu čitanja s diska (`/dev/sdb`)

```
dd if=/dev/sdb of=/dev/null bs=4k conv=sync status=progress
```

← Ovdje čitamo podatke s diska (`/dev/sdb`) a izlaz se prosljeđuje u posebnu datoteku (`/dev/null`) koja označava da se podaci nigdje ne zapisuju. Naznačili smo i metodu sinkronog rada (`conv=sync`) iako ona kod čitanja nema nekog efekta. Dodatno smo koristili i (`status=progress`) kako bismo vidjeli napredak čitanja pa ćemo vidjeti kojom brzinom se podaci čitaju s diska tijekom procesa čitanja.

---

**Kreiranje datoteka**
**8.** U ovom primjeru kreirat ćemo praznu datoteku u direktorij `/DATA`, veličine 10MB:

```
dd if=/dev/zero of=/DATA/prazno.dat bs=1M count=10
```

← Ovdje smo koristili veličinu bloka od 10MB (`bs=1M`) uz desert ponavljanja (`count=10`). Pri tome smo kao ulaz koristili psebnu datoteku (`/dev/zero`) koja je generator nula. Stoga će odredišna datoteka biti popunjena s nulama.
Sadržaj ovakve datoteke možemo vidjeti sa:

```
hexdump blankfile
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
0a00000
```

← Pošto je cijela datoteka ispunjena smo s nulama, ispis je skraćen te pre kra vidimo zvjezdicu (`*`)

**8.** Kreiramo novu datoteku, tako da tek nakon pozicije od 10MB nadodajemo još 1MB i sve zapisujemo u novu datoteku:

```
dd if=/dev/zero of=output.txt bs=1M count=1 seek=10
```

← Dakle veličina bloka je 1MB (`bs=1M`), ali skačemo na poziciju 10x1MB (`seek=10`) i tamo kreiramo jedan blok (`count=1`) od 1M. To znači da dobijamo datoteku veličine 11MB, ali koja se nije cijela popunjavala nulama od početka, već se odmah skočilo na poziciju 10MB nakon početka datoteke.



***Ovakva metoda rada omogućuje vrlo brzo kreiranje velikih datoteka.***

**9.** Kreiranje datoteke s nasumičnim podacima možemo napraviti sa:

```
dd if=/dev/urandom of=randomdata.txt bs=1M count=10
```

← Ovdje smo kao ulaz koristili posebnu datoteku koja je generator nasumičnih brojeva (`/dev/urandom`), a datoteka će biti veličine 10MB, kako smo već naučili.

**10.** Kopiranje datoteke s konverzijom svih slova u velika

```
dd if=input.txt of=output.txt conv=ucase
```

← Ovdje kopiramo sadržaj prve datoteke (`input.txt`) u izlaznu datoteku (`output.txt`), ali pri tome radimo konverziju svih slova u velika slova (`conv=ucase`).

**11.** Kombinacija s drugim naredbama
U ovom primjeru ulaz podataka će nam biti iz naredbe *dd*, ali ćemo izlaz proslijediti drugoj naredbi:

```
dd if=input.txt bs=1M | md5sum
```

← U ovom primjeru se sadržaj ulazne datoteke (`input.txt`) u blokovima od 1MB (`bs=1M`) preko pipe (`|`) šalje naredbi `md5sum` koja će napraviti *MD5* provjerni zbroj (engl. *hash/checksum*) sadržaja datoteke koji preuzima od naredbe *dd*.

**12.** Kopiranje diska (backup) u komprimiranu datoteku na drugom disku, u direktorij `/DATA/`:

```
dd if=/dev/sda | gzip > /DATA/disk_image.gz
```

← Ovdje je ulaz cijeli disk (`/dev/sda`) i sve se prosljeđuje preko pipe (`|`) i šalje naredbi za komprimiranje dolaznih podataka, a na kraju se sve zapisuje u izlaznu datoteku `/DATA/disk_image.gz`, koja mora biti na drugom disku (ako je `/dev/sda` sistemski disk).



Pogledajte i neke druge primjere upotrebe naredbe **dd**:
**4.6. Prorijeđene (*Sparse*) datoteke.**
**5.3.1. Napredno kreiranje i brisanje (sadržaja) datoteka.**
**5.7.5. Naredba *tr*.**
**8.1.1.1.1. Naredba *dd* u kombinaciji sa naredbom *gzip*.**
**9.4.1. Unix Pipes.**
**9.5. Linux poslovi (*jobs*).**
**11.1.2. Uređaji (*devices*) ukratko.**
**13.4.1. Napredno: MBR Detaljnije.**
**13.8.2. Swap particija.**

**Izvori infomacija: (1638), `man dd`**

## 13.6. Rad s particijama

Svaki disk odnosno takozvani *blok uređaj* u pravilu treba biti podijeljen na jednu ili više logičkih cjelina koje se nazivaju ***particije***. Naime datotečni sustavi poput: ***NTFS***, ***ext2/3/4***, ***XFS***, ***ZFS*** i drugih, mogu se instalirati samo na *particiju* na disku, pa prema tome, sve dok disk nije podijeljen na *particije* (minimalno jednu), na njega se ne može instalirati datotečni sustav, a samim time niti operativni sustav. U ovom poglavlju intenzivno ćemo raditi s *particijama* diskova.

### 13.6.1. Kreiranje particija

Za kreiranje particija, ako govorimo o **MBR** vrsti *particija*, najčešće imamo dostupna dva programa:

- `fdisk` – naredbeni program.
- `cfdisk` – *grafički* naredbeni program i drugi programi.

U slučaju upotrebe velikih diskova odnosno *particija*, većih od 2 TB, moramo koristiti ***GPT*** vrstu particija, a sâmim time moramo koristiti i programe koji znaju raditi s ***GPT*** particijama, a to su:

- `gdisk` – naredbeni program.
- `cgdisk` – *grafički* naredbeni program ili neki drugi programi.

Pošto su primjeri vezani za ***MBR*** vrstu *particije*, mi ćemo koristiti `cfdisk` koji će nam prikazati disk i *particije* na *vizualan* i malo razumljiviji način. U ovom primjeru koristit ćemo drugi SATA hard disk `/dev/sdb` i na njemu kreirati sâmo jednu particiju, koja će postati `/dev/sdb1`. Ali prvo pogledajmo koje sve blok uređaje imamo na sustavu, s naredbom `lsblk`:

```
lsblk

NAME   MAJ:MIN RM  SIZE RO  TYPE  MOUNTPOINT
sda      8:0    0   12G  0  disk
├─sda1   8:1    0   11G  0  part  /
└─sda2   8:2    0 1023M  0  part  [SWAP]
sdb      8:16   0    2G  0  disk
```

Vidimo kako imamo sistemski disk: `sda` (to jest `/dev/sda`) koji je podijeljen na dvije *particije*, a to su:

- Prva *particija*: `sda1` to jest `/dev/sda1` je veličine 11 GB koja je *montirana* (o tome kasnije) u vršni direktorij `/`
- Druga *particija*: `sda2` to jest `/dev/sda2` je veličine 1023 MB (~1GB) koja je ***swap*** *particija* pa nema točku montiranja (nema *mountpoint)* - o tome također kasnije.

Zatim imamo naš drugi tvrdi disk: `sdb` (datoteka `/dev/sdb`) koji još nije *particioniran* jer pod `TYPE` vidimo samo `disk`
Sada konačno pokrenimo program `cfdisk` na našem novom `/dev/sdb` disku s kojim ćemo kreirati *particiju* na njemu:

```
cfdisk /dev/sdb

                        Disk Drive: /dev/sdb
                        Size: 2147483648 bytes, 2147 MB
              Heads: 255   Sectors per Track: 63   Cylinders: 261

    Name         Flags       Part Type  FS Type           [Label]        Size (MB)
 ------------------------------------------------------------------------------
                             Pri/Log    Free Space                        2147.49*

                   [Help]  [New]  [Print]  [Quit]  [Units]  [Write]
```

Sada s kursorskim tipkama možemo odabrati: `[NEW]` te stisnimo *ENTER*. Pošto želimo kreirati primarnu *particiju*, odabrat ćemo `[Primary]` te potvrditi s *ENTER*. Sada će nas program za particioniranje pitati za veličinu prve *particije*. Zbog toga što od cijelog diska od 2GB želimo napraviti sâmo jednu *particiju*, odabrat ćemo cijeli (ukupni) kapacitet diska za *particiju*:

```
Size (in MB): 2147.48
```

Potvrdimo s *ENTER*. I sada smo dobili predloženo kreiranje prve *particije* (`/dev/sdb1`) na našem disku (`/dev/sdb`):

```
                         Disk Drive: /dev/sdb
                   Size: 2147483648 bytes, 2147 MB
         Heads: 255   Sectors per Track: 63   Cylinders: 261

  Name        Flags     Part Type FS Type          [Label]      Size (MB)
 ---------------------------------------------------------------------
  sdb1                  Primary   Linux                          2147.4*

       [Bootable] [Delete] [Help] [Maximize] [Print] [Quit]  [Type]  [Units] [Write]
```

I sada je potrebno to sve i potvrditi kako bi se stvarno i snimilo na disk. Ono što još u ovom trenutku možemo promijeniti je vrsta datotečnog sustava, s kojom ćemo kasnije formatirati ovu *particiju*. Naime, ako u sljedećim koracima želimo ovu *particiju* formatirati s nekim neobičajenim datotečnim sustavom (pr. *NTFS*-om za Windowse ili nekim drugim), sada je trenutak za to.

Mi ćemo promijeniti vrstu budućeg datotečnog sustava na ovoj *particiji* pa ćemo odabrati: `[Type]` i stisnuti *ENTER*.
Sada će nam se pojaviti mogućnost odabira vrste datotečnog sustava za našu *particiju*, poput ovog na ispisu dolje (skratili smo ispis):

```
 01 FAT12                 10 OPUS                 61 SpeedStor               AF HFS / HFS+
 02 XENIX root            11 Hidden FAT12         63 GNU HURD or SysV        B7 BSDI fs
 03 XENIX usr             12 Compaq diagnostics   64 Novell Netware 286      B8 BSDI swap
 04 FAT16 <32M            14 Hidden FAT16 <32M    65 Novell Netware 386      BB Boot Wizard hidden
 05 Extended              16 Hideen Fat16         70 DiskSecure Multi-Boo    BE Solaris boot
 06 FAT16                 4F QNX4.x 3rd part      75 PC/IX                   BF Solaris
 07 HPFS/NTFS/exFAT       50 OnTrack DM           80 Old Minix               C1 DRDOS/sec (FAT-12)
 08 AIX                   51 OnTrack DM6 Aux1     81 Minix / old Linux       C4 DRDOS/sec (FAT-16 <
 09 AIX bootable          52 CP/M                 82 Linux swap / Solaris    C6 DRDOS/sec (FAT-16)
 0A OS/2 Boot Manager     53 OnTrack DM6 Aux3     83 Linux                   C7 Syrinx
 0B W95 FAT32             54 OnTrackDM6           A9 NetBSD                  DA Non-FS data
 0C W95 FAT32 (LBA)       55 EZ-Drive             AB Darwin boot             DB CP/M / CTOS / ...
 0E W95 FAT16 (LBA)       56 Golden Bow           AF HFS / HFS+              E1 DOS access
 0F W95 Ext'd (LBA)       5C Priam Edisk          AF HFS / HFS+              E3 DOS R/O
```

U ovom koraku sâmo trebamo s tipkom *ENTER* potvrditi ulazak u ovaj menî te kada se pojavi poruka: `Enter filesystem type:` upisati identifikator datotečnog sustava, poput broja: `07` (koji je zadužen za: `HPFS/NTFS/exFAT`) te ponovno potvrditi sa *ENTER*.

Međutim mi ćemo ostati na vrsti budućeg datotečnog sustava: `Linux` (oznaka `83`).



Za cijelu listu podržanih vrsta particija odnosno datotečnih sustava na njima pogledajte poglavlje:
**13.4.1 Napredno: MBR Detaljnije.**

Cijelu proceduru završavamo tako što ćemo sada odabrati: `[Write]` te potvrditi s *ENTER*, a potom sve moramo dodatno potvrditi, tako što ćemo upisati `yes` i ponovno stisnuti tipku *ENTER*. Tek sada je particija kreirana i pripremljena za formatiranje s datotečnim sustavom koji je standardno označen pod stupcem: `FS Type`. U našem slučaju je to `Linux`, što podrazumijeva: ***ext2*, *ext3*** ili ***ext4*** datotečne sustave.

Sada možemo izaći iz programa `cfdisk` odabirom: `[Quit]` te *ENTER*. Dakle particija je sada spremna, a oznaka joj je: `/dev/sdb1` te je njena vrsta označena kao `Linux`, što i želimo jer ju kasnije želimo formatirati s nekim od Linux datotečnih sustava.

Istu proceduru kreiranja particije, mogli smo napraviti i s potpuno tekstualnim naredbama `fdisk` (za ***MBR*** particije) i `gdisk` (za ***GPT*** particije) ili programom `parted` s kojim ćemo se kasnije također upoznati.

Sada imamo kreiranu particiju koju moramo formatirati.



Za proceduru *formatiranja*, što je ona i čemu služi pogledajte poglavlje:
**13.6.2. Formatiranje particije.**

## Brisanje particije (i/ili ponovno kreiranje prema potrebi)

U ovom primjeru probajmo obrisati upravo napravljenu *particiju* `/dev/sdb1`, ali ne s programom `cfdisk` već s tekstualnom naredbom `fdisk` (pogotovo pošto se rad o ***MBR*** vrsti particije).

```
fdisk  /dev/sdb

Welcome to fdisk (util-linux 2.32.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

Sada za brisanje stisnimo tipku **d**:

```
Command (m for help): d
Selected partition 1
Partition 1 has been deleted.
```

I potom snimimo stanje na disk (tipka **w**):

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Sada smo obrisali tu (jednu jedinu) particiju na disku `/dev/sdb`.

Zatim ponovno s programom fdisk kreirajmo istu primarnu particiju /dev/sdb1:

```
fdisk  /dev/sdb
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
```

Pošto želimo kreirati primarnu *particiju*, stisnimo tipku **p**

```
Select (default p): p
```

S obzirom da želimo kreirati prvu primarnu particiju upišimo **1**

```
Partition number (1-4, default 1): 1
First sector (2048-16777215, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-16777215, default 16777215):
Created a new partition 1 of type 'Linux' and of size  2 GiB.
```

I potom snimimo stanje na disk (tipka **w**):

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

S istom naredbom pogledajmo novo stanje *particija* na disku:

```
fdisk -l /dev/sdb
Disk /dev/sdb: 2 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x1df9cb9e

Device     Boot Start      End  Sectors Size Id Type
/dev/sdb1        2048 16777215 16775168   2G 83 Linux
```

Pogledajmo kako sada izgleda naš disk /dev/sdb i njegove *particije*, s naredbom lsblk:

```
lsblk /dev/sdb

NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdb      8:16   0   2G  0 disk
└─sdb1   8:17   0   2G  0 part
```

Ovdje vidimo kako smo na disku sdb kreirali prvu (jednu) particiju sdb1 te kako je ta particija veličine 2 GB.

Nakon ovog koraka, particija je spremna na formatiranje sa željenim datotečnim sustavom.



**Po pitanju rada s particijama pogledajte i sljedeće poglavlje!**

**Izvori informacija: (667),(669), `man lsblk`, `man cfdisk`, `man fdisk`, `man gdisk`.**

### 13.6.1.1. Program `parted` i (napredni) rad s particijama

Program `parted` je alat naredbenog retka u Linuxu za particioniranje diska i promjenu veličine particije. S njim je moguće stvarati, uništavati, mijenjati veličinu, premještati i kopirati razne vrste Linux particija, slično kao i s već navedenim programima poput: `fdisk`, `gdisk`, `cfdisk` i `cgdisk` o kojima smo govorili prije nekoliko poglavlja.
Međutim, program `parted` je znatno napredniji od njih i nudi nam puno više mogućnosti.
Pošto ćemo u ovoj cjelini raditi s njima, upoznat ćeom se i sa osnovnim stvarima, poput kreiranja i brisanja particija i slično.

***Morate biti oprezni pri upravljanju particijama diska jer se podaci mogu obrisati ili izgubiti!***

Naredba/program `parted` može se koristiti u dva načina rada: interaktivni način i način rada u vlastitom naredbenom retku.

Primjerice, ako ju pokrenemu samo ovako (bez opcija i parametara):

```
parted
GNU Parted 3.5
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

← Vidimo da se na kraju pojavio *parted* naredbeni redak, koji očekuje unos specifičnih *parted* naredbi.
Naredba za izlazak je `quit`.

Za ispis particija na disku (konkretno imamo disk `/dev/sda`), koristimo *parted* naredbu `print`:

```
(parted) print
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sda: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system     Flags
 1      1049kB  20.4GB  20.4GB  primary  xfs             boot
 2      20.4GB  21.5GB  1073MB  primary  linux-swap(v1)  swap
```

← Na dnu ispisa vidimo naš disk `/dev/sda` s dvije particije, koje su označene s **1.** i **2.**. Osim toga vidimo da su veličine diskova prikazane u MB ili GB (stupac `Size`). Vidimo i da su obije particije primarne (stupac `Type`).
Vidimo i datotečni sustav s kojim su formatirane (stupac `File system`) te zastavice (stupac `Flags`): primjerice prva particija je označena kao `boot` što znači da se s nje može pokretati operativni sustav odnosno da je ona i sistemska particija.

**Dodavanje diska i particioniranje**
U primjeru koji slijedi dodajemo novi hard disk, koji će biti vidljiv kao `/dev/sdb`. Stoga pokrećemo `parted`:

```
parted
(parted) select /dev/sdb
Using /dev/sdb
```

← Ovdje vidimo da smo nakon pokertanja *parted*-a promjenili disk u `/dev/sdb` s naredbom `select`.

A sada ćemo ispisati particijku tablicu na tom disku, da vidimo što se na njemu nalazi:

```
(parted) print
Error: /dev/sdb: unrecognised disk label
Model: VBOX HARDDISK (scsi)
Disk /dev/sdb: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
```

← Vidimo da ovaj disk nije particioniran; zbog poruke: „`Partition Table: unknown`".

**Kreiranje particijske tablice**
U ovom koraku moramo kreirati **MBR** zapis i particijsku tablicu. Ovdje za PC kompatibilna računala imamo dva izbora odnosno načina particioniranja: s **MBR** shemom ili s **GPT** shemom. Pri tome je *parted* naredba za kreiranje **MBR** zapisa i particijske tablice `mklabel`.
Parted opcija za **MBR** shemu particioniranja jest `msdos`, dok je za **GPT** shemu particioniranja naziv `gpt`.
U našem slučaju kreirajmo **MBR** particijsku shemu:

```
(parted) mklabel msdos
Warning: The existing disk label on /dev/sdb will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? Y
```

← U zadnjem koraku moramo potvrditi izradu MBR zapisa i particijske tablice sa **`Y`**.

Isto smo mogli postići i iz naredbenog retka, ovako:
```
parted /dev/sdb mklabel msdos
```
← I ponovno je na kraju potrebno sve potvrditi, također sa **Y**.

Sada pogledajmo što smo dobili:
```
(parted) print
Model: VBOX HARDDISK (scsi)
Disk /dev/sdb: 41943040s
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End  Size  Type  File system  Flags
```
← Vidimo samo da je particijska tablica odnosno **MBR** shema kreirana, prema poruci: `Partition Table: msdos`.
Sada je potrebno kreirati barem jednu particiju.
Moguće je ili nastaviti s *parted* naredbom `mkpart` te odgovarati na pitanja ili koristiti naredbeni redak.

**Kreiranje particije (iz *parted* naredbenog retka)**
Sada prelazimo na naredbeni redak, jer je lakši zbog toga što sve *parted* naredbe možemo koristiti u jednom retku.
Dakle naš disk `/dev/sdb` je veličine 20GB, a na njemu želimo napraviti jednu ***primarnu*** particiju veličine cijelog diska.
```
parted /dev/sda mkpart primary ext4 0% 100%
```
← Sada je *parted* kreirao particiju veličine cijelog diska, od početka (`0%`) do kraja cijelog diska (`100%`).

Odnosno kreiranje particijske tablice i kreiranja ovakve particije smo mogli napraviti i u jednom koraku:
```
parted /dev/sdb mklabel msdos mkpart primary ext4 0% 100%
```



Vezano za navedeni početak particije od `0%`, ova nula ne označava nulti sektor diska, već dio nakon **MBR** (sektor **0**) i međuprostora od sektora **1** do **63**. Dakle prema **MBR** shemi particioniranja `0%` se odnosi na minimalno **64**-ti sektor (jer se broji od **0**).
Za detaljniji pogled na disk te **MBR**, međuprostor i **VBR** pogledajte poglavlje:
**12.1.2. Boot proces detaljnije.**

Vezano za kreiranje MBR zapisa i particijske tablice te particioniranja diska na jednu primarnu particiju, slično bismo dobili i sa sljedećim naredbama (u dva koraka). Ovdje smo, međutim definirali početni sektor kao sektor (`s`), a kraj u **GB**:
```
parted /dev/sdb mklabel msdos
parted /dev/sdb mkpart primary ext4 2048s 20GB
```
← Ovdje smo kao početni sektor odabrali **2048**-i sektor zbog poravnavanja koje bi *parted* ionako preporučio.

*Parted* nam nudi mogućnost definiranja različitih vrsta jedinica za prikaz upotrebom naredbe `unit`, pa primjerice možemo koristiti: `s` – za prikaz u sektorima, `%` – za prikaz u postotku, `GB` – za prikaz u gigabajtima, `TB` – za prikaz u terabajtima, … .



U trenutku particioniranja diska, unutar **MBR** zapisa na prvom sektoru diska, zapisuje se i jedinstveni identifikator diska.



Vezano za navedeni identifikator diska, ali i oznaku particije koja se kreira kasnije, odnosno tek tijekom formatiranja particije (tzv. **UUID** oznaku), pogledajte poglavlje:
**13.6.2.3. Universally unique identifier (UUID) oznake.**

Ovaj identifikator se primjerice za disk `/dev/sda` može vidjeti s naredbom `gdisk`:
```
gdisk -l /dev/sda |  grep "Disk identifier"
Disk identifier (GUID): 3128486B-74DD-429E-9575-694059B75832
```

Naredba `gdisk` dolazi u istoimenom programskom paketu imena: `gdisk`.

U slučaju da smo krivo definirali početak ili kraj diska u sektorima, MB i slično, dobili bismo ovakvu poruku:
```
Warning: You requested a partition from 0.00B to 21.5GB (sectors 0..41943039).
The closest location we can manage is 512B to 1048kB (sectors 1..2047).
```

Ovdje je važno razumjeti poravnavanje particija, a stoga *parted* i kaže "`(sectors 1..2047)`", jer je preporuka, zbog navedenog poravnavanja, da prvi sektor bude upravo sektor **2048** (ili njegov višekratnik).

Naime programi za particioniranje diska poput `parted`, `fdisk` i `gdisk` ne počinju particiju od sektora **64** (kako bi bilo za očekivati kod **MBR** sheme), već od sektora 2048 zbog dva razloga:

- *Poravnanja particije* – moderni diskovi i SSD-ovi obično imaju sektore veličine 4096 bajta (4KB). Početak particije na sektoru 2048 omogućava poravnanje particije na granici od 4KB, što poboljšava performanse diska.
- *Kompatibilnost sa GPT* – GPT (*GUID Partition Table*) je moderniji i fleksibilniji način particioniranja diska. Prostor od sektora 0 do 2048 omogućuje potencijalnu rezervaciju prostora za GPT metapodatke. Naime GPT ne koristi MBR particijsku tablicu, već umjesto toga koristi GPT metapodatke. Zbog ovog odmaka od 2048 sektora omogućena je laka konverzija diska iz MBR (*Master Boot Record*) u GPT format u budućnosti.

Pogledajmo detaljnije kako sada (prema sektorima) izgledaju zapisi na našem disku `/dev/sdb`:

```
parted /dev/sdb unit s print free
Model: VBOX HARDDISK (scsi)
Disk /dev/sdb: 41943040s
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start  End        Size       Type     File system  Flags
        2s     2047s      2046s               Free Space
 1      2048s  41943039s  41940992s  primary
```

← Ovdje vidimo prostor na početku diska (od sektora dva [`2s`] do `2047s`) označen kao slobodan prostor, iako znamo da je početak diska rezerviran za *MBR* (ili *GPT*) zapise. U konkretnom slučaju pošto je disk particioniran s *MBR* shemom, u nultom sektoru (`0s`) se nalazi *MBR* zapis, dok je kod *GPT* sheme u sektoru nula (`0s`) takozvani zaštitini *MBR* zapis.
U sektoru jedan (`1s`) se kod *GPT* nalazi *GPT* zaglavlje, a upravo zbog kompatibilnosti s *GPT*, ovdje je vidljiv početak tek od sektora dva `2s`. Ono što mi ovdje vidimo jest da sektori od 2 do 2048 nisu iskorišteni za particiju, ali na njima se (u konkretnom slučaju MBR sheme particioniranja) nalaze međuprostor između *MBR* i **VBR** te u konačnici i **VBR** (particijski) zapis. Na kraju vidimo da prva particija počinje tek od sektora 2048 (`2048s`).



Vezano za **MBR** shemu pogledajte poglavlje:
**13.4.1. Napredno: MBR Detaljnije.**



Nakon kreiranja particije, *udev* sustav je možda neće prepoznati. Stoga je potrebno prisiliti *udev* da ponovno pročita sve diskove i prepozna sve (potencijalno nove) particije!

Stoga se sada preporuča izvršiti naredbu:

```
udevadm settle
```

ili naredbu koja će učinit isto:

```
partprobe
```

Za provjeru diskova i particija, možemo pokrenuti naredbu:

```
lsblk
```

**Dodavanje zastavica (flags)**
Moguće je i dodavanja zastavica poput boot zastavica (ako imamo sistemsku particiju [za *GRUB2* i `/boot/` direktorij]).
Za *MBR* (BIOS) *boot* odnosno postavljanje ove zastavice za prvu particiju (br. **1.**) `/dev/sdb1`, tada bismo to napravili sa:

```
parted /dev/sdb set 1 boot on
```

Odnosno obrisali bismo ovu zastavicu sa:

```
parted /dev/sdb set 1 boot off
```

**Brisanje particije**
Ako želimo obrisati primjerice prvu particiju (broj **1.**) na disku `/dev/sdb`, to možemo učiniti sa:

```
parted /dev/sdb rm 1
```

**Rescue particije**
Ako smo greškom obrisati particiju, primjerice na disku `/dev/sdb`, to možemo pokušati popraviti sa:

```
parted /dev/sdb rescue
Start? 2048s
End? 100%
```

← Odabrali smo početni sektor particije (**`2048s`**), te kraj diska (**`100%`**). Ako particija bude pronađena *parted* će ispraviti (osvježiti) i particijsku tablicu, jer se popravkom particije mora osvježiti i particijska tablica.

**Izvori informacija: (1630),(1631),(1632),** **`man lsblk`**,**`man parted`**,**`man udevadm`**

### 13.6.2. Formatiranje particije

Nakon što smo kreirali particiju, kako bismo ju mogli koristiti, moramo ju formatirati s nekim datotečnim sustavom. *Formatiranje* se odnosi na instalaciju datotečnog sustava na odabranu particiju. Naime nakon što smo kreirali *particiju* na disku, potrebno je zapravo „instalirati“ datotečni sustav. Sâmo formatiranje odnosno oblikovanje diska je postupak pripreme uređaja za pohranu podataka kao što je tvrdi disk, SSD disk, disketa ili USB disk za početnu uporabu.

Ako ćemo biti još malo precizniji, prije faze particioniranja diska zapravo postoji i prvi dio procesa *formatiranja* koji izvodi osnovnu pripremu medija, a često se naziva *niskom razinom formatiranja* (engl. *low-level format*), a koja je već napravljena od strane proizvođača sâmog diska. Mada se u nekim posebnim slučajevima ona može i ponovno napraviti.
Postoje i slučajevi u kojima, kada diskove stavljamo u hardverski RAID kontroler, RAID kontroler nakon što se kreira željeno RAID polje diskova, na tim diskovima prvo radi upravo ovakvu nisku razinu formatiranja.

Međutim pošto je particioniranje uobičajeni pojam za drugi dio procesa, što čini uređaj za pohranu podataka vidljivim operativnom sustavu, stoga se može reći kako je klasično formatiranje koje slijedi nakon particioniranja, treće u nîzu.

Ovaj treći dio procesa se obično naziva formatiranje na visokoj razini (engl. *high-level formatting*), a najčešće se odnosi na proces stvaranja datotečnog sustava. Pri tome pojam "*format*" podrazumijeva radnju u kojoj je novi medij diska potpuno spreman za pohranu datoteka (i direktorija).

Drugim riječima formatiranje na visokoj razini je proces postavljanja praznog datotečnog sustava na particiji diska ili logičkog volumena (pr. ***LVM***-a). Tijekom formatiranja particije ili logičkog volumena (pr. *LVM* diska) moguće je i dodatno skenirati površinu diska u potrazi za neispravnim dijelovima diska, što može potrajati neko vrijeme. Ipak ovakvo skeniranje je obično ostavljeno kao dodatna opcija (za one koji to baš žele).

Pogledajmo što smo do sada napravili: u prvom koraku nakon instalacije particijske tablice i **MBR** zapisa (*gledano od lijevo prema desno*). Pri tome smo **MBR** zapis i particijsku tablicu kreirali s naredbom:

```
parted /dev/sdb mklabel msdos
```

| MBR shema diska | | |
|---|---|---|
| **MBR (512 bajta)** | **Međuprostor (32kB)** | **Prazan prostor na disku** |
| → <br> ***Booot kôd*** <br> ***Particijska tablica*** <br> ***MBR potpis*** | → <br> ***Boot kôd*** <br> *(ako se koristi)* | |

---

Pogledajmo što se zatim dogodilo nakon particioniranja diska na jednu particiju:

| MBR shema diska s jednom particijom | | | |
|---|---|---|---|
| **MBR (512 bajta)** | **Međuprostor (32kB)** | **Boot record prve particije (Volume Boot record)** | **Prva particija (za EXT4 datotečni sustav)** |
| → <br> ***Booot kôd*** <br> ***Particijska tablica*** <br> ***← Osvježeno sa zapisom za novu particiju br.1*** <br> ***MBR potpis*** | → <br> ***Boot kôd*** <br> *(ako se koristi)* | → <br> *Boot kôd* <br> *(**ne koristi ga GRUB2**)* <br> *Naziv particije* <br> *Vrsta medija* <br> *Vrsta datotečnog sustava* <br> *(**EXT4** u ovom slučaju)* <br> . . . | ***Prostor za datotečni sustav*** |

Tek sada slijedi formatiranje particije.



*Mi ćemo particiju formatirati s Linuxovim **ext4** datotečnim sustavom, na sljedeći način, te ćemo koristiti standardne postavke formatiranja odnosno nećemo navoditi posebne opcije ili parametre.*

*Sintaksa je: naredba za formatiranje particija koju želimo formatirati (njen blok uređaj).*

Naredba koju ćemo pri tome koristiti je `mkfs.ext4` (*engl. Make file system*). U našem slučaju to će onda biti:

```
mkfs.ext4 /dev/sdb1
```

Nakon toga će se kreirati datotečni sustav na toj particiji, što može potrajati ovisno o veličini particije.

Tijekom formatiranja particije dobivat ćemo informacije poput sljedećih:

```
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
131072 inodes, 524286 blocks
26214 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=536870912
16 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

Pogledajmo opis vidljivih informacija koje smo dobili tijekom formatiranja s ***ext4*** datotečnim sustavom (sâmo osnovno):

- `Filesystem label=` je oznaka (*labela)* particije; mi nismo stavili nikakav naziv, stoga je prazno.
- `OS type: Linux` znači kako se radi o vrsti particije s ID brojem `83` koji označava Linux vrstu particije.
- `Block size=4096` znači kako je veličina bloka 4096 *bajta* (4kB), ovo nije veličina ***LBA*** bloka na disku, već je ovo zapravo veličina klastera, kao najmanjeg bloka podataka koji se može snimiti na disk. Osim toga ovo je automatski odabrana vrijednost. Moguće ju je i ručno promijeniti i to sâmo prije sâmog formatiranja particije.
- `131072 inodes, 524286 blocks` znači kako smo formatirali ovu particiju diska s ***ext4*** datotečnim sustavom, koji je kreirao ukupno `524286` blokova/klastera (od 4kB): 4.096 bajta x 524.286 blokova = 2GB. Dodatno je vidljivo kako na ovom datotečnom sustavu imamo dostupno 131.002 *inode* unosa.
- `26214 blocks (5.00%) reserved for the super user` znači kako je sustav rezervirao 5% kapaciteta ove particije za potrebe sâmog sustava. U slučajevima da se sustav prepuni podacima (datotekama) ovo je zaštitni mehanizam sustava.
- Vidimo kako postoji *Superblock* (`Superblock backups stored on blocks:`) i kako su njegove sigurnosne kopije kreirane na pozicijama: `32768, 98304, 163840, 229376, 294912`
- Postoje i drugi unosi poput:
    - ***Group tables*** odnosno tablica s grupama te ***inode tables*** to jest *inode* tablica.
    - ***Journal*** to jest takozvani *journaling* sustav.
- Nadalje imamo i blok grupe (`16 block groups`) i vidimo koliko je blokova u svakoj od njih (`32768 blocks per group`), kao i koliko ima *inode* unosa po grupi (`8192 inodes per group`).

Naime sve ovo je specifično za datotečni sustav s kojim smo formatirali našu particiju, pa u ovom slučaju govorimo o specifičnostima ***ext4*** datotečnog sustava.

U slučaju da smo istu particiju primjerice htjeli formatirati s ***XFS*** datotečnim sustavom, to bi napravili sa:

```
mkfs.xfs /dev/sdb1
```

Prema tome naredba koju bi koristili je `mkfs.xfs` (*engl. Make file system*).



Važno je razumjeti kako se tijekom formatiranje particije, kreira i posebna identifikacijska oznaka datotečnog sustava na toj (konkretnoj) particiji. Dakle svaka formatirana particija dobiva svoju jedinstvenu identifikacijsku oznaku koja se naziva ***UUID*** odnosno točnije ***Filesystem UUID***.

***UUID*** oznaka se može koristiti kao identifikator formatirane particije diska na način da se primjerice u datoteci **`/etc/fstab`** navodi upravo ona, a ne oznaka particije diska, poput primjerice `/dev/sda1`, `/dev/sdb1` i tako dalje. Za poveznicu oznake particije diska (**`/dev/*`**) i ***UUID*** oznake, možete koristiti naredbu: `blkid`.

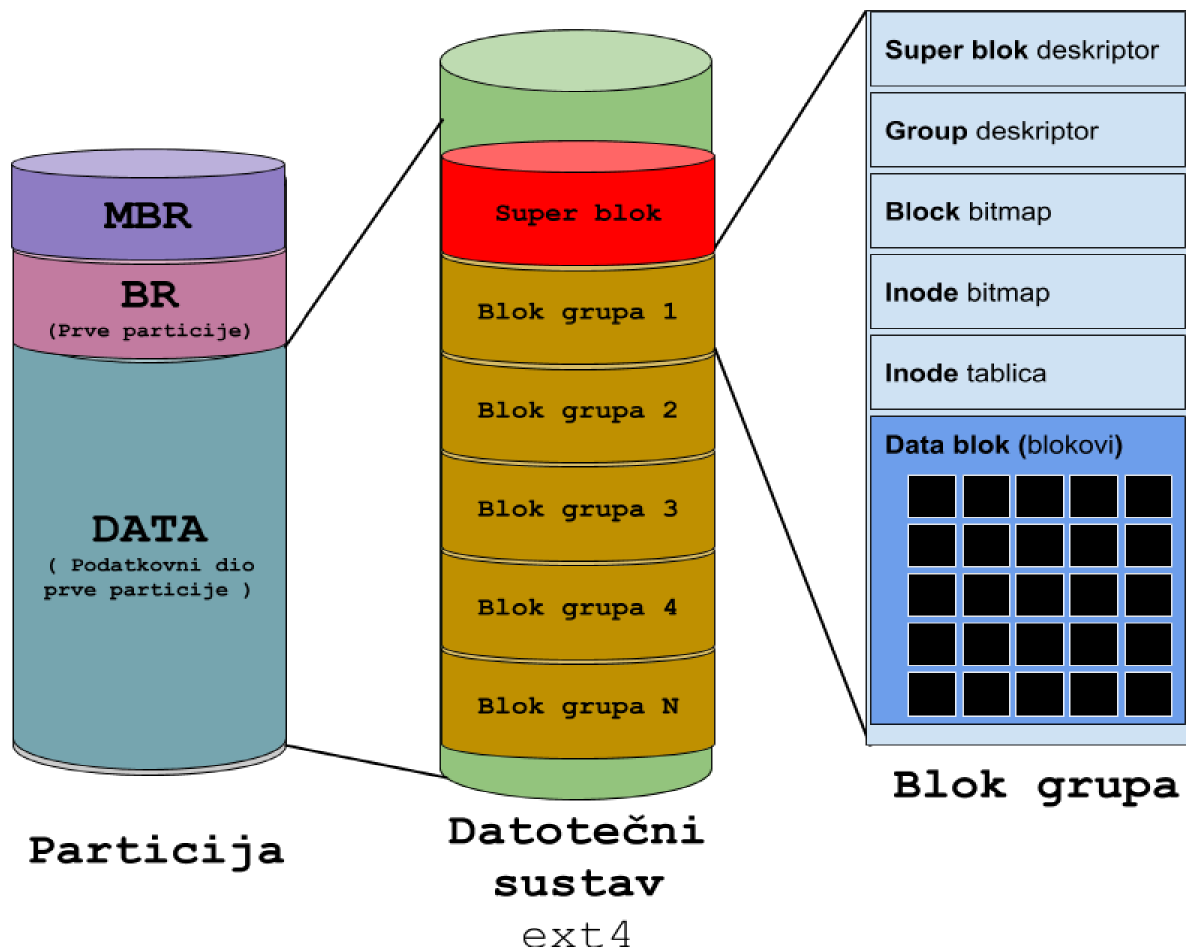


Za više detalja o datoteci **`/etc/fstab`** pogledajte:
**13.7.3. Ograničenje zauzeća prostora *na disku*** (*quota*).
**13.8. Što je datoteka /etc/fstab.**

**Izvori informacija: (669),(670),** **`man mkfs.ext4`**,**`man mkfs.xfs`**,**`man 5 ext4`**,**`man 5 xfs`**.

### 13.6.2.1. *Ext4* datotečni sustav detaljnije

**Slijedi napredna cjelina!**
Prvo pogledajmo sliku 115 koja prikazuje strukturu ***ext4*** datotečnog sustava. s
**lika 115. Pogled na strukturu ext4 datotečnog sustava.**




***Superblock*** odnosno super blok i njegova lokacija(e) na disku se određuju u trenutku formatiranja particije. Kako smo vidjeli u prijašnjem ispisu, super blok je kopiran na još pet pozicija na disku, koje se koriste kao rezervne kopije super bloka, a u njih se kopira izvorni super blok tijekom svake njegove promjene. Za više informacija o super bloku, za ***ext4*** datotečni sustav o kojemu i pričamo, možemo koristiti naredbu: `dumpe2fs`, a to bi konkretno za particiju `/dev/sdb1` bilo ovako:

```
dumpe2fs -h /dev/sdb1
```

Detalje ove naredbe ovdje nećemo objašnjavati.
Naime super blok sadrži informacije o cijelom datotečnom sustavu (konkretno ***`ext4`***) na konkretnoj particiji, poput:

- Vrste i inačice datotečnog sustava te veličinu datotečnog sustava, kao i njegovo stanje.
- Te drugih informacija kao i raznih meta podataka datotečnog sustava.

*Super blok* struktura predstavlja cijeli datotečni sustav i nužna je za njegovo funkcioniranje.
Bez super bloka nije moguće niti montirati (*mountati)* datotečni sustav.

**Blok grupe**
Cijeli datotečni sustav ***ext4*** se sastoji od blok grupa. Ove blok grupe nemaju nikakve veze s blokovima podataka na blok uređajima na nižoj razini diskovnog sustava, već se nalaze na ovoj (višoj) razini datotečnog sustava. Svaka blok grupa sadrži kopiju krucijalnih kontrolnih informacija važnih za cijeli datotečni sustav, poput super bloka (ako je unutar konkretne blok grupe) te *group deskriptora*, kao i dijela koji sadrži dio metapodataka za blokove (*block bitmap*). Ona sadrži i *inode bitmap* te blokove u koje se zapisuju stvarni podaci (*data* blokovi). *Group* deskriptor sadrži lokacije gdje se unutar blok grupe nalaze: *Block bitmap* i *inode* tablica. Dakle ukupno gledajući; toliko je grupnih deskriptora, koliko postoji blok grupa. Pri tome *blok bitmap* prati iskorištenost *data* blokova unutar blok grupe, dok *inode bitmap* prati koji unosi u *inode* tablici su u upotrebi.
U konačnici možemo reći kako ***ext4*** datotečni sustav alocira i koristi prostor za pohranu u nizu blokova odnosno blok grupa. *Inode* tablica pohranjuje strukturu *inode*-ova.
Pozicija *inode* tablice unutar blok grupe definirana je u polju *group deskriptor*. *Inode* je podatkovna struktura za svaki pojedini direktorij i datoteku. Dakle *inode* čini podatkovnu strukturu za objekte datotečnog sustava kao što su datoteke i direktoriji (mape) te njihovi atributi: ovlasti, oznake vremena kreiranja, pristupanja i slično.

Ako ćemo biti još precizniji, u ovoj tablici se nalaze i točne lokacije na kojima se nalaze podaci (iz datoteka), koji su pohranjeni za svaki *inode* unos, a koji se nalaze u dijelu „*Data blok*". Dakle konkretno: sâmi podaci svake datoteke se pohranjuju u *Data blok* dio koji predstavlja *inode* unose, svaki sa svojim identifikatorom, a koji su pak popisani u *inode* tablici.



Za više detalja o funkcioniranju *inode*-a pročitajte poglavlje:
**4.5. Datotečni sustav detaljnije.**

Prema svemu navedenom možemo zaključiti: svaki bît u ***Data** bloku*, a sâmim time vidljiv u ***inode bitmapu*** predstavlja jedan unos u *inode* tablici. Zbog toga što je veličina *inode bitmapa* jedan blok koji je standardne veličine 4KB odnosno 32.768 bitova (32kb) time je ograničen maksimalan broj blokova u svakoj pojedinoj blok grupi. Analogno tome i maksimalan broj *inode* unosa u jednoj blok grupi je isti (32.768).

Naravno, ako formatiramo particiju s većom veličinom bloka i ova računica se mijenja.
Međutim u trenutku formatiranja particije standardno se ova vrijednost postavlja na dosta nižu, u našem slučaju: `8192 inodes per group` dakle samo `8192` *inode*-a po blok grupi. I još jedan mali detalj, a to je informacija `131072 inodes, 524286 blocks` znači kako na cijeloj particiji koju smo upravo formatirali možemo pohraniti maksimalno sveukupno `131 072` datoteka i direktorija s obzirom na činjenicu kako svako od njih mora zauzeti barem jedan inode broj odnosno poziciju.
To možemo i provjeriti kada *montiramo* našu particiju, te pozovemo naredbu `df` na sljedeći način:

```
df -i /dev/sdb1
```

Dobit ćemo nešto poput sljedećeg ispisa (gledamo ispis samo naše particije):

```
Filesystem     Inodes IUsed  IFree IUse% Mounted on
/dev/sdb1      131072    11 131061    1% /DISK-2
```

Vidimo da ukupno imamo `131072` inode unosa (stupac: `Inodes`) od kojih je u upotrebi njih `11` (stupac: `IUsed`), a slobodno njih `131061` (stupac: `IFree`) što znači kako ih je u upotrebi samo `1%` (stupac: `IUse%`).

Općenito gledano upotreba blok grupa u datotečnim sustavima *ext2/3/4* donijela je mnoge prednosti, od kojih ćemo navesti neke:

- Kontrolne strukture se nalaze u svakoj blok grupi pa je u slučaju grešaka na super blokovima ili u slučaju gubitka istih, lako povratiti podatke iz razine svake pojedine blok grupe.
- S obzirom na ove kontrolne strukture (meta)podataka unutar svake blok grupe dobilo se na performansama, smanjivanjem pretrage po disku, jer se inode tablica za sve inode unose (i praktično podatke) nalazi unutar svake blok grupe, što pogotovo za mehaničke diskove znači manje pomicanja glava za čitanje i brzo dohvaćanje inode tablice te pozicije *inode*-a koji se traži, kao i dohvaćanje sâmih podataka iz traženog *inode*-a.
- *Ext4* je uveo i takozvane fleksibilne blok grupe, koje su proširenje standardnih, tako da jedna fleksibilna blok grupa može sadržavati više blok grupa koje, kao što je bilo i prije, sadrže više blokova.
- *Ext4* standardno podržava takozvani ***Journaling*** koji je uveden još u ***ext3*** datotečnom sustavu, a u ***ext4*** je standardno u upotrebi. *Journaling* nam daje novu funkcionalnost u kojoj se umjesto pisanja podataka izravno na područje datotečnog sustava gdje bi se standardno trebali i zapisivati podaci, *journaling* zapisuje podatke o datotekama zajedno s meta podacima na posebno određeno područje na disku. Jednom kada su podaci na tvrdom disku sigurno zapisani, mogu se spojiti ili dodati u ciljnu datoteku s vrlo malom vjerojatnosti za gubitkom podataka. Naime tek kada su svi podaci uredno zapisani na disk, *journaling* se ažurira tako da će datotečni sustav ostati u dosljednom stanju u slučaju kvara sustava prije nego što se svi podaci u *journalingu* uredno zapišu. Pri sljedećem pokretanju računala datotečni sustav će provjeriti moguće nekonzistentnosti, a podaci koji su eventualno preostali u *journalingu* bit će snimljeni na podatkovna područja diska na kojima i trebaju završiti, sve u cilju kako bi dovršili ažuriranja ciljne datoteke. Na ovaj način se podigla razina konzistentnosti podataka u slučajevima naglog i neplaniranog gašenja računala te malo usporio pristup disku. Ipak, moguće je odabrati jedan od tri načina rada *journalinga*: od performantnijeg i nesigurnijeg do malo sporijeg i sigurnijeg.
- Posebnim mehanizmima alociranja/zapisivanja podataka datotečni sustav pomaže minimiziranju ***fragmentacije*** odnosno pomaže u slučajevima kada se ona ipak dogodi.
- Korisni programi za ***ext2, ext 3*** ili ***ext4***, koji dolaze u softverskom paketu **`e2fsprogs`** su: `badblocks`, `debugfs`, `dumpe2fs`, `e2fsck`, `e4crypt`, `resize2fs`, `tune2fs` i mnogi drugi.

Za provjeru konzistencije i popravke datotečnog sustava imamo: `fsck` odnosno `fsck.ext4` ili `e2fsck` naredbe.

**Izvori informacija: (312),(313),(314),(315),`man dumpe2fs`,`man badblocks`,`man debugfs`, `man dumpe2fs`,`man e2fsck`,`man e4crypt`,`man resize2fs`,`man tune2fs`,`man fsck`,`man 5 ext4`.**

### 13.6.2.2. Provjera konzistencije podataka datotečnog sustava na particiji diska

**Slijedi napredna cjelina!**

Datotečni sustav jedna je od najkritičnijih komponenti računala. Bez datotečnog sustava, računalo ne može pohraniti nikakve podatke (datoteke i direktorije) na tvrdi disk, bilo da se radi o mehaničkim diskovima ili SSD-u.
Zapravo, datotečni sustav se mora praktično instalirati na particiju diska prije nego što se operativni sustav može instalirati na tvrdi disk. Dakle mora postojati nešto u što se mogu pohraniti datoteke (i direktoriji) operativnog sustava.
I tijekom instalacije operativnog sustava, na sistemsku particiju diska instalira se datotečni sustav.
Datotečni sustav odnosno sustav datoteka stvara se softverom, prema tom softver i zapisuje i softver čita podatke s njega.
Već je poznato da svaki kompleksni softver ima greške, pitanje je samo jesu li već otkrivene i poznate ili će to tek biti.
Na kraju priče, naši podaci su nam iznimno važni pa imamo puno povjerenje u datotečni sustavi i softveru koji ga stvara i koristi. Ako nešto pođe po zlu, možemo izgubiti pristup dijelovima datotečnog sustava ili čak cijeloj particiji.

Suvremeni datotečni sustavi koji koriste takozvani *journaling*, bolje rješavaju probleme koji mogu biti uzrokovani iznenadnim gubitkom napajanja ili rušenjem sustava. Robusni su, ali nisu neuništivi. Ako se njihove interne tablice poremete, može se izgubiti trag o tome gdje se svaka datoteka nalazi na particiji (disku), koja je njena veličina, koje ime ima i koja su joj dopuštenja odnosno ovlasti za datoteku postavljene. Naredba `fsck` omogućuje nam provjeru ispravnosti datotečnog sustava. Ako ona pronađe bilo kakve probleme, obično ih može riješiti i za vas. Važno je znati da naredba `fsck` traži **root** (administratorske) ovlasti. U slučajevima kada se sustav nije mogao pokrenuti (*boot proces*) ili smo imali greške poput ***„input/output error“*** ili sličnih, pogledajmo kako provjeriti konzistenciju podataka na disku.

Važno je da particija koju provjeravamo nije montirana; stoga napravimo `umount` navedene particije.
Dakle ovaj primjer je za particiju diska koja nije sistemska (na kojoj nije instaliran operativni sustav).
Tek nakon toga krećemo dalje. Ako se radi primjerice o `/dev/sdb1` particiji diska `/dev/sdb` koja je formatirana sa ***ext2/3/4*** datotečnim sustavom, tada možemo koristiti naredbu `fsck` na sljedeći način.

Međutim, prvo provjerimo što sve imamo na ovom disku (`/dev/sdb`), s naredbom `fdisk`:
```
fdisk -l /dev/sdb

Disk /dev/sdb: 1 GiB, 1073741824 bytes, 2097152 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x150bc449

Device     Boot Start     End Sectors  Size Id Type
/dev/sdb1        2048 2097151 2095104 1023M 83 Linux
```
Na kraju ispisa vidimo de je naša testna particija: `/dev/sdb1` veličine `1023M`B.

Sada prvo provjerimo je li ona montirana u neki direktorij (mapu), što ćemo najjednostavnije dobiti s naredbom `lsblk`:
```
lsblk -f /dev/sdb

NAME   FSTYPE LABEL UUID                                 MOUNTPOINT
sdb
└─sdb1 ext4         085ec62e-bfdc-494e-b4ff-c9d23aa0d9fd /DISK2
```
Vidimo kako je naša particija (formatirana s `ext4` datotečnim sustavom) montirana u direktorij: `/DISK2`. Stoga ju prvo demontirajmo (*unmount*).

Ali prvo se za svaki slučaj pozicionirajmo u vršni dio (`/`) stabla datotečnog sustava:
```
cd /
umount /DISK2
```

## Ext2, ext3 i ext4 datotečni sustavi

Slijedi metoda sâmo za pronalazak grešaka na navedenoj particiji `/dev/sdb1`:
```
fsck /dev/sdb1

fsck from util-linux 2.32.1
e2fsck 1.45.6 (20-Mar-2020)
/dev/sdb1: clean, 13/65536 files, 8860/261888 blocks
```

Na ispisu je konkretno vidljivo da nema nikakvih grešaka. Ako želimo forsirati skeniranje particije (`-f`) te želimo automatski ispraviti sve pronađene greške (`-y`).

Za ***ext2, ext3 ili ext4*** datotečne sustave, to možemo postići na sljedeći način:

```
fsck -y -f /dev/sdb1

fsck from util-linux 2.32.1
e2fsck 1.45.6 (20-Mar-2020)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/sdb1: 13/65536 files (0.0% non-contiguous), 8860/261888 blocks
```

Želimo li popraviti *root* (**/**) particiju, to ne možemo uraditi sve dok je montirana.
Dakle prvo ju moramo demontirati ili napravimo sljedeće, za ***ext2, ext3*** ili ***ext4*** datotečne sustave***:***
*Ova promjena će biti će aktivirana kod prvog restarta, kod kojega će se pokrenuti **fsck** procedura na **root** particiji (samo za **ext2/3/4**):*

```
touch /forcefsck
```

U svakom slučaju navedene naredbe će vas obavijestiti o statusu provjere i eventualno nađenim greškama.

- Za provjeru stanja fragmentiranosti koristi se naredba `e2freefrag` ili u novijim inačicama i naredba `e4defrag`. Za provjeru stanja fragmentiranosti točno određene datoteke, naredba: `filefrag`
- Za promjenu parametar i opcija datotečnog sustava, možemo koristiti naredbu `tune2fs`, a za *debuggiranje* datotečnog sustava, možemo koristiti naredbu: `debugfs`.
- U slučajevima kada želimo proširiti ili smanjiti formatiranu particiju, možemo koristiti naredbu: `resize2fs` dok za izvlačenje detaljnijih informacija o stanju datotečnog sustava, naredbu: `dumpe2fs`

## XFS datotečni sustav

Ako zbog grešaka ne možete montirati **XFS** datotečni sustav, možete koristiti naredbu `xfs_repair` da provjerite njegovu konzistentnost. Ovu naredbu možemo koristiti samo na nemontiranom datotečnom sustavu za koji vjerujete da ima problem.
Ako `xfs_repair` nakon provjere svih elemenata prikazuje poruke o greškama, datotečni sustav ima nedosljednosti ili greške.
Naredbu `xfs_repair` tada moramo pokrenuti naredbu na sljedeći način:

```
xfs_repair /dev/sdb1
```

S druge strane naredba `xfs_db` pruža nam interni skup naredbi koji vam omogućuje da ručno otklanjate pogreške i popravljate ***XFS*** datotečni sustav. Ove interne naredbe vam omogućuju skeniranje datotečnog sustava te navigaciju i prikaz njegovih struktura podataka. Ako navedete opciju `-x` za omogućavanje stručnog načina rada, možete i izmijeniti strukture podataka.
Ovu naredba se može pozvati na slijedeći način (za istu particiju):

```
xfs_db /dev/sdb1
```

Za druge vrste datotečnih sustava, koriste se druge naredbe.

**Proširivanje kapaciteta datotečnog sustava**

Datotečni sustav ***XFS*** može se proširiti čak i dok je montiran, pomoću naredbe `xfs_growfs`.
Za mislimo slučaj u kojem smo proširili particiju `/dev/sdb1`. Naime particija ili disk mogu biti na ***RAID*** kontroleru ili ***LVM2*** polju koje je moguće proširiti. Nadalje disk ili particija mogu biti spojeni na virtualno računalo, primjerice kao ***QCOW2*** disk, pa ih je također vrlo lako moguće proširivati.



Za više informacija o proširenju ***LVM2*** polja i ***QCOW***2 diskova za virtualizaciju, pogledajte sljedeća poglavlja:
**13.11.2.5. Proširivanje kapaciteta LVM polja.**
**27.1.2.4. Disk image formati.**

Sa sljedećom naredbom možemo vidjeti veličinu ***XFS*** datotečnog sustava:

```
xfs_growfs -n /dev/sdb1
```

- ✓ Obratite pažnju na statistiku `blocks=` odnosno broj blokova koje datotečni sustav trenutno zauzima.

Međutim nakon svakog proširivanja particije, datotečni sustav toga nije svjestan te on ostaje na istim pozicijama na kojima je bio prije proširenja te je stoga i njega potrebno proširiti. Proširimo datotečni sustav na particiji, do maksimalne moguće veličine: na novu veličinu particije (kolika god ona bila):

```
xfs_growfs -d /dev/sdb1
```

Sada ponovno pogledajmo koliko blokova zauzima novi, prošireni datotečni sustav:

```
xfs_growfs -n /dev/sdb1
```

- ✓ Ponovno obratite pažnju na statistiku `blocks=` odnosno broj blokova koje datotečni sustav sada zauzima.

**Izvori informacija: (312),(313),(314),(315),(1008),(K-14)**,**`man dumpe2fs`**,**`man fsck`**,**`man xfs_repair`**,**`man fdisk`**,**`man touch`**,**`man lsblk`**,**`man umount`**,**`man xfs_db`**,**`man xfs_growfs`**,**`man 5 xfs`**,**`man 5 ext4`**.

### 13.6.2.3. Universally unique identifier (UUID) oznake

**Napredna cjelina!**
Univerzalni jedinstveni identifikator (**UUID**) je 128-bitna oznaka koja se koristi za informacije u računalnim sustavima. Osim naziva **UUID** također se koristi izraz globalno jedinstveni identifikator odnosno **GUID**. Kada se generiraju prema standardnim metodama, UUID-i su jedinstveni. UUID-e standardizira *Open Software Foundation* (OSF**)** kao dio distribuiranog računalnog okruženja (**DCE**). UUID-i su dokumentirani kao dio **ISO/IEC 11578:1996** i nedavno u **ITU-T Rec. X.667 | ISO/IEC 9834-8:2005**. Zatim je *Internet Engineering Task Force* (**IETF**) objavio **RFC 4122** tehničkim ekvivalentom **ITU-T Rec. X.667 | ISO/IEC 9834-8** prema kojem se danas i koristi.

**Format**
U tekstualnom prikazu, 16 okteta UUID-a predstavljeno je kao 32 heksadecimalnih znamenki, prikazanih u pet grupa odvojenih crticama, u obliku 8-4-4-4-12 za ukupno 36 znakova (32 heksadecimalna znaka i 4 crtice). Na primjer:
`123e4567-e89b-12d3-a456-426614174000`
`xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx`
Četiri bita znamenke **M** su inačica UUID-a, a 1 do 3 najznačajnija bita znamenke **N** kodiraju varijantu UUID-a.
Naime postoji nekoliko definiranih varijanti UUID-a, koje se koriste za različite namjene, pa se tako UUID-i kao jedinstveni identifikatori to jest kao jedinstveni ključevi, koriste i u bazama podataka (pr. **MySQL**, **PostgreSQL**, **Oracle**, …).
Nadalje, **UUID** se može koristiti i kao jedinstveni identifikator mrežnih sučelja, primjerice za mrežno sučelje **`enp0s3`**.
Pogledajmo kako kreirati **UUID** za ovo mrežno sučelje pomoću naredbe `uuidgen`:

```
uuidgen enp0s3
89f698ae-8a71-42d1-99a4-d7b48e013987
```

Sada dobiveni **UUID** možemo postaviti u varijablu `UUID=` u *ifcfg* datoteku: `/etc/sysconfig/network-scripts/ifcfg-enp0s3`.

Međutim **UUID** je jedinstveni identifikator koji se najčešće koristi na particijama diskova za jedinstvenu identifikaciju particija u operativnim sustavima Linux. UUID je stoga svojstvo same particije diska. Primjerice, ako premjestite tvrdi disk koji sadrži particije (i datotečne sustave na njima), na drugo Linux računalo, particije će imati isti **UUID** kao prije.
Što se tiče diskova i particija, razlikujemo nekoliko jedinstvenih identifikatora:

- `PTUUID` – je **UUID** identifikator same particijske tablice, odnosno jedinstveni identifikator za cijeli disk dodijeljen u vrijeme kada je disk particioniran. To je ekvivalent diskovnog potpisa na **MBR** particioniranim diskovima, ali s više bitova i standardiziranim postupkom za njegovo generiranje.
- `PARTUUID` – je također **UUID** identifikator na razini particijske tablice za sve particije na **GPT** particioniranim diskovima. Budući da se dohvaća iz particijske tablice, može mu se pristupiti bez ikakvih pretpostavki o stvarnom sadržaju particije. Ako je particija šifrirana pomoću neke nepoznate metode šifriranja, ovo bi mogao biti jedini dostupni jedinstveni identifikator za tu particiju.
- `UUID` – je **UUID** identifikator na razini datotečnog sustava, koji se dohvaća iz metapodataka datotečnog sustava unutar particije. Može se pročitati samo ako je vrsta datotečnog sustava poznata i čitljiva. Dakle ova oznaka postoji samo i isključivo ako je particija diska formatirana s nekim datotečnim sustavom poput: **ext2/3/4**, **XFS**, … .

Pogledajmo jedan disk `/dev/sdd` s naredbom `lsblk`:

```
lsblk -f /dev/sdd
NAME FSTYPE LABEL UUID MOUNTPOINT
sdd
```

← Vidimo da nema particija niti ikakvih ***UUID*** oznaka, što možemo vidjeti i s naredbom `blkid`:

```
blkid /dev/sdd

```

← I ovdje nema nikakvih podataka.

Sada ćemo na tom disku napraviti particiju (proceduru nećemo objašnjavati).
Novu particiju `/dev/sdd1` pogledajmo s prethodnim naredbama:

```
blkid /dev/sdd1
/dev/sdd1: PARTUUID="1c590704-01"

lsblk -f /dev/sdd1
NAME   FSTYPE LABEL UUID MOUNTPOINT
sdd
└─sdd1
```

← Jasno vidimo da se pojavio `PTUUID` jer je particija kreirana kao **MBR** particija i dobila je svoju jedinstvenu oznaku.

Međutim ona i dalje nema svoju **UUID** oznaku jer ju još nismo formatirali s datotečnim sustavom, što ćemo sada učiniti s **ext4**.

```
mkfs.ext4 /dev/sdd1

mke2fs 1.45.6 (20-Mar-2020)
Creating filesystem with 2621184 4k blocks and 655360 inodes
Filesystem UUID: d8ffc6cd-0134-4aa9-95e7-fa24f5adcc5d
Superblock backups stored on blocks: 32768,98304,163840,229376,294912,819200,884736
```

Sada kada smo formatirali particiju `/dev/sdd1` s Linuxovim datotečnim sustavom **ext4**, pogledajmo sve ponovno:

```
blkid /dev/sdd1
/dev/sdd1: UUID="d8ffc6cd-0134-4aa9-95e7-fa24f5adcc5d" BLOCK_SIZE="4096" TYPE="ext4"
PARTUUID="1c590704-01"
lsblk -f /dev/sdd1
NAME FSTYPE LABEL UUID                                 MOUNTPOINT
sdd1 ext4         d8ffc6cd-0134-4aa9-95e7-fa24f5adcc5d
```

← Iz ispisa obije naredbe je sada vidljivo sljedeće:

- Vidimo `PARTUUID` oznaku same particijske tablice odnosno cijelog diska (`PARTUUID="1c590704-01"`).
- Ali sada vidimo i **UUID** oznaku (`d8ffc6cd-0134-4aa9-95e7-fa24f5adcc5d`) datotečnog sustava na `/dev/sdd1` particiji.

Najčešća primjena **UUID** oznake datotečnog sustava je tijekom montiranja datotečnog sustava u neki direktorij. Kreirajmo privremeni direktorij i montirajmo novi datotečni sustav u njega.

```
mkdir /TEST
mount /dev/sdd1 /TEST
```

Kod ovakvog montiranja (engl. *mount*) nismo koristili **UUID** oznaku već oznaku particije diska (`/dev/sdd1`).

Pogledajmo što smo sada dobili:

```
lsblk -f /dev/sdd
NAME   FSTYPE LABEL UUID                                 MOUNTPOINT
sdd
└─sdd1 ext4         d8ffc6cd-0134-4aa9-95e7-fa24f5adcc5d /TEST
```

← Vidimo našu novu particiju (`/dev/sdd1`) i njen **UUID** broj te da je ona montirana u direktorij `/TEST`.

Na sve više novijih sustava se tijekom montiranja datotečnog sustava u direktorij umjesto oznake diska (`/dev/XYZ`) koristi **UUID** oznaka, jer ona ne ovisi kako je disk spojen, što se koristi kod `/dev/XYZ` oznaka.
To je također moguće definirati u datoteci `/etc/fstab` koja služi za montiranje datotečnih sustava tijekom pokretanja računala. Pogledajmo nekoliko unosa u ovoj datoteci, za koje se koriste UUID oznake:

```
UUID=3ef71f37-ad03-4463-818a-6d69a73d9333 /          xfs     defaults        0 0
UUID=535c4997-daf2-444c-a2e0-2bce4a2794be none       swap    defaults        0 0
```

**Što ako imamo potrebu promijeniti UUID oznaku već formatiranog datotečnog sustava?**
I to je moguće, a sama naredba ovisi o datotečnom sustavu, što postižemo na sljedeće načine.
**1.** Potrebno je *odmontirati* datotečni sustav od direktorija (mape).
**2.** Ponekada je potom potrebno napraviti provjeru konzistencije podataka datotečnog sustava:

Za **ext2/3/4** datotečni sustav:

```
e2fsc/k -f /dev/sdd1
```

Odnosno za **XFS** datotečni sustav:

```
xfs_repair -L /dev/sdd1
```

**Slijedi promjena UUID-a!**
Za **ext2/3/4** datotečne sustave. Primjerice za `/dev/sdd1` particiju i novi **UUID** (*promijenili smo zadnje dvije oznake u* **dd**):

```
tune2fs /dev/sdd1 -U d8ffc6cd-0134-4aa9-95e7-fa24f5adccdd
```

Za **XFS** datotečni sustav. Primjerice za `/dev/sdd1` particiju i novi **UUID** (*promijenili smo zadnje dvije oznake u* **dd**):

```
xfs_admin -U d8ffc6cd-0134-4aa9-95e7-fa24f5adccdd /dev/sdd1
```



Za više detalja i primjena **UUID**-a, pogledajte poglavlja:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**
**13.4. Logička shema diska.**
**13.6.2. Formatiranje particije.**
**13.6.2.2. Provjera konzistencije podataka datotečnog sustava na particiji diska.**
**13.8. Što je datoteka `/etc/fstab`.**

U trenutku particioniranja diska, unutar **MBR** zapisa na prvom sektoru diska, zapisuje se i jedinstveni identifikator diska. Ovaj identifikator može se primjerice za disk `/dev/sda` vidjeti s naredbom `gdisk`:

```
gdisk -l /dev/sda |  grep "Disk identifier"
```

**Izvori informacija: (1458), `man lsblk`,`man blkid`,`man 5 fstab`,`man mount`,`man xfs_repair`, `man xfs_admin`,`man uuidgen`,`man gdisk`,`RFC 4122`.**

# 13.7. Kako se montiraju particije u direktorije i zašto

U Linux svijetu i sâma particija diska je posebna datoteka u `/dev/` direktoriju, kojoj u konačnici želimo pristupiti preko datotečnog sustava koji je prethodno i kreiran na njoj, a koji je zadužen za rad s datotekama i direktorijima. Dakle svaka particija diska pod Linuxom se predstavlja kao posebna datoteka kojoj normalno možemo pristupiti i koristiti ju, kao da se radi o bilo kojoj drugoj (običnoj) datoteci.

Međutim svaka se particija prethodno formatirana s određenim datotečnim sustavom, mora povezati s određenom točkom odnosno konkretno s direktorijem u postojećem stablu direktorija (mapa) našeg Linux sustava.



***Povezivanje formatirane particije diska s postojećim direktorijem (mapom) se naziva montiranje ili mountanje.***

U prijašnjem poglavlju smo drugi SATA tvrdi disk (*`/dev/sdb`*) odnosno njegovu prvu particiju (`/dev/sdb1`) formatirali s **ext4** datotečnim sustavom. Nju ćemo sada povezati s novim direktorijem koji smo nazvali `/hdd-novi`
Dakle particija `/dev/sdb1` koju smo nakon kreiranja i formatirali, *montirati* (*mountati*) će se u novi direktorij:`/hdd-novi`.
Kod *montiranja* je potrebno znati i koji je datotečni sustav u pitanju, jer se i on mora dodati kao parametar u trenutku montiranja.

Važno je razumjeti da se prilikom montiranja datotečnog sustava na pariciji može navesti:

- Standardna oznaka particije; primjerice: `/dev/sdb1`.
- **UUID** oznaka particije (pogledajte prethodno poglavlje).

Prilikom montiranja, datotečne sustave **`ext3`** i **`ext4`** nije potrebno navoditi jer se podrazumijevaju.
Za sve ostale datotečne sustave se mora navesti vrsta datotečnog sustava. Takav slučaj je primjerice ***`XFS`*** datotečni sustav.

## Provjera diskova i particija

Prije nego krenemo dalje, potrebno je provjeriti status diskova po pitanju particija i pripadajućih datotečnih sustava.
Stoga pokrenimo naredbu `lsblk`:

```
lsblk -f
NAME   FSTYPE FSVER LABEL UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
sda
├─sda1 xfs                14548a88-c873-4825-b020-b8d62e3d3a5b   16.1G    15% /
└─sda2 swap   1           39650e0e-88a5-42eb-9a74-64524d88f18b                [SWAP]
sdb
└─sdb1 ext4   1.0         0242a7a9-c02a-4e8b-83ad-5a2587bfc7af
```

Vezano za disk `/dev/sdb` odnosno njegovu prvu particiju: `/dev/sdb1`, vidimo da je ona formatirana s datotečnim sustavom `ext4`.
Dodatno vidimo i **UUID** oznaku datotečnog sustava na ovoj particiji (`0242a7a9-c02a-4e8b-83ad-5a2587bfc7af`).

Istu **UUID** oznaku na particiji, možemo vidjeti i s naredbom `blkid`:

```
blkid /dev/sdb1
/dev/sdb1: UUID="0242a7a9-c02a-4e8b-83ad-5a2587bfc7af" TYPE="ext4" PARTUUID="3222c50c-01"
```

## Ručno montiranje particije

Ručno montiranje možemo odraditi na dva načina.
Prvo moramo kreirati odredišni direktorij, primjerice `hdd-novi` pomoću naredbe `mkdir` na sljedeći način:

```
mkdir /hdd-novi
```

Potom možemo napraviti *montiranje* pomoću standardne oznake particije (konkretno `/dev/sdb1`).
To ćemo napraviti pomoću naredbe `mount` na sljedeći način:

```
mount /dev/sdb1 /hdd-novi
```

Da je ovo bio **XFS** datotečni sustav, naredba za ručno montiranje bi izgledala ovako:

```
mount -t xfs /dev/sdb1 /hdd-novi
```

Drugi način montiranja bi bio pomoću **UUID** oznake datotečnog sustava na već navedenoj particiji diska:

```
mount UUID="0242a7a9-c02a-4e8b-83ad-5a2587bfc7af" /hdd-novi/
```

Prilikom montiranja, s prekidačem `-t` se odabire koji datotečni sustav je u pitanju. Nakon tog trenutka sadržaj direktorija: `/hdd-novi` postaje sadržaj formatirane particije prvog SATA tvrdog diska i to njegove prve particije, konkretno `/dev/sdb1`, sve do novog restarta računala.

**Navedeno montiranje (*mountanje*) koje smo napravili nije permanentno (trajno)!**

Objasnit ćemo i automatski način trajnog montiranja, koji osigurava rad i nakon restarta računala. Svi *montirani* datotečni sustavi su vidljivi pozivanjem naredbe `mount` bez parametara, odnosno na sljedeći način (skratili smo ispis):

```
mount

/dev/mapper/VolGroup-lv_root on /          type ext4    (rw)
/dev/sda1                    on /boot      type ext4    (rw)
/dev/sdb1                    on /hdd-novi  type ext4  (rw,relatime)
```

Puno ljepši ispis možemo dobiti s naredbom `findmnt` stoga pogledajmo njen (skraćeni) ispis.

```
findmnt

TARGET                                SOURCE      FSTYPE   OPTIONS
/                                     /dev/sda1   ext4     rw,relatime,seclabel,at
├─/hdd-novi                           /dev/sdb1   ext4     rw,relatime
```

Nas zanima `/dev/sdb1` koji izgleda u redu jer vidimo kako se uredno *montirao* odnosno kako su se formatirana particija: `/dev/sdb1` i direktorij: `/hdd-novi` povezali. Pogledajmo i veličine particija koje su *montirane*, pomoću naredbe `df`:

```
df -h

Filesystem   Size   Used   Avail   Use%   Mounted on
-----------------------------------------------------
/dev/sdb1     2G      1M     2G       1%   /hdd-novi
```

Dakle `/dev/sdb1` je veličine 2GB i *montiran* je u direktorij `/hdd-novi`.

## Demontiranje particije

U ovom primjeru ćemo demontirati (*odmount*ati) datotečni sustav na particiji `/dev/sdb1` koja je povezana s direktorijem `/hdd-novi`. To ćemo postići s naredbom `umount`, i to stoga kako bismo kasnije provjerili hoće li se ponovno *montirati* tijekom svakog slijedećeg pokretanja sustava.
Navedeno demontiranje odrađujemo na sljedeći način:

```
umount /hdd-novi
```

U slučaju kada prilikom pokušaja demontiranja datotečnog sustava dobijemo poruku poput:

```
umount: /hdd-novi: target is busy.
```

Tada je potrebno izaći iz trenutnog direktorija (pr. u `/`) ili pronaći koji proces drži otvorene datoteke u njemu i zatvoriti ga:

```
lsof /hdd-novi
```

← S navedenom naredbom ćemo pronaći koji proces (pokrenuti program) eventualno drži otvorenu neku datoteku unutar direktorija u koji je montiran naš datotečni sustav.



**Važno je razumjeti da od trenutka kada smo montirali datotečni sustav i on je u upotrebi, pristup svim datotekama i direktorijima na njemu, kao i pristup sadržaju datoteka se sprema u (RAM) međumemoriju.**
**To znači da se svako zapisivanje ili čitanje podataka s te particije prvo zapisuje u posbenu međumemoriju koja se naziva *Page cache*. Tek kasnije se sve to stvarno i zapisuje na površinu diska. Ova optimizacija drastično ubrzava pristup podacima, ali u slučaju nasilnog demontiranja, možemo izgubiti podatke koji su ostali u RAM memoriji, a nisu se stigli zaopisati na disk.**

**Stoga je vrlo važno uredno odraditi demontiranje prije odspajanja diska, posebno za USB i druge prijenosne diskove!**



Za detalje o ***Page cache*** (RAM) međumemoriji za diskove, pogledajte poglavlje:
**14.1.2. Optimizacija *Page Cache* sloja.**

**Slijedi korak trajnog montiranja particije u direktorji**.
Sada na kraj datoteke `/etc/fstab` dodajmo sljedeći redak (primjer klasinčnog montiranja bez **UUID** oznake):

```
/dev/sdb1 /hdd-novi ext4 errors=remount-ro 0 1
```

Nakon prvog restarta cijelog poslužitelja/računala, *montiranje* nove particije će biti automatski odrađeno.
To smo mogli napraviti i ručno odnosno odmah da ne moramo restartati cijelo računalo, sa sljedećom naredbom:

```
mount -a
```

Neki od korisnih prekidača naredbe `mount` su:

- `-a` – za montiranje svih točki montiranja navedenih u datoteci `/etc/fstab`. ← **Pogl. poglavlje: 13.8.**
- `-r` – naznačava da će datotečni sustav koji se montira bit zaštićen od pisanja (engl. *read only*).
- `-U` – naznačava da želimo koristiti particiju sa željenom **UUID** oznakom.
- `-t` – nakon njega se definira datotečni sustav na particiji (pr.: *ext2, ext3, ext4, xfs, nfs, ntfs4, sysfs, tmpfs, …* ).

## Slijedi napredna cjelina!

**Dodatne opcije i mogućnosti tijekom montiranja.**
Moguće je montirati i postojeći montirani direktorij na drugi direktorij. To radimo pomoću naredbe `mount` s parametrom `--bind`. O ovom takozvanom ***bind mountu*** možemo razmišljati kao o *aliasu* na izvornu točku montiranja, što se često koristi kod izoliranih prostora Linuxa (Linux namespaces). Međutim tada eventualne dodatne točke montiranja unutar postojećeg izvorišnog direktorija neće biti naslijeđene u drugu točku montiranja, a ako i to želimo, možemo koristiti opciju: `--rbind`.
Nadalje, od kernela 2.6.15 moguće je označiti točke montiranja i podmontiranja kao dijeljene (`--make-shared`), privatne (`--make-private`), podređene (`--make-slave`) ili nepovezane (`--make-unbindable`). Dijeljeno montiranje pruža mogućnost stvaranja zrcala tog montiranja tako da se montiranja i demontiranja unutar bilo kojeg zrcala šire na drugo zrcalno montiranje. Podređeno montiranje prima propagaciju od svog vlasnika, ali ne i obrnuto. Dok privatno montiranje nema mogućnosti širenja. Nepovezano montiranje je privatno i ne može se klonirati operacijom vezanja (*bind*). Za sve navedene posebne točke montiranja, mogući su i rekurzivni načini njihovog montiranja, i to redom: `--make-rshared`, `--make-rprivate`, `--make-rslave` i `--make-runbindable`.


Druge važnije opcije kod *montiranja* smo opisali u poglavlju: **13.8. Što je datoteka /etc/fstab.**

Dodatno, svaki *montirani* datotečni sustav ili mrežni disk, vidljiv je i u posebnoj particiji odnosno datotečnom sustavu `/proc` i to u dvije datoteke:

- `/proc/mounts` – sadržaj ove datoteka prikazuje ono što i vidimo pozivanjem čiste naredbe `mount`. Dakle jednostavan prikaz svih *montiranih* particija se zapisuje ovdje.
- `/proc/self/mountinfo` - ovdje se vidi znatno više detalja, koje ćemo sada i objasniti, u primjeru s jednim redom unosa. Ova datoteka se inače povezuje s trenutnim aktivnim, radnim procesom.

Pogledajmo jedan unos u datoteci `/proc/self/mountinfo`, on izgleda ovako: (ovo je samo primjer)

```
36 35  8:17  /  /mnt2 rw,noatime master:1 - ext4 /dev/sdb1 rw,errors=continue
 |  |   |    |    |        |        |     |   |      |              |
(1)(2) (3)  (4)  (5)      (6)      (7)   (8) (9)   (10)           (11)
```

Objašnjenje stupaca unutar ove datoteke slijedi:

- (1) - *mount ID* - jedinstveni identifikator *montiranja*, koji se može ponovno iskoristiti nakon demontiranja.
- (2) - *parent ID* - identifikator vršnog procesa koji je napravio montiranje (*mount).*
- (3) - ***major****:minor* - oznake uređaja koji je *montiran;* pogledajte i poglavlje: **11.1.2.1 Uređaji (devices) detaljnije.**
- (4) - *root* - odnosno vršni direktorij datotečnog sustava u koji se *montira.*
- (5) - *mount point* je točka *montiranja* u direktorij (mapu).
- (6) - *mount options* odnosno opcije i parametri s kojima je napravljeno *montiranje.*
- (7) - opcionalna polja (ovo polje može biti prazno ako nema dodatnih opcija). Opcije mogu biti:
  - `shared:X` - *mount* je dijeljen (*shared*) u *peer* grupi **X.**
  - `master:X` - *mount* je (*slave/pripadnik*) *peer* grupi **X.**
  - `propagate_from:X` - *mount* je (*slave peer grupi X*) te prima propagacije iz *peer* grupe **X.**
  - `unbindable` - *mount* je (*unbindable*) - ne poveziv.
- (8) - ovdje je znak minus (-) koji je separator između opcija (7) i polja devet (9).
- (9) - je *filesystem type* polje koje označava datotečni sustav koji je *montiran.*
- (10) - je *source* odnosno izvorna particija ili disk koji je *montiran* - pr. */dev/sdb1.*
- (11) - ovo su takozvane *super options* - opcije za datotečni sustav - za blok uređaj koji je *montiran*

Pogledajmo (`/proc/self/mountinfo`) kako izgleda jedan stvarni unos za naš drugi SATA disk i to njegovu prvu particiju `/dev/sdb1` formatiranu s ***ext4*** datotečnim sustavom, a koju smo *montirali* u direktorij `/hdd-novi`:

```
25 21 8:17 / /hdd-novi rw,relatime - ext4 /dev/sdb1 rw,barrier=1,data=ordered
```

**Izvori informacija: (1476),(K-12), `man mount`, `man fstab`,`man findmnt`,`man umount`,`man 5 fstab`.**

## 13.7.1. Kako *montir*ati CD/DVD-ROM ?.

CD/DVD-ROM disk ubačen u CD/DVD-ROM uređaj se *montira* praktično isto kao i tvrdi disk, uz razliku da kada nam više ne treba, moramo napraviti i *demontiranje* jer u protivnom nećemo moći izvaditi CD/DVD medij iz uređaja. Pogledajmo primjer u kojem je CD/DVD-ROM uređaj vidljiv kao uređaj: `/dev/cdrom` i ako ga želimo *montirati* u direktorij `/cdrom`, napravit ćemo sljedeće, s naredbom `mount`:

```
mount -t iso9660 /dev/cdrom /cdrom
```

Nakon završetka rada sa CD/DVD medijem i kada ga želimo izvaditi, moramo izaći iz direktorija koji predstavlja sâm uređaj odnosno konkretno `/cdrom` te moramo napraviti demontiranje CD/DVD diska, s naredbom `umount` ovako:

```
umount /dev/cdrom
```

**Izvori informacija: `man mount`,`man umount`.**

## 13.7.2. Kako *montirati* CD/DVD-ROM iz ISO datoteke

U određenim slučajevima, kada na lokalnom ili mrežnom disku imamo dostupnu **ISO** sliku (*image*) datoteku, nekog CD-ROM ili DVD-ROM diska, moguće ju je *montirati* u neki lokalni direktorij kao da se radi o normalnom CD/DVD-ROM disku ubačenom u CD/DVD-ROM uređaj. Kreirajmo prvo lokalni direktorij: `/mnt/CD-DVD` u koji ćemo ju *montirati*:

```
mkdir -p /mnt/CD-DVD
```

Ako se *ISO (slika)* datoteka nalazi na lokaciji `/ISO/Linux.iso` tada napravimo:

```
mount -o loop /ISO/Linux.iso /mnt/CD-DVD
```

Napomena: naredbi `mount` smo dodali opciju (`-o loop`) kako bi koristila poseban takozvani ***loop*** uređaj, a koji Linux koristi upravo kako bi se datotekama koje se u ovom slučaju nalaze unutar *ISO* slike *(image)* CD/DVD-ROM diska, moglo pristupati kao da se nalaze na stvarnom *blok uređaju* odnosno disku ili CD/DVD-ROM-u.
Od sada, u direktoriju: `/mnt/CD-DVD` će se nalaziti sve datoteke s gornje *ISO image* datoteke.
Ako u bilo kojem trenutku više ne trebamo ovaj CD/DVD disk, moramo se nalaziti u nekom direktoriju izvan direktorija koji predstavlja taj CD/DVD-ROM disk. U našem slučaju izađimo iz direktorija `/mnt/CD-DVD` primjerice u *root* direktorij (`/`):

```
cd /
```

Sada možemo *demontirati* CD/DVD disk *image* s naredbom `umount` na sljedeći način:

```
umount /mnt/CD-DVD
```



Za više informacija o montiranim datotečnim sustavima, pogledajte i poglavlje:
**13.10.1. Naredba *lsblk*.**

**Izvori informacija: (K-12),** `man mount`, `man umount`, `man lsblk`, `man blkid`, `man findmnt`, `man 4 loop`.

## 13.7.3. Ograničenje zauzeća prostora na disku (*quota*)

**Slijedi napredna cjelina!**
Na Linux sustavima moguće je ograničiti upotrebu određene particije na disku što se tiče zauzeća prostora. Dakle moguće je za određene korisnike ili korisničke grupe postaviti ograničenjem koliko prostora na disku smiju iskoristiti. Ova opcija standardno nije postavljena pa niti jedan korisnik niti jedna korisnička grupa nemaju ograničenje na upotrebu prostora na particijama diska. Važno je razumjeti da se ograničenje zauzeća prostora na disku odnosno *kvota* (Engl. *Quota*) definira tijekom montiranja (formatirane) particije diska, a ovisi o datotečnom sustavu, odnosno njegovoj podršci za *kvote*. Dakle ona se mora aktivirati odnosno definirati u datoteci `/etc/fstab` (o njoj u sljedećem poglavlju). Zamislimo slučaj u kojem imamo dodan novi tvrdi disk (`/dev/sdc`) na kojem smo prethodno kreirali jednu particiju (`/dev/sdc1`) te ju formatirali s ***EXT4*** datotečnim sustavom.

Potom smo kreirali direktorij (mapu) `/DATA` u koju ćemo ju montirati. Međutim, nećemo ju montirati na standardan (klasičan) način, već ćemo za određene korisnike i korisničke grupe postaviti ograničenja, kako oni ne bi mogli prepuniti tu particiju. Ova particija je veličine 100GB i na nju ćemo uglavnom snimati radne podatke (datoteke) ali ne želimo da na nju određeni korisnici ili korisničke grupe mogu snimiti više od 90GB, kako bi zaštitili ovu particiju od prepunjavanja.

Za ovakvo montiranje imamo dvije opcije: ručno montiranje s posebnim parametrima ili automatski tijekom svakog podizanja sustava, dodavanjem unosa u `/etc/fstab`, što ćemo upravo napraviti. Dodajmo sljedeći unos u datoteku `/etc/fstab`:

```
/dev/sdc1       /DATA  ext4    defaults,usrquota,grpquota 0 0
```

Nakon što smo dodali ovaj unos u datoteku `/etc/fstab`; kako bi se nova particija montirala, trebamo pokrenuti naredbu:

```
mount -a
```

U slučaju kada želimo isprobati rad *kvote* na navedenoj particiji odnosno točki montiranja, bez trajnog upisivanja *kvote* u datoteku `/etc/fstab`, to možemo napraviti na sljedeći način (ova promjena je privremena, do sljedećeg restarta računala):

```
mount -o remount,usrquota,grpquota /DATA
```

I sada je naša particija montirana s uključenim ograničenjima odnosno disk kvotama. Pogledajmo kako to izgleda:

```
mount | grep DATA
/dev/sdc1 on /DATA type ext4 (rw,relatime,quota,usrquota,grpquota)
```

Dakle vidimo kako su dodane *kvote* (primarno nas zanimaju: `usrquota` i `grpquota`). Prvo instalirajmo sve potrebno:

```
yum -y install quota quotatool
```

Sada ćemo pokrenuti proces upotrebe disk *kvote* za korisnike (`u`) i za korisničke grupe (`g`) na sljedeći način (`quotacheck`):

```
quotacheck -cugm /DATA
```

I potom moramo i aktivirati disk *kvotu*, što ćemo postići s naredbom `quotaon`:

```
quotaon -v /DATA
/dev/sdc1 [/DATA]: group quotas turned on
/dev/sdc1 [/DATA]: user quotas turned on
```

Iz ispisa vidimo da je disk kvota sada aktivirana na navedenoj particiji (`/dev/sdc1`). Ako ju ipak želimo isključiti:

```
quotaoff -v /DATA
```

Nakon ovih operacija, automatski su kreirane dvije posebne datoteke vezane za kvotu korisnika (`/DATA/aquota.user`) te za *kvotu* korisničkih grupa (`/DATA/aquota.group`). Ove datoteke su standardno skrivene običnim korisnicima.

## Rad s disk kvotama

Za rad s disk *kvotama* koristi se naredba `edquota`, uz dvije napomene; disk *kvote* se mogu definirati s dvije veličine:

- Disk *kvote* se definiraju u blokovima na disku (a ne u MB). Pogledajte poglavlje: **4.5. Datotečni sustav detaljnije**.
- I/ili se disk *kvote* definiraju s brojem ***inode*** unosa. Pogledajte poglavlje: ***13.2. Geometrija diskova***.

**Izračun disk blokova**

Prvo pogledajmo kolika je veličina bloka (u bajtima) na našoj particiji s kojom radimo, pomoću naredbe `blockdev`:

```
blockdev --getbsz /dev/sdc1
4096
```

Vidimo da je veličina bloka (tzv. disk klaster) 4kB (što je uglavnom standardno).
Sada izračunajmo koliko nam disk blokova treba za primjerice 1GB prostora:

$$1GB\ prostora = 1024\ MB = 1024\ x\ 1024\ kB, prema\ tome\ 1GB\ prostora = \frac{1024\ x\ 1024}{4\ (veličina\ bloka)} = 262.144\ \ blokova$$

Dakle za alociranje **1GB** prostora na disku, sa 4kB blokovima, potrebno je **262.144** blokova na disku.

**Izračun broja inode unosa**

Broj **inode** unosa na datotečnom sustavu odnosno na formatiranoj i montiranoj particiji, definira maksimalan broj datoteka ili direktorija (mapa) koje možete stvoriti u datotečnom sustavu. Naime, ako određeni korisnik ima ograničenje (kvotu) na 1.000 inode unosa, to znači da on može stvoriti samo 1.000 datoteka ili direktorija (mapa) na toj particiji diska.



*Važno je razumjeti da čak i ako je ukupna veličina tih 1.000 datoteka ili direktorija manja od broja blokova koje korisnik može koristiti (definirano u kvoti), on i dalje neće moći stvoriti nove datoteke ili direktorije.*

Stavljajući ograničenje kvote na broj ***inode*** unosa direktno na razini datotečnog sustava ograničavamo korisnika ili korisničku grupu na maksimalan broj datoteka ili direktorija koje oni mogu stvoriti, pa stoga treba biti vrlo oprezan pri odabiru prave vrijednosti. U našem primjeru, mi ne ograničavamo *root* odnosno korijensku sistemsku particiju (`/`) već sâmo montiranu (`/DATA`) particiju koja će se koristiti za pohranu datoteka koje kreiraju određeni korisnici odnosno korisnici koji pripadaju određenoj korisničkoj grupi. Moguće je postaviti ograničenja (kvote): sâmo na broj blokova ili sâmo na broj ***inode*** unosa ili na oboje; odnosno i na blokove i na ***inode*** unose. Mi ćemo u primjerima koji slijede postaviti oba ograničenja istovremeno.

**Postavljanje disk kvote za određenog korisnika.**

Sada ćemo postaviti disk kvotu za korisnika `pero`; tako da mu ograničavamo prostor na 1 GB (262.144 blokova od 4kB) te na 1.000 ***inode*** unosa. To postižemo s naredbom `edquota`, na sljedeći način:

```
edquota -u pero
```

Zatim će nam se otvoriti poseban uređivač teksta s kojim trebamo napraviti sljedeće promjene:

```
Disk quotas for user pero (uid 1003):
  Filesystem        blocks       soft        hard     inodes      soft      hard
  /dev/sdc1           0           0           0         0           0         0
```

Ovdje imamo dvije cjeline: ***ograničenje blokova*** ***ograničenje inode unosa***

Dakle i za ograničenje blokova i ograničenje ***inode*** unosa, ćemo postaviti svoje željene krajnje vrijednosti pod `hard` što označava krajnje ograničenje. Moguće je definirati i tzv. meko ograničenje (`soft`), ali mi njega nećemo koristiti.

Dakle mi ćemo imati postavljeno sljedeće ograničenje za navedenog korisnika `pero`:

```
Filesystem     blocks        soft        hard         inodes      soft       hard
  /dev/sdc1             0           0      262144              0         0       1000
```

Iz ovog uređivača teksta (to je `vi` uređivač teksta), snimamo promjene i izlazimo s tipkom ***ESC*** pa znakom `:` te `wq!`

**Postavljanje disk kvote za korisničku grupu.**
Ograničenje na korisničku grupu radimo na isti način. Pošto imamo korisničku grupu `korisnici`, a na nju želimo postaviti veće ograničenje; 90GB za prostor na disku (23.592.960 blokova) te ograničenje na broj *inode* unosa na 100.000.

```
edquota -g korisnici
```

Te ćemo postaviti i snimiti sljedeće vrijednosti:

```
Disk quotas for group korisnici (gid 1102):
  Filesystem   blocks      soft        hard      inodes      soft      hard
  /dev/sdc1         0         0    23592960           0         0    100000
```



***Da bi kvota za korisničku grupu radila, definirana korisnička grupa mora biti primarna za određenog korisnika. U našem primjeru je to korisnik* `hrvoje`*, kojem primarna grupa mora biti grupa* `korisnici`.**

To možemo postići sa:

```
usermod -g korisnici hrvoje
```

Ako određena kvota ima meka ograničenja (Engl. *Soft)*, moguće je urediti razdoblje trajanja odnosno vremensko ograničenje koje se može premašiti (Engl. *Grace period*)), sa sljedećom naredbom:

```
edquota -t
```

Pojavit će nam se sličan okvir za unos vrijednosti kao iz prethodnog primjera, ali ovaj puta u danima vremena trajanja.
U bilo kojem trenutku, kada želimo vidjeti koja ograničenja disk kvote imamo postavljena, te koja ograničenja je dosegao određeni korisnik ili korisnička grupa, tada trebamo pokrenuti naredbu `repquota` na sljedeći način:

```
repquota -aug

*** Report for user quotas on device /dev/sdc1
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --      24       0       0              3     0     0
pero      --   53248       0  262144              2     0 10000

*** Report for group quotas on device /dev/sdc1
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
Group           used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --      16       0       0              1     0      0
pero      --   53248       0       0              2     0      0
korisnici --       8       0  23592960            2     0 100000
```

Ovdje prvo vidimo disk kvote prema korisnicima (ograničenja i ono što je trenutno u upotrebi), a ono što je dodatno važno je da za korisnike za koje nemamo definirana ograničenja; poput korisnika `root`, vodi se sâmo statistika o iskorištenju bez ograničenja. Zatim u donjem dijelu vidimo statistike prema korisničkim grupama (pr. `korisnici`).
S naredbom `quota` za trenutno logiranog korisnika možemo vidjeti disk *kvotu* (ako je postavljena za tog korisnika).

Kada se datotečni sustav ne uspije čisto demontirati (na primjer zbog pada sustava), potrebno je pokrenuti provjeru disk *kvota*. Međutim, provjeru disk *kvota* moguće je pokretati i redovito, čak i ako se sustav nije srušio. Preporuka bi bila da se particija prvo demontira te montira sâmo za čitanje (Engl. *Read only*), a zatim privremeno isključi *kvota* (s naredbom `quotaoff`) te nastavi s provjerom. Ipak moguće je sve to napraviti i bez toga, sa sljedećom naredbom (mada nije preporučeno):

```
quotacheck -vgumaf
```

Druga preporuka bi bila navedenu naredbu (`quotacheck`) staviti u skriptu koja bi se trebala staviti u ***crontab*** servis da se izvršava primjerice jednom tjedno, kako bi stanje *kvota* uvijek bilo konzistentno i uslijed neplaniranog restarta računala.

**Izvori informacija: (947),(948), (K-12), `man mount`, `man umount`,`man quota`,`man quotacheck`, `man edquota`,`man quotaon`,`man quotaoff`,`man fstab`.**

# 13.8. Što je datoteka `/etc/fstab`

**Slijedi napredno poglavlje (13.8.x)!**
Kako bismo mogli razumjeti automatsko *montiranje* odnosno „*mountanje*" particija, prvo moramo razumjeti čemu služi datoteka `/etc/fstab`. Datoteka `/etc/fstab` sadrži popis svih datotečnih sustava, njihovih lokacija i ostalih parametara koji su potrebni da bi se neki datotečni sustav *montirao* prilikom pokretanja operativnog sustava. Tijekom svakog pokretanja operativnog sustava, nakon inicijalizacije diskova, čita se datoteka `/etc/fstab` te se na osnovu njenih unosa prvo *montira* ***root*** odnosno korijenski direktorij (`/`) i to s datotečnog sustava i *particije* na kojoj se nalazi cijeli operativni sustav.

Nakon toga se *montiraju* i ostali datotečni sustavi prema redoslijedu kojim su navedeni u navedenoj konfiguracijskoj datoteci. Svi datotečni sustavi tek nakon što su *montirani* zapisuju se u datoteku `/etc/mtab` u sličnom formatu kao i datoteka `fstab`. Što se nalazi u `/etc/fstab` i kako to izgleda na primjeru korijenskog (`/`) direktorija, pogledajte u sljedećoj cjelini.



Za druge primjere upotrebe datoteke **`/etc/fstab`** pogledajte poglavlje o ***NFS*** mrežnom dijeljenom sustavu:
**25.8.7. NFS.**

Unutar datoteke `/etc/fstab` nalaze se definicije točki montiranja (*mount points*) i to u svakom (novom) redu za drugu točku montiranja odnosno *mount point,* prema sljedećoj sintaksi:

**Particija ili mrežni dijeljeni sustav** direktorij u koji se montira vrsta datotečnog sustava opcije dumps pass

Pri tome je:

- ***Particija ili mrežni dijeljeni sustav*** - ovdje se navodi *particija* ili mrežni dijeljeni sustav koji želimo koristiti. Pri tome *particija* osim u formatu ***/dev/XY*** (pr. `/dev/sda1`) može biti navedena kao ***UUID*** oznaka, koju možemo dobiti s naredbom `blkid`. Pogledajmo naredbu `blkid` u primjeni (obratite pažnju na ***UUID*** oznake particija), koje možemo vidjeti sa sljedećom naredbom:

```
blkid

/dev/sda1: UUID="3ef71f37-ad03-4463-818a-6d69a73d9333" BLOCK_SIZE="512" TYPE="xfs"
PARTUUID="301fcb98-01"
/dev/sdb1: UUID="813dc12c-af1c-4c42-92ba-2e1f68df60c1" BLOCK_SIZE="512" TYPE="xfs"
PARTUUID="1df9cb9e-01"
```

Naime ***UUID*** je jedinstveni identifikator koji se koristi na particijama za jedinstvenu identifikaciju particija u Linux operativnim sustavima. ***UUID*** je svojstvo same particije diska. Dakle, ako instalirate tvrdi disk koji već sadrži particije na drugom Linux računalu, particije će imati isti ***UUID*** broj odnosno oznaku kao i prije.



Vezano za ***UUID*** oznake diskova pogledajte i poglavlja:
**13.10.1.Naredba *lsblk***
**13.6.2.3. Universally unique identifier (UUID) oznake.**

- ***Direktorij u koji se montira*** - ovdje se navodi lokalni direktorij (mapa) u koji želimo montirati particiju ili mrežni sustav iz prvog stupca.
- ***Vrsta datotečnog sustava*** - ovdje se navodi o kojem se datotečnom ili mrežnom dijeljenom sustavu radi. To primjerice mogu biti: ***ext3***, ext4, ***xfs***, ***nfs***, ….
- ***Opcije*** - ovdje se navode opcije koje dajemo točki montiranja, a dobrim dijelom ovise o datotečnom sustavu. Ove opcije se mogu nîzati jedna za drugom, odvojene zarezom. Neke od opcija koje možemo navoditi ovdje su sljedeće:
    - `acl` – omogućava upotrebu Linux ACL-a dok `noacl` isključuje ACL.
    - `bind` – omogućava montiranje direktorija već montiranog datotečnog sustava u drugi (dodatni) direktorij, ali bez nasljeđivanja dodatnih točki montiranja unutar tog dodatnog direktorija. Ako želimo i nasljeđivanje, tada se može koristiti opcija `rbind`.
    - `defaults` – označava standardne opcije koje ovise o distribuciji Linuxa i vrsti particije. Ova opcija se preporuča u velikoj većini slučajeva a koristi opcije: `rw`, `suid`, `dev`, `exec`, `auto`, `nouser`, `async`, and `relatime`.
    - `discard` – slanje ***Discard/TRIM*** poruke blok uređaju kada se blokovi oslobode (za ***SSD/NVMe*** i ***LVM Thin***).
    - `rw` – označava kako će particija biti montirana kao sposobna za zapisivanje i čitanje (*read+write*).
    - `ro` – označava kako će particija biti montirana kao sposobna samo za čitanje (*read only*).
    - `sync` – označava da će particija biti montirana kao *sinkrona* za operacije prema diskovnom podsustavu (`sync`=sinkrono zapisivanje na disk). Obično se koristi *asinkrono* zbog brzine rada.

- `async` – označava da će particija biti montirana kao *asinkrona* za operacije prema diskovnom podsustavu (`async`=sinkrono zapisivanje na disk). Obično se koristi *asinkrono* zbog brzine rada.
- `suid` ili `nosuid` – označava hoće li se omogućiti (`suid`) ili onemogućiti (`nosuid`) postavljenje *SUID* bîtova ovlasti na montiranu particiju.
- `exec` ili `noexec` – označava: zabranjuje (`noexec`) li se pokretanje izvršnih datoteka s montirane particije ili se ne zabranjuje (`noexec`).
- `_netdev` – odnosi se na mrežni datotečni sustav (pr. **NFS**) tako da sustav prvo čeka na inicijalizaciju mreže, a tek potom će se montirati ovakav mrežni dijeljeni datotečni sustav.
- `user` ili `nouser` – označava tko može montirati datotečni sustav. Postavka (`nouser`) je standardna i daje ova prava samo *root* korisniku. Dok (`user`) označava da to može napraviti bilo koji korisnik, ali se tada podrazumijevaju sljedeća prava (`noexec`,`nosuid`,`nodev`) koja je, ako je potrebno, moguće ručno isključiti.
- `dev` ili `nodev` – naznačava sustavu da interpretira (`dev`) ili ne interpretira (`nodev`) karakter ili blok uređaje na datotečnom sustavu koji se montira.
- `noatime` – naznačava datotečnom sustavu koji će se montirati da ne zapisuje vrijeme pristupanja datotekama dakle vrijeme koje se standardno zapisuje kod otvaranja/čitanja datoteka na disku. Ova opcija može ubrzati datotečne poslužitelje jer se tijekom svakog pristupa bilo kojoj datoteci ne mora zapisivati vrijeme kada joj se pristupilo. Suprotna opcija je `atime` koja uključuje zapisivanje vremena pristupa.
- `nodiratime` – naznačava datotečnom sustavu koji će se montirati da ne zapisuje vrijeme pristupanja direktorijima dakle vrijeme koje se standardno zapisuje tijekom pristupanja direktorijima na disku. Ova opcija može ubrzati datotečne poslužitelje jer se kod svakog pristupa bilo kojem direktoriju ne mora zapisivati vrijeme kada mu se pristupilo. Suprotna opcija je `diratime` koja uključuje ovo zapisivanje.
- `dirsync` – naznačava datotečnom sustavu koji će se montirati da kod svih promjena na razina direktorija, promijenjene snimaju *sinkrono* na disk (*sync* metoda). To se odnosi na sve sistemske pozive koji uključuju sljedeće operacije: *creat(e), link, unlink, symlink, mkdir, rmdir, mknod i rename*.
- `relatime` – naznačava datotečnom sustavu koji će se montirati da zapisuje vrijeme pristupanja datotekama ili direktorijima tako da osvježavaju sâmo ***inode*** unose za *modify* i *change* vremena. To znači da se vrijeme pristupanja (*access time* unos) neće mijenjati svaki puta, već samo, ako je vrijeme modifikacije (*modify* ili *change*) novije u odnosu na *access* vrijeme. Ova opcija može ubrzati operacije zapisivanje ili čitanja, a ipak minimalno zadržava snimanje promjena o vremenima pristupa datotekama ili direktorijima. Suprotnost odnosno isključivanje je opcija `norelatime`.
- `quota`, `usrquota` ili `grpquota` – označava kvotu (ograničenje) prostora diska (standardno je isključena).

Ovisno o datotečnom sustavu, postoje i dodatne opcije, primjerice:
Za **ext2**:
- `acl` – za POSIX kompatibilne ACL liste ili `noacl` za isključivanje podrške za njih.
- `errors={continue|remount-ro|panic}` – definira ponašanja u slučaju grešaka na disku.
- `user_xattr` - za dodatne atribute ili `nouser_xattr` za isključivanje podrške za njih.
- …

Za **ext3** (podržava sve opcije kao i za **ext2** uz dodatne**)**:
- `data={journal|ordered}` – za odabir journal načina rada: `journal` – prvo se zapisuje *journal*, a onda podaci. `ordered` – (*zadano*) prvo se zapisuju podaci pa *journal* metapodaci, …
- `commit=`**`sec`** – definira se vrijeme svakih koliko sekundi će se sinkronizirati podaci i metapodaci (*journal*). Zadano je 5 sekundi.

Za **ext4** (podržava sve opcije kao i za **ext3** uz dodatne**)**:
- `barrier` – za korištenje sinkrone metode zapisivanja na disk odnosno `nobarrier` za asinkroni rad.
- `data=` – također definira metodu zapisivanaj na disk, opcije su:
  - `writeback` – za korištenje međumemorije za snimanje, s odgođenim zapisivanjem na disk.
  - `ordered` – za zapisivanje direktno na disk bez upotrebe međumemorije, …

- *dumps* - ovdje se navodi opcija za *backup* (više se ne koristi pa se uvijek postavlja na `0`).
- *pass* - ovdje se navodi mora li se periodički pokretati provjera ovog datotečnog sustava (`0`-ne, `1`-da).



Vezano za načine zapisivanja podatak na disk, pogledajte poglavlja:
**14.1.1. Sinkroni i asinkroni *I/O*.**
**27.1.2.3. Disk kontroleri.**

**Izvori informacija:(1627),(1628),(1629),(K-12),** **`man 5 fstab`**,**`man blkid`**,**`man mount`**.

## 13.8.1. root ( / ) particija

Ako je primjerice `/dev/sda1` sistemska particija diska na kojoj se nalazi cijeli operativni sustav Linux, a formatirana je s **ext4** datotečnim sustavom i ona se treba *montirati* u korijenski odnosno takozvani „*root*“ to jest `/` direktorij. Vršni direktorij `/` predstavlja korijen (eng. *root* = korijen) stabla direktorija cijelog operativnog sustava.

Pogledajmo kako to izgleda za ovaj korijenski (vršni) direktorij `/` odnosno pogledajmo ono što bi se trebalo nalaziti u datoteci `/etc/fstab` za njega:

| *Particija* | *Direktorij u koji se montira* | *Datotečni sustav* | *Opcije* | *Dump* | *Pass* |
|---|---|---|---|---|---|
| /dev/sda1 | / | ext4 | errors=remount-ro | 0 | 1 |

Ovaj unos u tablici, u datoteci: `/etc/fstab` izgleda kao sljedeći redak teksta:

```
/dev/sda1  /            ext4  errors=remount-ro     0     1
```

U opcijama koje vidimo, može se dodavati dosta parametara, a ovaj koji mi imamo (`errors=remount-ro`) znači kako će se u slučaju grešaka ova particija *montirati* sâmo kao *read-only* odnosno bez prava snimanja; samo čitanje. Stoga kako bi se mogla pokrenuti provjera tog datotečnog sustava, ako dođe do greške. Za provjeru i popravljanje integriteta podataka, postoje alati koji su specifični za točno određeni datotečni sustav (pogledajte poglavlje: **13.6.2.2.**). U našem slučaju to je ***ext4*** datotečni sustav pa se obično koristi alat *`fsck.ext4`* (kada dođe do procedure provjere integriteta podataka). Za sada je važno znati da kada se radi provjera ili popravljanje “podataka” na datotečnom sustavu, isti mora biti u stanju *read-only (samo čitanja)*. To je zato, kako se ne bi dogodilo da netko (ili neki program) u pozadini radi izmjene na podacima, jer one mogu dodatno pogoršati stanje.
Pogledajmo što znače i druge opcije:
***Dump*** označava staru metodu *backupa*, kod gašenja sustava. Pošto se više ne koristi, preporuka je da ostane na nula (`0`),
***Pass*** parametar može imati vrijednost `0`, `1` ili `2`:

- `0` – znači isključeno.
- `1` – označava da se pokreće `fsck` procedura provjere stanja datotečnog sustava prvo na ovoj particiji.
- `2` – označava da se pokreće `fsck` procedura na ovoj particiji nakon one koja ima broj jedan (1).

***Pass****-ing* parametar označava postoji li potreba za automatskim pokretanjem provjera datotečnog sustava (`fsck`) za korijenski direktorij odnosno *root* (`/`), za koji se preporučuje da bude postavljen na `1`. U slučaju root (`/`) datotečnog sustava to znači kako će datotečni sustav biti provjeravan automatski nakon određenog broja restarta sustava. Kako je to konfigurirano, možemo vidjeti s naredbom `dumpe2fs`. Probajmo saznati i kako to izgleda na našem sustavu, na *root* particiji `/dev/sda1`.
Izlistati će se dosta toga, a nas zanima samo donji dio ispisa (koji smo ispisali i označili):

```
dumpe2fs -h /dev/sda1

Mount count: 9
Maximum mount count: 36
```

To znači kako je sustav restartan `9` puta te da će se nakon `36` puta restarta pokrenuti naredba `fsck` odnosno provjera stanja datotečnog sustava ove particije diska (tzv. *File System Check*).

**Izvor informacija: (K-12), `man mount`,`man 5 fstab`.**

## 13.8.2. Swap particija

Druga važna particija koju svaki Linux (i UNIX sustav) mora imati je takozvana *swap* particija. U *Windows* operativnim sustavima postoji *swap* datoteka, a u Linux svijetu je za to potrebna posebna *swap* particija, s posebnim *swap* datotečnim sustavom prilagođenim za ovakvu primjenu. Naime u slučaju kada nam je sva radna (RAM) memorija puna, a za rad nam je potrebno još RAM memorije, sustav uključuje *SWAP* memoriju odnosno počinje koristiti *swap* particiju na tvrdom disku kao proširenje RAM memorije. To je proces koji se odrađuje automatski od strane operativnog sustava.
Pri tome *SWAP* nije zamjena za RAM memoriju, već je sâmo privremena pomoć odnosno njeno privremeno proširenje.



Za više detalja o sustavu virtualne memorije, unutar kojega je implementiran i *swap*, pogledajte poglavlje:
**12. Sustav virtualne memorije i memorijski menadžment.**

Preporuke tijekom kreiranja *swap particije* (ovo se radi sâmo tijekom instalacije operativnog sustava) su:

- Ako sustav ima do 1GB RAM, kreirati *swap particiju* dvostruke veličine RAM memorije.
- Ako sustav ima preko 2 GB RAM (do cca. 8GB), kreirati *swap particiju* iste veličine kao RAM memorija.
- Za veće količine RAM memorije (8+GB), sve ovisi o aplikacijama koje se koriste na sustavu.

U slučaju kada vam sustav intenzivno počinje koristiti *swap*, to je znak da je potrebno nadograditi RAM memoriju.
Drugi razlog upotrebe *SWAP* memorije može biti u slučaju kada neki program traži još memorije ili ako se pokreće novi program, a sustav je u tom trenutku nema slobodno, ali ima dosta *cache* memorije (*Page/disk cache*), koja je u tom trenutku zauzeta. Stoga će sustav privremeno početi koristiti *SWAP* za potrebe našeg programa.

Međutim on će istovremeno početi prazniti *cache* memoriju i potom ju smanjiti kako bi oslobodio RAM memoriju za naš program, koji potom može prebaciti u nju. Sljedeći slučaj su programi koji odmah ili u nekom trenutku mogu zatražiti veći blok memorije u nîzu, koje trenutno nema jer je trenutno fragmentirana, pa će se koristiti *SWAP* sve dok sustav ne defragmentira i oslobodi potreban dio RAM memorije.
Zauzeće RAM memorije se može provjeriti i pomoću naredbe `free`. Provjerimo koliko nam je zauzeće memorije (u GB), pri čemu poslužitelj (računalo) u primjeru ima 96GB RAM i *SWAP* particiju veličine 93GB:

```
free -g
                      total     used          free   shared   buffers   cached
Mem:                    96        60            33      0         0        40
-/+ buffers/cache:      19        74
Swap:                   93         0            93
```

Pogledajmo kako izgleda unos u datoteci `/etc/fstab` za *swap* particiju, ako je swap particija `/dev/sda2`:

```
/dev/sda2  none  swap  sw    0     0
```

Pogledajmo koja *swap* particija je u upotrebi. Za ovu namjenu ćemo koristiti naredbu `swapon` s prekidačem `-s`, koja će nam dati status vezan za *swap* particije. Naredba `swapon` ove podatke izvlači iz datoteke: `/proc/swaps`.

```
swapon -s

Filename                          Type             Size        Used     Priority
/dev/sda2                         partition        127965744   0        -1
```

Vidimo kako se za razliku od root (`/`) particije, *swap* particija ne *montira (mounta)* nigdje, te da je njegova vrsta: `swap`.

**U posebnim slučajevima, kod nekih oblika virtualizacije, možemo i trajno isključiti swap particiju brisanjem cijelog njenog unosa iz datoteke: `/etc/fstab` odnosno privremeno, za konkretnu particiju, sa sljedećom naredbom:**

```
swapoff /dev/sda2
```



*Za virtualizaciju je ponekad bolje ostaviti SWAP particiju u upotrebi te postaviti sysctl varijablu:* **`vm.swappiness`** *na nula* (0). *Važno je razumjeti da SWAP memorija može biti alocirana, ali ne i u upotrebi u datom trenutku. To možemo vidjeti u sljedećoj statistici (**opisano u cjelini: 13.8.2.1. Malo više detalja o swap sustavu i particiji.**), upotrebom naredbe* `vmstat` *kako je vidljivo u stupcu „*`---swap--`*“.*

***Osim SWAP particije, u posebnim slučajevima, moguće je kreirati i Swap datoteku ili Swap LVM ili drugi logički volumen.***

## Kako kreirati novu swap particiju?

Kreirajmo particiju na klasičan način s programima `fdisk`, `cfdisk`, `gdisk` ili `cgdisk`; To probajmo napraviti s programom `fdisk`, ako se radi o disku `/dev/sdc`, na sljedeći način (izbore koje smo napravili smo podebljali):

```
fdisk /dev/sdc
```

Upisujemo **n** za kreiranje nove particije

```
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
```

Upisujemo **p** jer želimo kreirati primarnu particiju:

```
Select (default p): p
```

Upisujemo **1** jer želimo kreirati prvu particiju:

```
Partition number (1-4, default 1): 1
First sector (2048-16777215, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-16777215, default 16777215):

Created a new partition 1 of type 'Linux' and of size 8 GiB.
```

Sada, pošto je vrsta particije standardna „*Linux*“, to moramo promijeniti u *swap*, odnosno oznaku **82**, stoga upisujemo **t**:

```
Command (m for help): t
Selected partition 1
Hex code (type L to list all codes): 82
Changed type of partition 'Linux' to 'Linux swap / Solaris'.
```

I na kraju sve moramo snimiti sa **w**:

```
Command (m for help): w
The partition table has been altered.
Syncing disks.
```

S ovim zadnjim korakom prva *swap* particija na disku `/dev/sdc` je kreirana pa dobivamo particiju: `/dev/sdc1`.
Sada slijedi formatiranje *swap* particije s posebnim *swap* datotečnim sustavom, pomoću naredbe `mkswap`:

```
mkswap /dev/sdc1
```

Ako ju želimo dodati u datoteku `/etc/fstab`, tada u nju na kraj trebamo dodati sljedeći redak teksta odnosno konfiguracije:

```
/dev/sdc1                swap                     swap    defaults         0 0
```

I sada je na redu aktivacija novo kreirane *swap* particije, pomoću naredbe `swapon` koja će aktivirati sve *swap* particije iz: `/etc/fstab` datoteke.

```
swapon -a
```

I konačno provjerimo koje sve *swap* particije imamo aktivne:

```
swapon -s

Filename                                Type            Size    Used    Priority
/dev/sda2                               partition       3104764 0       -2
/dev/sdc1                               partition       8387580 0       -3
```

- Vidimo dvije *swap* particije u upotrebi: već prije kreiranu (`/dev/sda2`) i novo dodanu (`/dev/sdc1`).

**Kako isključiti ovu *swap* particiju privremeno?**

```
swapoff /dev/sdc1
```

Trajno ju možemo isključiti brisanjem unosa u datoteci: `/etc/fstab`.

## Kako kreirati swap datoteku (i zašto)?

U slučaju kada nam je hitno potrebno kreirati novu *swap* particiju to jest *swap* prostor, ali nemamo više mjesta na disku za kreiranje nove *swap* particije, moguće je kreirati i posebnu *swap* datoteku i koristiti ju kao *swap* particiju.

Prvo trebamo kreirati praznu datoteku; što možemo napraviti na više načina, Mi ćemo to uraditi s naredbom `dd` kreirajući datoteku veličine 1GB. U našem slučaju ovu datoteku ćemo kreirati u direktoriju to jest mapi `/NOVI`.

```
dd if=/dev/zero of=/NOVI/swapfile bs=1M count=1024

1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 2.20851 s, 486 MB/s
```

Ovisno o sigurnosnim željama, ovu datoteku možemo zaštititi (sve ovo radimo kao korisnik `root`) da samo `root` korisnik ima potpuna prava upotrebe:

```
chmod  600 /NOVI/swapfile
```

I sada konačno kreirajmo posebni *swap* datotečni sustav na ovoj datoteci, kako da se radi o *swap* particiji:

```
mkswap /NOVI/swapfile

Setting up swapspace version 1, size = 1024 MiB (1073737728 bytes)
no label, UUID=83b44a5b-d7a8-4559-9b5f-a1127f2e499a
```

Nakon što je formatiranje gotovo, dobili smo *swap* datoteku.
Ako ju želimo dodati u datoteku `/etc/fstab`, tada u nju na kraj trebamo dodati sljedeći redak teksta odnosno konfiguracije:

```
/NOVI/swapfile              swap                    swap    defaults        0 0
```

I sada je na redu aktivacija novo kreirane *swap* particije, pomoću naredbe `swapon` koja će aktivirati sve *swap* particije iz: `/etc/fstab` datoteke.

```
swapon -a
```

I konačno provjerimo koje sve *swap* particije imamo aktivne:

```
swapon -s

Filename                                Type            Size    Used    Priority
/dev/sda2                               partition       3104764 0       -2
/NOVI/swapfile                          file            1048572 0       -3
```

- Vidimo već prije kreiranu *swap* particiju `/dev/sda2` te novo kreiranu *swap* datoteku `/NOVI/swapfile`.

**Kako privremeno isključiti ovu *swap* datoteku?**

```
swapoff /NOVI/swapfile
```

Trajno ju možemo isključiti brisanjem unosa u datoteci: `/etc/fstab`.

Zauzeće i upotrebu *swap* particije možemo vidjeti sa sljedećim naredbama: `free`, `top`, `vmstat` i drugima.

**Izvori informacija: (476), (K-12), `man free`, `man swapon`, `man swapoff`, `man mkswap`, `man top`, `man vmstat`, `man 5 fstab`, `man 5 proc`.**

### 13.8.2.1. Malo više detalja o *swap* sustavu i particiji

Kako ćete vidjeti u sljedećem poglavlju, u određenim datotekama unutar `/proc` direktorija, nalaze se i datoteke u koje se zapisuju razne statistike vezane i za *swap* particiju. Jedna od tih datoteka je i datoteka `/proc/meminfo`. Ukratko u njoj se zapisuju statistike o upotrebi *swap* memorije, a koje nam daje i naredba `free`. Druga korisna naredba je naredba koja dolazi u softverskom paketu imena `sysstat`, a zove se `vmstat`. Ova naredba nam također može dati osnovne podatke o iskorištavanju *swap* particije. **Više o ovoj naredbi, pogledajte u poglavlju: 10.7.2.3.5. te u narednim poglavljima.**
Drugi primjer upotrebe naredbe `vmstat` za ovu potrebu bi bio sljedeći (fokusirali smo se na trenutne *SWAP* statistike):

```
vmstat

procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  0      0 1639248  37048 253984    0    0     8    12   36  336  0  0 99  0
```

Pogledajmo stupac: `--swap--` u kojem imamo sljedeća polja (vezano za *swap*):

- `si` (*swap IN*) - on označava koliko se memorije trenutno čita sa *swap* particije.
- `so` (*swap OUT*) - on označava koliko se podataka iz RAM memorije počinje prebacivati u *swap* memoriju na disku.

Za više informacija moramo pogledati statistike iskorištenja sustava virtualne memorije, za svaki proces (prema ***PID*** broju).
Ove statistike se nalaze u direktoriju: `/proc/PID/status` pri čemu je *PID*, *process ID* od svakog pojedinog pokrenutog Linux procesa. Ako nas recimo zanima statistika za program (servis) `nginx`, njegov ***PID*** broj ćemo saznati ovako:

```
pidof nginx

3953 3952 3951
```

Probajmo prvo sa prvim procesom (***PID*** broj `3953`), jer ih *nginx* servis u ovom slučaju ima pokrenuta tri.
Dakle ovdje tražimo ključnu riječ `VmSwap` jer je tu vidljiva količina *swap* memorije u upotrebi; ako se uopće koristi:

```
grep VmSwap /proc/3953/status
VmSwap:         0 kB
```

Vidimo kako je vrijednost `0`, što znači kako ovaj proces uopće ne koristi *swap* memoriju.

**Primjer: kako pronaći konkretno zauzeće *swap* memorije za sve pokrenute programe (*procese*) pod Linuxom?**
Izlistat ćemo sve datoteke sa statistikama upotrebe sustava virtualne memorije, za sve procese (sve ***PID*** *brojeve*), i to tražeći ključnu riječ: `VmSwap` na sljedeći način:

```
grep VmSwap /proc/*/status
```

Gore navedenom naredbom bi dobili ispis za svaki ***PID*** *broj* procesa. Sada to napravimo malo ljepše, uz sortirani ispis.
Naime ovdje možemo koristiti i polje `Name`, koje nam daje ime procesa/programa, uz polje koje tražimo: `VmSwap`.
Sada pokrećemo nîz naredbi koje trebate izvršiti u jednom redu:

```
for file in /proc/*/status ; do awk '/VmSwap|Name/{printf $2 " " $3}END{ print ""}'
$file; done | sort -k 2 -n -r | uniq | grep -v "0 kB" | grep kB
```

U ispisu ovog nîza naredbi, neki od posebnih programa, a ovo se odnosi na kernel *thread* programe, imaju malo drugačije statistike pa one neće biti vidljive brojčano, a i ne interesiraju nas statistike kernel *thread* programa, već *normalnih* programa.
Gore navedenu skriptu (ovaj nîz naredbi u jednom redu) možete i sâmi malo poboljšati.

**Primjer ispisa gore navedene skripte, u slučaju kada se *swap* particija (disk) počinje intenzivno koristiti**
Dolje je vidljiv primjer u kojem su na računalu pokrenuta virtualna računala, a koja su iskoristila svu dostupnu dodijeljenu im RAM memoriju, te su u nedostatku RAM memorije počela koristiti i *swap* memoriju (disk). Ovdje se moglo raditi o bilo kojem programu koji je potrošio svu dostupnu RAM memoriju, a tada bi operativni sustav morao početi koristiti *swap* disk, kao nadogradnju odnosno proširenje RAM memorije:

```
kvm 3841912 kB
kvm 1762252 kB
kvm 1204392 kB
```

Vidljiva su tri procesa (programa) `kvm`, a to su *hipervizori* za virtualna računala, od kojih svaki troši podosta *swap* memorije jer je iskoristio svu dodijeljenu mu RAM memoriju, što naravno uopće **NIJE** dobro za cijeli sustav.

Pogledajmo koji su to procesi koji su počeli intenzivnije koristiti *swap* particiju (disk):

- Prvi `kvm` proces troši oko 3,8 GB (`3841912 kB`) *swap particije.*
- Drugi `kvm` proces troši oko 1,7 GB (`1762252 kB`) *swap particije.*
- Treći `kvm` proces troši oko 1,2 GB (`1204392 kB`) *swap particije.*

Rješenje ovog problema je nadogradnja (kupnja) RAM memorije, kako virtualna računala ne bi bila primorana koristiti *swap* disk. Dodatno, potrebno je razmisliti i o postavkama upotrebe swap memorije (ili čak isključivanje iste za virtualna računala).



Za optimizaciju memorije, pogledajte cjelinu:
**12.5. *Out of Memory* stanje rada i konfiguracija *OOM killera*.**

**Izvori informacija: (671),(672),`man awk`,`man sort`,`man uniq`,`man 5 proc`.**

### 13.8.2.2. *Swappiness* i druge opcije

Svi Linux i Unix sustavi pokušat će iskoristiti svu fizičku (RAM) memoriju koja je dostupna sustavu, često stvaranjem među spremnika za rad s datotekama, što jednostavno predstavlja I/O (diskovnu) međumemoriju koja će poboljšati performanse sustava. Tehnički ova međumemorija za datotečni sustav nije stvarno i stalno sva u uporabi, iako je alocirana za ovu vrstu pred memorije. Stoga će sustav, ako je to potrebno; primjerice za nove aplikacije ili za one već pokrenute koje trebaju više memorije, osloboditi dio ove predmemorije i dodijeliti ju aplikacijama kojima je to potrebnije (ova procedura se zove ***memory pressure***). Istovremeno sustav pazi i na to da određeni dio slobodne RAM memorije bude stalno dostupan, za aplikacije kojima hitno treba alociranje određene količine memorije, bez čekanja na proceduru koja će osloboditi nekorišteni dio diskovne međumemorije. Naime postoje i aplikacije koje mogu zahtijevati od sustava, odmah dostupan određeni veći blok memorije, bez čekanja na normalnu proceduru oslobađanja RAM memorije. Dakle postoji i takozvana atomska alokacija memorije, kod koje se zahtjev za alociranjem memorije mora zadovoljiti bez čekanja na neku dodatnu intervenciju sustava, što je recimo potrebno kada aplikacija ima programsku nît koja trenutno ne može biti zaustavljena. To se najčešće događa i kod rutina za signale prekida (*interrupt [**IRQ**]*), ali se primjenjuje i na sve slučajeve u kojima je potrebno brzo alociranje memorije u trenutku dok primjerice aplikacija drži zaključanu određenu varijablu, sprječavajući drugima da ju u tom trenu mijenjaju.

Stoga ovo dodjeljivanje memorije mora biti što brže jer si aplikacija ne možete priuštiti da čeka sistemsku proceduru koja će osloboditi memoriju. Drugi primjer je programski jezik ***Java*** u kojemu je moguće tijekom pokretanja aplikacije postaviti takozvanu ***heap size*** memoriju i to minimalnu *heap* veličinu (*`-Xms`*); jer je moguće i definirati maksimalni *heap size* (*`-Xmx`*).

Aplikacija koja ima postavljen minimalni *heap size,* a ako je on prilično velik, tako da u trenutku pokretanja programa nema dostupno upravo toliko memorije u komadu, aplikacija će se sâma srušiti.
Nadalje, postoji i vrijednost minimalne, uvijek dostupne količine RAM memorije koju će Linux uvijek pokušati čuvati slobodno, a ona se definira u *sysctl* varijabli imena: `vm.min_free_kbytes`. Važno je znati kako sustav virtualne memorije (*VM*) koristi taj broj za izračunavanje postavljanja rezervirane količine RAM memorije unutar zona memorije označenih kao `pages_low`. Pri tome svaka zona memorije označena kao `pages_low` dobiva određeni broj rezerviranih stranica memorije proporcionalno njegovoj veličini, a sveukupne veličine definirane u gornjoj varijabli.



Što su zône memorije i kako rade, pogledajte u poglavlju:
**12.3.5 Zone RAM memorije.**

U svakom slučaju, ako je ova (`vm.min_free_kbytes`) vrijednost veća, sustav će se stalno brinuti da čisti memoriju kako bi je uvijek bilo dovoljno slobodne (definirano u navedenoj varijabli), pa se sustav efektivno zbog tog procesa stalnog čišćenja s vremenom može usporavati; ovisno koliko ima RAM memorije i koliko je slobodno. Dok se postavljanjem ove vrijednosti na još veću (**7% - 10%** sistemske RAM memorije) sustav može dovesti u stanje da brzo ostane bez slobodne memorije.
Dakle potrebno je oprezno odabrati ovu veličinu ili obično ništa ne dirati. Ukupnu količinu memorije dostupne za *normalne* programe možete vidjeti s naredbom: `free -m` u stupcu `available`. Pogledajte i poglavlje: **12.3.3.1. Naredba *free***.

**Kako promijeniti ovu vrijednost?**
Ako primjerice imamo poslužitelj sa 128GB RAM memorije, a imamo aplikacije koje trebaju *atomsku* operaciju alociranja memorije koju smo objasnili (ili ako želimo rezervirati ovu količinu memorije za sistemske servise). Stoga bi recimo mogli rezervirati 5% RAM memorije, što je na gornjoj preporučenoj granici. To u našem slučaju znači 5GB RAM za ove operacije. Pri tome je 5GB = `5.248.000` kB. Stoga možemo napraviti sljedeće:

```
echo 5248000 > /proc/sys/vm/min_free_kbytes
```

Ili sve naravno ubaciti u datoteku: `/etc/sysctl.conf` u varijabli imena: `vm.min_free_kbytes`.

## *Swapiness*

U slučajevima kada sustav ostane bez slobodne RAM memorije, memorijski menadžment mora krenuti u proširenje memorije upotrebom *swap particije* kao proširenja RAM memorije, što naravno usporava sustav, ali ipak osigurava njegovu stabilnost.
***Swappiness*** je parametar Linux kernela koji kontrolira relativnu težinu koja se daje u procesu zamjene korištenja radne memorije (RAM), koja se koristi za aplikacije u korist prebacivanja iste na swap (disk).

*Swappiness* može biti postavljen na vrijednosti između 0 i 100 uključujući obje vrijednosti.
Niža vrijednost govori kernelu da izbjegava upotrebu *swap particije* koliko god je to moguće. Veća vrijednost govori kernelu da pokušava koristiti swap particiju znatno ranije nego što bi možda bilo potrebno. Zadana vrijednost je 60; postavljanje veće će povećati performanse "vrućih" procesa uz trošak vraćanja na neaktivne "hladne" procese odnosno one koji uzimaju dužu pauzu u radu odnosno duže su neaktivni. Dok postavljanje niže (čak 0 ili 1) može smanjiti latenciju sustava.



*To znači da će kod veće vrijednosti sustav pokušati one procese koji su slabo ili povremeno aktivni prebaciti iz RAM memorije u swap memoriju odnosno na swap particiju. Stoga kako bi se memorija oslobodila za one procese koji su aktivniji, a koji će možda trebati više RAM memorije, koja će tada biti oslobođena.*

Kod sustava s više od potrebne količine RAM memorije za bilo koju očekivanu zadaću možda želite drastično smanjiti ove postavke. Pogledajmo konfiguraciju *swapinessa* i parametra vezanih za njega, prema ***sysctl*** varijablama:

- `vm.swappiness=`**`X`** – odnosno `/proc/sys/vm/swappiness` - ovdje se definira koliko agresivno će sustav koristiti ili ne koristiti swap particiju, pri tome **`X`** može biti:
  - `0` - ovo znači kako se *swap particija* koristi najminimalnije moguće (**ovisno o inačici kernela**) pa se sustav neće dovesti u ***OOM*** stanje u slučaju nedostatka RAM memorije. U slučaju nedostatka RAM i SWAP memorije sustav prelazi ili u ***panic*** način rada ili prvo u ***OOM*** način rada, pa tek potom, ako nakon ubijanja prevelikih aplikacija ne ostane slobodne RAM memorije u ***panic*** način rada, ovisno o konfiguraciji sustava.



Pogledajte poglavlje: **12.5. *Out of Memory* stanje rada i konfiguracija *OOM killera*** i varijablu `vm.panic_on_oom`.

  - `1` - ovo znači kako je *swap particija* gotovo u minimalnoj, ali ne i u najminimalnijoj upotrebi.
  - `10` - ovo je preporučena vrijednost, ako se *swap particija* želi minimalno koristiti na sustavim s više nego dovoljno RAM memorije.
  - `60` - ovo je preporučena i standardno postavljena vrijednost za većinu sustava.
  - `100` - ovo je maksimalna vrijednost pri kojoj će se *swap particija* maksimalno koristiti, kada god je to moguće.
- Pogledajte i varijablu: `vm.vfs_cache_pressure` u poglavlju: **14.1.1 Sinkroni i asinkroni I/O**.



Pogledajte i poglavlje: **14.1.2 Optimizacija Page Cache sloja** u kojem su definirane i ostale optimizacije koje utječu na rad i iskorištavanje virtualne memorije (*RAM+SWAP*).

Postavljanje ovih vrijednosti može se napraviti pomoću `sysctl` naredbe, pa recimo *swapiness* možemo postaviti na `10`, na sljedeći način:

```
sysctl -w vm.swappiness=10
```

Naravno sve je moguće i postaviti permanentno (trajno), zapisivanjem slijedećeg retka u datoteku: `/etc/sysctl.conf` :

```
vm.swappiness=10
```



Za više detalja o jednom od stanja rada sustava kada ostane bez slobodne memorije, pročitajte u poglavlju: **12.5 *Out of Memory* stanje rada i konfiguracija *OOM killera*.**

**Izvori informacija:** (476),(477),(1405),**`man sysctl`**,**`man 5 procfs`**.

# 13.9. Posebne particije (`/proc`, `/sys` i `/tmpfs`)

**Slijedi napredna cjelina (13.9.x)!**
Osim *root* (`/`) particije, na kojoj se nalazi cijeli operativni sustav, tijekom podizanja sustava, *montiraju* se i posebni pseudo datotečni sustavi za posebne namjene, ali i neki direktoriji (mape) čija namjena je također specifična.

U sljedećim cjelinama upoznat ćemo se sa:

- `/proc` – direktorij (mapa) u koji se montira pseudo datotečni sustav naziva ***procfs***, čiji sadržaj se nalazi u virtualnoj memoriji.
- `/sys` – direktorij (mapa) u koji se montira pseudo datotečni sustav naziva **sysfs**, čiji sadržaj se također nalazi u virtualnoj memoriji.
- `/dev/shm`, `/run`, i drugi – direktoriji (mape) u koje se montira posebni pseudo datotečni sustav naziva **tmpfs**, čiji sadržaj se također nalazi u virtualnoj memoriji.
- `/tmp` direktorij ne koristi se od strane posebnih (pseudo/virtualnih) datotečnih sustava već je njegova namjena za pohranu privremenih datoteka. On se nalazi unutar već montiranog sistemskog `/` vršnog direktorija (o njemu kasnije).

**Izvori informacija:** **`man 5 procfs`**,**`man 5 sysfs`**,**`man 5 tmpfs`**.

### 13.9.1. Direktorij `/proc`

Virtualni datotečni sustav ***procfs*** *montira* se u direktorij `/proc` a kreira se u *RAM* memoriji, tijekom svakog pokretanja sustava. On nam daje informacije o svim procesima, poruke kernela te sistemske atribute (koje je moguće mijenjati).
Naredbe `ps`, `top`, `free` se koriste za čitanje iz njega. Dok primjerice `sysctl` i slične, koriste za čitanje i zapisivanje u specifične datoteke unutar ovog direktorija. Kreira ga ***proc(fs)*** datotečni sustav u trenutku podizanja sustava. Dakle ovaj virtualni datotečni sustav kreira hijerarhijsku strukturu poddirektorija i datoteka koje predstavljaju trenutna stanja i/ili postavke komponenti kernela. Pogledajmo opis nekih od datoteka unutar `/proc/` sustava (direktorija):

- `/proc/apm` – daje nam informacije o *Advanced power management (**APM**)* sustavu.
- `/proc/bus` - direktorij koji sadrži informacije o uređajima spojenim na sve sabirnice (PCI, USB, …).
- `/proc/cmdline` - ispisuje parametre s kojima se kernel učitao u trenutku pokretanja sustava.
- `/proc/cpuinfo` - pokazuje podatke o procesoru (CPU): model, vrsta, generacija, takt, zastavice (*flags*),…
- `/proc/devices` - sadrži listu svih upravljačkih programa koji su trenutno aktivni u kernelu.
- `/proc/diskstats` - sadrži statistike rada svih diskova (osvježava se u realnom vremenu). Pogledajte [(215)].
- `/proc/dma` - informacije o DMA kanalima.
- `/proc/filesystems` - lista datotečnih sustava koji su trenutno podržani od sustava.
- `/proc/ide` - direktorij unutar kojega su razni IDE (disk) uređaji.
- `/proc/iomem` - memorijske adrese svih uređaja (Video ROM, VGA, PCI sabirnica, ACPI, …).
- `/proc/ioports` - I/O portovi (memorijske adrese) za komunikaciju sa svim uređajima.
- `/proc/irq` - direktorij s poddirektorijima za svaki hardverski signal prekida IRQ (*interrupt*).
- `/proc/interrupts` - pokazuje podatke o hardverskim signalima prekida (*interrupts*).
- `/proc/kcore` - *image* (slika) cijele RAM memorije računala (trenutno aktivne - stalno se osvježava).
- `/proc/kmsg` – sadrži trenutne poruke kernela.
- `/proc/loadavg` – prikazuje trenutno opterećenje sustava.
- `/proc/locks` - kernel *locks* odnosno lista datoteka koje je kernel zaključao (*lock*).
- `/proc/meminfo` - prikazuje podatke o RAM memoriji, zauzeću *swap* memorije i druge podatke o ostalim dijelovima sustava virtualne memorije.
- `/proc/modules` - pokazuje podatke o trenutno učitanim kernel modulima.
- `/proc/mounts` - pokazuje što je sve montirano (*mountano)* od particija.
- `/proc/mttr` - daje nam informacije o **MTRR** *Memory Type Range Registers* vrsti procesorskih registara, zaduženih za pristup memorijskom opsegu unutar kojega se nalaze uređaji poput ***PCI*** ili ***AGP*** grafičkih kartica i njihove memorije.
- `/proc/net/` - unutar ovog direktorija su statistike i konfiguracije mrežnih kartica, protokola i opcija:
  - `/proc/net/arp` - ovdje se nalazi ***ARP*** tablica.
  - `/proc/net/dev` - ovdje su mrežni uređaji i njihove statistike.
  - `/proc/net/dev_mcast` - Layer 2 *multicast* grupe.
  - `/proc/net/dev_stat` - status mrežnih sučelja.
  - `/proc/net/ip_fwchains` - Firewall *chain linkage.*
  - `/proc/net/ip_fwnames` - imena *firewall chaina.*
  - `/proc/net/ip_masq` - *masquerading* tablice.
  - `/proc/net/netstat` - mrežne statistike.
  - `/proc/net/raw` – *raw (sirove)* mrežne statistike.
  - `/proc/net/route` - tablica usmjeravanja (*routing*) kernela.
  - `/proc/net/snmp` - SNMP podaci.
  - `/proc/net/sockstat` - statistika na razini *socketa.*
  - `/proc/net/tcp` - TCP *socketi.*
  - `/proc/net/udp` - UDP *socketi.*
  - `/proc/net/unix` - UNIX *domain socketi*
  - `/proc/net/wireless` - *wireless* mrežna sučelja – podaci.
  - `/proc/net/igmp` - multicast IP adrese kojima se računalo pridružilo
  - `/proc/net/psched` - globalni *packet scheduler* parametri
  - `/proc/net/udp6` - UDP ipv.6 *socket.*
  - `/proc/net/tcp6` - TCP ipv.6 *socket.*
  - `/proc/net/raw6` - *raw (sirove)* mrežne statistike za IP v.6..
- `/proc/parport` - informacije o paralelnom portu.
- `/proc/partitions` - sadrži podatke o diskovima i particijama diskova (uz *MAJOR* i *MINOR* oznake).
- `/proc/rtc` - informacije o RTC (*Real time clock*).
- `/proc/self/` - je poveznica na trenutno pokrenuti proces. Omogućava svakom procesu da gleda svoje parametre i opcije bez potrebe da poznaje svoj ***PID***.
- `/proc/scsi/` - direktorij sa ***SCSI*** uređajima, njihovoj konfiguraciji i sl.
- `/proc/slabinfo` - daje informacije o zauzeću memorije na *slab* razini.

- `/proc/stat` - daje razne statistike o sustavu (*CPU, iowait, IRQ, soft IRQ, steal time, …* ).
- `/proc/swaps` - daje statistike o *swap* prostoru/particiji.

- `/proc/sys/` - u ovom direktoriju se nalaze poddirektoriji i datoteke zadužene za spremanje raznih opcija i parametara svih komponenti sustava te dodatnih informacija o njemu:
  - `/proc/sys/fs` - daje informacije o datotečnom sustavu (*Filesystem*):
    - `/proc/sys/fs/dentry-state` - status *directory cache*-a.
    - `/proc/sys/fs/file-max` - sadrži maksimalan broj *file deskriptora* koji se može alocirati
    - `/proc/sys/fs/…`
  - `/proc/sys/kernel` - direktorij koji sadrži mnoge pôddirektorije i datoteke zadužene za funkcioniranje globalnih funkcija kernela.
  - `/proc/sys/vm` - direktorij u kojemu se nalaze mnoge opcije (datoteke) zadužene za sustav virtualne memorije (*Virtual Memory*) i komponente koje se oslanjaju na sustav virtualne memorije, čije trenutne postavke se mogu mijenjati.
    - `/proc/sys/vm/drop_caches` - ovdje se može promijeniti ponašanje *cache* memorije (standardno je 0):
      - `1` - za pražnjenje (oslobađanje) disk *cache* memorije.
      - `2` - za pražnjenje *slab cache* memorije.
      - `3` - za pražnjenje (oslobađanje) i disk i *slab cache* memorije.
    - `/proc/sys/vm/…`
  - `/proc/sys/dev` - ovdje se nalaze parametri specifični za pojedine uređaje (RAID kontroler, SCSI, …)
    - `/proc/sys/dev/scsi/logging_level` - ovo je konfiguracijska datoteka u kojoj se za *SCSI* uređaje (SCSI/SAS/SATA) može uključiti znatno veća razina logiranja. Standardno je isključena (**0**), što znači kako se logira samo ono osnovno.
    - `/proc/sys/dev/raid/` - ovdje se nalaze konfiguracijske datoteke za Linux softverski RAID, vezane za ograničenja brzine *RAID rebuilda* (kada se dogodi)[216].
  - `/proc/sys/net/ipv4/` - direktorij u kojemu se nalaze parametri i konfiguracija mrežnih protokola od razine OSI 3 na dalje: IP v.4., ICMP, TCP, UDP, …
- `/proc/version` pokazuje podatke o kernelu (inačica, arhitektura, …)

Standardni unos `/proc` datotečnog sustava/particije u datoteci `/etc/fstab` je sljedeći:

```
proc          /proc         proc      defaults         0    0
```

Unutar `/proc/` direktorija nalaze se prvo pôddirektoriji s brojevima u nazivu. Od njih svaki broj predstavlja ***PID*** pokrenutog procesa, a točno je definirana i struktura poddirektorija unutar ***PID*** direktorija. Tako je primjerice ***PID*** broj od našeg trenutno pokrenutog *httpd* (web poslužitelj) procesa broj. *4027* pa bi njegove statistike tada bile unutar direktorija `/proc/4027/`

Stoga je definirana sljedeća struktura unutar ***PID*** direktorija (*PID* ćemo nazvati direktorijem koji nosi neki *PID* broj):

- `/proc/`**`PID`**`/cmdline` - argumenti s kojima je proces (program/aplikacija s ovim ***PID*** brojem) izvršena.
- `/proc/`**`PID`**`/cpu` - trenutna CPU jezgra, koja ga pokreće.
- `/proc/`**`PID`**`/cwd` - link na trenutni radni direktorij ovog procesa.
- `/proc/`**`PID`**`/environ` - vrijednosti *environment* varijabli pokrenutog procesa.
- `/proc/`**`PID`**`/exe` - link na izvršnu datoteku (*executable*).
- `/proc/`**`PID`**`/fd/` - u ovom direktoriju se nalaze svi *file deskriptori* od ovog procesa.
- `/proc/`**`PID`**`/maps` - ovo su memorijske mape (*Memory maps*) izvršnih datoteka i biblioteka ovog procesa, iz kojih se vidi u koje su se memorijske lokacije učitale.
- `/proc/`**`PID`**`/mem` - ovo datoteka koja pokazuje na memoriju koju koristi ovaj proces.
- `/proc/`**`PID`**`/net/` - ovdje se nalaze mrežne statistike za pojedini program/proces/***PID***:
  - `/proc/`**`PID`**`/net/dev` - ovdje se nalaze mrežne statistike za sva mrežna sučelja (mrežne kartice) dostupne ovom programu/procesu/***PIDu***.
  - `/proc/`**`PID`**`/net/netstat` - ovdje se nalaze ukupne mrežne statistike.
- `/proc/`**`PID`**`/root` - link na *root* direktorij ovog procesa.
- `/proc/`**`PID`**`/stat` - statusi ovog procesa.
- `/proc/`**`PID`**`/statm` - statusi memorije, koju zauzima ovaj proces.
- `/proc/`**`PID`**`/status` - status ovog procesa, prikazuje i statistike zauzeća sustava virtualne memorije (*RAM*, *swap*).
- `/proc/`**`PID`**`/task/` - sadrži pôddirektorije prema brojevima programskih nîti (ako ih program ima) s parametrima.

**Izvori informacija: (1033),(1035) , `man 5 procfs`,`man mount`.**

## 13.9.2. Direktorij /sys

Virtualni datotečni sustav ***sysfs*** se *montira* u `/sys` direktorij, a kreira se u RAM memoriji tijekom svakog pokretanja sustava. Ovaj pseudo datotečni sustav preko kernela izvozi informacije o raznim podsustavima kernela, kao i komponentama računala, ali i sâmog sustava, a sve preko virtualnih datoteka, koje se kreiraju u njemu. Drugim riječima on strukturirano i uniformirano prikazuje informacije o sustavu te nudi kontrolu nad njim. S jedne strane upravljački programi (*kernel moduli*) pomoću kernela kreiraju svoju strukturu direktorija i datoteka, slično kao kod `/proc/` sustava, ali na strukturiran način. Tijekom pokretanja sustava, kernel pronalazi uređaje: ***CPU***, disk kontroler, mrežna kartica, usb kontroler i slično te kreira ovakve datoteke. Primjerice njih kao pomoć pri identifikaciji, odabiru i učitavanju ispravnog upravljačkog programa (*kernel modula*), koristi **udevd** servis. S druge strane, razne komponente sustava također preko kernela kreiraju svoje datoteke u kojima se nalaze njihove informacije, ali i razne postavke koje je moguće mijenjati. Isto tako su svi podaci, konfiguracije i parametri za recimo ***SATA*** ili ***SCSI*** disk: `/dev/sda` dostupni unutar poddirektorija: `/sys/block/sda/`. Kada se recimo dodaje vanjski ***USB*** disk, **kernel** kreira pripadajuću datoteku unutar `/sys/` direktorija: `/sys/bus/usb/devices/`.
Virtualni datotečni sustav ***sysfs*** ima sljedeći standardni unos u datoteci `/etc/fstab`, koji izgleda ovako:

```
sysfs        /sys         sysfs   defaults        0    0
```

Pogledajmo i kako je ***sysfs*** *montiran*:

```
mount | grep sysfs

sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime,seclabel)
```

Kako smo već rekli *`/sys`* je strukturiran, a na početku njegove hijerarhijske strukture imamo nekoliko osnovnih dijelova, koji se dalje granaju, pa imamo sljedeću strukturu direktorija (objasniti ćemo samo njih nekoliko):

- `/sys/block/` - ovdje se nalaze *blok uređaji*, poput raznih diskova, odnosno njihove postavke i parametri. Za više detalja pogledajte: https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-block te https://www.kernel.org/doc/Documentation/block/.
    - `/sys/block/sda` - direktorij koji predstavlja prvi SATA/SCSI/SAS disk (`/dev/sda`), unutar njega se nalaze opcije i parametri vezani za rad ovog diska:
        - `/sys/block/sda/capability` - u ovu datoteku se upisuju mogućnosti diska, prema tablici s listom mogućnosti i pripadajućih kôdova.
        - `/sys/block/sda/dev` - u ovu datoteku se upisuju *MAJOR:MINOR* identifikatori disk uređaja.
        - `/sys/block/sda/removable` - u ovu datoteku se upisuje je li disk izmjenjiv (*removable*):
            - `0` znači kako nije izmjenjiv.
            - `1` znači kako je izmjenjiv.
        - `/sys/block/sda/ro` - u ovu datoteku se upisuje je li disk samo za čitanje, a ne i za pisanje (*Read only*).
        - `/sys/block/sda/stat` - datoteka u koju se upisuju sve *I/O* statistike rada diska, podijeljene u **11** zasebnih stupaca od kojih svaki predstavlja zasebnu statistiku.
        - `/sys/block/sda/size` - u ovu datoteku se upisuje veličina diska.
        - `/sys/block/sda/uevent` - u ovu datoteku se upisuju informacije o disku, od strane *udevd* servisa.
    - `/sys/block/sda/power/` - unutar ovog direktorija se nalaze datoteke vezane za tzv. „*Power management*" funkcije diska. Za više detalja pročitajte: https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-devices-power
    - `/sys/block/sda/sda1/` - unutar ovog direktorija se nalaze datoteke vezane za prvu particiju ovog diska.
    - `/sys/block/sda/sda2/` - unutar ovog direktorija se nalaze datoteke vezane za drugu particiju ovog diska.
    - `/sys/block/sda/queue/` - unutar ovog direktorija se nalaze datoteke vezane za *disk queue*:
        - `/sys/block/sda/queue/hw_sector_size` - u ovu datoteku se zapisuje hardverska veličina sektora.
        - `/sys/block/sda/queue/iostats` - u ovu datoteku se zapisuje je li trenutno uključeno (1) ili isključeno (0) praćenje I/O statistika ovog diska.
        - `/sys/block/sda/queue/max_sectors_kb`- u ovu datoteku se zapisuje maksimalna veličina (u kB) koje *block layer* (sloj) diskovnog podsustava dopušta. Ovo je najveća veličina bloka podataka prema blok sloju.
        - `/sys/block/sda/queue/nr_requests` - u ovu datoteku se zapisuje maksimalan broj upita prema blok sloju (isti broj vrijedi za upite čitanja i zapisivanja).
        - `/sys/block/sda/queue/physical_block_size` - u ovu datoteku se zapisuje fizička veličina klastera na disku.
        - `/sys/block/sda/queue/read_ahead_kb` - u ovu datoteku se zapisuje broj kB za čitanje unaprijed, kod svake operacije čitanja podataka s diska (*read-ahead*).
        - `/sys/block/sda/queue/rotational` - u ovu datoteku se zapisuje (1) ako se radi o mehaničkom (rotacijskom) disku.
        - `/sys/block/sda/queue/rq_affinity` - u ovu datoteku se zapisuje (**1**) samo, ako se na blok razini diskovnog podsustava za ovaj disk, nalaže sustavu da bilo koja CPU jezgra iz iste "grupe" koja je dohvatila I/O upit prema ili od diska, može nastaviti I/O operacije. Ako je ova vrijednost (**2**) tada se I/O operacije za ovaj disk "zaključavaju" na CPU jezgru koja ih je i započela.
        - `/sys/block/sda/queue/scheduler` - u ovu datoteku se zapisuje koji je *disk scheduler* odabran za ovaj disk.
        - `/sys/block/sda/queue/iosched` - unutar ovog direktorija se nalaze datoteke vezane za odabrani *disk queue*.

- `/sys/bus/` - u ovom direktoriju se nalaze poddirektoriji i njihove pripadajuće datoteke koje su vezane za razne sistemske sabirnice, poput:
    - `/sys/bus/acpi` - u ovom direktoriju se nalaze poddirektorij i njihove pripadajuće datoteke koje su vezane za ***ACPI*** objekte.
    - `/sys/bus/cpu` - u ovom direktoriju se nalaze razni atributi vezani sa *CPU.*
    - `/sys/bus/i2c` - u ovom direktoriju se nalaze poddirektoriji sa uređajima spojenim na **I2C** sabirnicu.
    - `/sys/bus/pci` - u ovom direktoriju se nalaze poddirektoriji sa uređajima spojenim na **PCI** sabirnicu.
    - `/sys/bus/pci_express` - u ovom direktoriju se nalaze poddirektoriji sa uređajima spojenim na **PCI Express** sabirnicu.
- `/sys/class/` - u ovom direktoriju se nalaze grupe poddirektorija poredanih po klasi (kategoriji) uređaja, poput:
    - `/sys/class/ata_device` - u ovom direktoriju se nalaze grupe poddirektorija za **ATA** uređaje.
    - `/sys/class/block` - u ovom direktoriju se nalaze grupe poddirektorija za blok uređaje.
    - `/sys/class/net` - u ovom direktoriju se nalaze grupe poddirektorija za mrežne uređaje.
- `/sys/dev/` - u ovom direktoriju se nalaze grupe poddirektorija kategoriziranih prema dvije vrste uređaja:
    - `/sys/dev/block/` - za sve *blok* uređaje.
    - `/sys/dev/char/` - za sve *karakter* uređaje.
- `/sys/devices/` ovaj direktorij predstavlja vršni direktorij unutar kojega se nalaze svi prepoznati hardverski uređaji, koji su strukturirani u poddirektorijima (svaki u svom), pa tako imamo:
    - `/sys/devices/breakpoint/`
    - **`/sys/devices/pci`**`XXXX:YYY/` - ovo predstavlja **PCI** sabirnice, a XXXX:YYY predstavlja njihovu adresu. Unutar ovog direktorija se nalaze poddirektoriji koji predstavljaju uređaje na toj sabirnici (opet svaki u svom poddirektoriju), primjerice:
        - **`/sys/devices/pci`**`XXXX:YYY/0000:00:00.0/` - prvi uređaj - konkretno CPU timer (IRQ 0).
        - **`/sys/devices/pci`**`XXXX:YYY/0000:00:01.0/`- drugi uređaj - *IR-PCI-MSI-edge* (IRQ 25).

    Pogledajte poglavlje: **11.1.1.7. Mehanizam prepoznavanja hardvera i učitavanje kernel modula**.
    - **`/sys/devices/system/`** - sadrži određene sistemske dijelove, poput:
        - `/sys/devices/system/clocksource/clocksource0/` - sadrži obično dvije datoteke, u kojima su popisani generatori takta za mjerenje vremena:
            - `/sys/devices/system/clocksource/clocksource0/available_clocksource` - ovdje su popisani izvori takta.
            - `/sys/devices/system/clocksource/clocksource0/current_clocksource`
              - ovdje je upisan izvor koji se koristi za takt.
        - `/sys/devices/system/node/` - sadrži razne opcije i parametre, prema ***NUMA*** *CPU* topologiji, pa imamo razne datoteke za općenito ***NUMA*** sustav, ali i poddirektorije od kojih:
            - `/sys/devices/system/node/node0/` - predstavlja prvi fizički *NUMA CPU* te imamo razne poddirektorije koje sadrže mnoge dijelove ovog podsustava, kao i vršne datoteke:
                - `/sys/devices/system/node/node0/numastat` - ova datoteka daje *NUMA* statistike za cijeli fizički *NUMA CPU 0.*
                - `/sys/devices/system/node/node0/meminfo` - ova datoteka daje *NUMA* RAM statistike za cijeli fizički *NUMA CPU 0*.
                - `/sys/devices/system/node/node0/hugepages/` - direktorij koji sadrže statistike upotrebe *HUGEPAGES* memorije.
            - `/sys/devices/system/node/node1/` - predstavlja drugi fizički *NUMA CPU.*
        - `/sys/devices/system/cpu/` - ovdje su vidljive razne opcije *CPUa.* U ovom vršnom direktoriju su zajedničke opcije za sve jezgre a specifične opcije za svaku pojedinu jezgru su unutar poddirektorija. Pogledajte https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-devices-system-cpu:
            - `/sys/devices/system/cpu/modalias` - datoteka koja sadrži informacije o *CPUu*, kao i listu svih funkcionalnosti procesora (pr. **VTx**, **VTd**, **AES**, …).
            - `/sys/devices/system/cpu/cpufreq/` - ovdje su opcije vezane za takt rada procesora:
            - `/sys/devices/system/cpu/microcode/reload` - ovdje su opcije vezane za *CPU microcode*. Ako se ova opcija postavi na vrijednost **1** CPU microcode će se *reloadati* (ponovno učitati).
            - `/sys/devices/system/cpu/cpu0/` - ovdje su opcije za prvu *CPU* jezgru:
            - `/sys/devices/system/cpu/cpu0/topology` - ovdje su opcije vezane za topologiju *CPUa*. Pogledajte: https://elixir.free-electrons.com/linux/latest/source/Documentation/cputopology.txt
            - `/sys/devices/system/cpu/cpu1/` - ovdje su opcije za drugu *CPU* jezgru.
            - `/sys/devices/system/cpu/online` - datoteka u koju se upisuju aktivne jezgre CPUa.
            - `/sys/devices/system/cpu/offline` – ovdje se upisuju neaktivne CPU jezgre.

        - `/sys/devices/system/edac/mc/` - dio koji je zadužen za provjere i korekcije greške (*ECC*) u RAM memoriji. Ovaj direktorij i direktorij iznad njega (**edac**) postoji samo ako imamo ECC memorijski kontroler i ECC memoriju.
            - `/sys/devices/system/edac/mc/mc0/` - predstavlja prvi memorijski kontroler, ako imamo ***SMP*** sustav ovo je jedini od njih.
            - `/sys/devices/system/edac/mc/mc1/` - predstavlja drugi memorijski kontroler, ako imamo ***NUMA*** sustav ovo je drugi po redu (od drugog fizičkog *CPU*a).
        - `/sys/devices/system/machinecheck/` - sadrži poddirektorije, koji sadrže svoju strukturu, a vezano za rezne hardverske greške.
        - `/sys/devices/system/memory/` - sadrži informacije o RAM memoriji kernela:
            - `/sys/devices/system/memory/memory0/` - sadrži interna stanja memorije kernela. Pogledajte https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-devices-memory
- `/sys/firmware/` - unutar ovog direktorija nalaze se poddirektoriji koji su vezani za *BIOS* odnosno *Firmware*, pa obično imamo:
    - `/sys/firmware/acpi/` - unutar ovog direktorija nalaze se poddirektoriji koji su vezani za ***ACPI*** funkcionalnosti:
        - `/sys/firmware/acpi/tables/` - unutar ovog direktorija nalaze se datoteke koje predstavljaju ***ACPI*** tablice vezane za ***ACPI*** podržani hardver. Pogledajte https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-firmware-acpi
    - `/sys/firmware/memmap` - tijekom učitavanja svakog *firmwarea*, memorijska mapa odnosno adrese s memorijskim pozicijama gdje se on učitava u memoriju se zapisuju ovdje. Inače, sve lokacije zajedno su popisane u datoteci `/proc/iomem`, uz druge resurse/programe i njihove lokacije.
- `/sys/fs/` - unutar ovog direktorija nalaze se poddirektoriji u kojima se nalaze datoteke specifične za datotečni sustav koji nam je u upotrebi, pa imamo strukturu poput:
    - `/sys/fs/ext4` - unutar ovog direktorija nalaze se poddirektoriji s opcijama i parametrima za ovaj datotečni sustav, prema particijama na koje je instaliran, poput:
        - `/sys/fs/ext4/sda1` …
        - `/sys/fs/ext4/sdb1` …
- `/sys/hypervisor/` - ovdje se nalaze podaci o hipervizoru (za virtualizaciju) - ovisno o kernelu.
- `/sys/kernel/` - ovdje se nalaze razni poddirektoriji koji sadrže datoteke koje opet sadrže podatke o raznim dijelovima pokrenutog linux kernela, poput:
    - `/sys/kernel/debug/` - ovdje se mogu nalaziti razne datoteke kod *debugiranja* kernela.
    - `/sys/kernel/mm/` - razne datoteke koje sadrže statistike i opcije vezane za sustav virtualne memorije i to za:
        - `/sys/kernel/mm/hugepages` - za *Huge pages*.
        - `/sys/kernel/mm/transparent_hugepages` - za *Transparent Huge Pages*.
- `/sys/module/` - u ovom direktoriju se nalaze poddirektoriji koji predstavljaju, svaki zaseban *kernel modul* koji je učitan u sustav. Unutar svakog direktorija koji predstavlja *kernel modul* nalaze se minimalno dva poddirektorija:
    - `parameters/` - u njemu se nalaze datoteke koje predstavljaju parametre s kojima se *kernel modul* učitao.
    - `refcnt/` - ovaj direktorij postoji samo za one *kernel module*, koji mogu biti isključeni u radu (*unloaded*)
    - … postoje i drugi direktoriji, ovisno o specifičnim mogućnostima konkretnog *kernel modula*.
    - `/sys/module/acpi/` - u ovom direktoriju se nalaze razne opcije i parametri za *ACPI kernel modul*.
    - `/sys/module/ata_generic/` - u ovom direktoriju se nalaze razne opcije i parametri za ***ata_generic*** *kernel modul*.
    - `/sys/module/ata_piix/` - u ovom direktoriju se nalaze razne opcije i parametri za ***ata_piix*** *kernel modul*.
    - `/sys/module/xfs/` - u ovom direktoriju se nalaze razne opcije i parametri za ***xfs*** *kernel modul* (za *XFS* datotečni sustav).
- `/sys/power/` - ovdje se nalaze razne datoteke koje sadrže opcije i parametre vezane za ***Power management***, sustav poput:
    - `/sys/power/state` - ovdje su zapisana dostupna *sleep* stanja, koja je zapisivanjem moguće i mijenjati. Za više detalja pogledajte: https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-power. Moguće vrijednosti su:
        - `mem` - predstavlja *suspend* stanje sustava.
        - `standby` - predstavlja *power-on suspend* stanje sustava.
        - `freeze` - predstavlja *suspend-to-idle* stanje sustava.
        - `disk` - predstavlja *hibernation* stanje sustava.
    - `/sys/power/`… za svako od navedenih stanja (*state*) postoje konfiguracijske datoteke koje se nalaze u ovom direktoriju (`/sys/power/`).



***Za više detalja, pogledajte opise, locirane u zasebnim datotekama imena: sysfs-XXX-YYY, na adresi:*** ***https://www.kernel.org/doc/Documentation/ABI/testing/***

**Izvori informacija: (1034), `man 5 sysfs`,`man mount`.**

### 13.9.2.1. *Systool* programski paket

**Slijedi napredna cjelina!**
*Systool* je softverski paket unutar kojega nam dolazi `systool` sistemski program za lakši prikaz i rad s posebnim konfiguracijskim i statusnim datotekama unutar direktorija `/sys/`
Ovaj softverski paket je ponekad potrebno dodatno instalirati, pa ga stoga instalirajmo:

```
yum -y install systool
```

Sada smo dobili našu željenu naredbu, s kojom ćemo se upoznati kroz par primjera. Prije toga važno je znati kako nam ova naredba može (izvući) konfiguracijske, kao i podatke o uređajima, a koji su dostupni unutar `/sys/` strukture direktorija i datoteka, što samo po sebi znači kako moramo imati potpuno funkcionalan `/sys/` pseudo datotečni sustav.
Ova naredba prepoznaje razne metode preko kojih može doći do svih komponenti na sustavu:

- ***sysfs* sabirnice**: *acpi, cpu, edac, memory, pci, pci_expres, scsi, usb,* …
- ***sysfs* klase**: *bsg, cpuid, dmi, gpio, graphics, ipmi, raw, sas_*, scsi_*, thermal,* …
- ***sysfs* uređaje**: *cpu, pciXY:ABC, platform, pnpXY, power, software, system, tracepoint,*
- ***sysfs* (kernel) module**: *8021q, 8250, acpi, acpi_power_meter, aesni_intel, battery, block, bnx2x, bonding, …*

Pogledajmo i par primjera, kako doći do informacija od željenih komponenti sustava. Ovdje želimo vidjeti više informacija o našem SCSI/SATA disk kontroleru, što ćemo dobiti sa naredbom `systool` na sljedeći način:

```
systool -c scsi_host -v
```

I dobit ćemo nešto poput (skratili smo ispis na samo par detalja):

```
Class = "scsi_host"

  Class Device = "host0"
  Class Device path = "/sys/devices/pci0000:00/0000:00:02.2/0000:03:00.0/host0/scsi_host/host0"
    active_mode          = "Initiator"
    firmware_revision    = "6.34"
    host_busy            = "0"
    hp_ssd_smart_path_status= "HP SSD Smart Path enabled"
    proc_name            = "hpsa"
    sg_prot_tablesize    = "0"
    sg_tablesize         = "543"
    state                = "running"
    transport_mode       = "performant"
    unchecked_isa_dma    = "0"
    unique_id            = "27"
    use_blk_mq           = "0"
    Device = "host0"
    Device path = "/sys/devices/pci0000:00/0000:00:02.2/0000:03:00.0/host0"
      uevent               = "DEVTYPE=scsi_host"
```

Ovdje primjerice vidimo:

- Koju inačicu *firmware*-a ima naš RAID disk kontroler (`firmware_revision = "6.34"`).
- Koji upravljački program (*kernel* modul) je zadužen za rad našeg disk kontrolera (`proc_name= "hpsa"`).
- I druge informacije, poput toga ima li naš disk kontroler u kombinaciji s trenutnim upravljačkim programom (`hpsa`) podršku za *Block Multi Queue* tehnologiju (`use_blk_mq = "0"`) i slično.

Sada pogledajmo koje sve USB uređaje (i kontrolere) imamo na sustavu, sa sljedećom naredbom:

```
systool -b usb
```

A sada od navedenih, koje smo dobili, možemo dobiti detaljne informacije o USB uređaju (`usb1`), koji nas je i zanimao:

```
systool -b usb -d usb1 -v
```

Možemo i zatražiti informacije o određenom kernel modulu i njegovim parametrima rada (pr. za kernel modul imena: `nvme`)

```
systool -vm nvme
```

**Izvori informacija: (489),(490), `man systool`.**

---

### 13.9.2.2. Direktorij `/tmp` i privremene datoteke

Direktorij `/tmp` ne koristi se od strane posebnih datotečnih sustava već je njegova namjena malo drugačija. Naime u inicijalnom trenutku tijekom pokretanja sustava *montira* se korijenski to jest takozvani „*root*“ odnosno **/** direktorij na kojem se nalazi cijeli Linux operativni sustav sa stablom poddirektorija i datoteka u njemu. Pri tome vršni direktorij **/** predstavlja korijen stabla direktorija cijelog operativnog sustava. Unutar tog stabla direktorija, nalazi se i direktorij `/tmp` s posebnom namjenom. Naime programi i sam sustav koriste i `/tmp` i `/var/tmp` direktorije za privremenu pohranu podataka. Međutim, ključna razlika među njima je u tome koliko dugo će podaci pohranjeni unutar ovih direktorija biti sačuvani. Razdoblje zadržavanja podataka za `/var/tmp` mnogo je dulje nego za `/tmp` direktorij. Prema zadanim postavkama, sve datoteke i podaci koji se pohranjuju u `/var/tmp` žive do 30 dana. dok se u `/tmp` podaci automatski brišu nakon deset dana, kako je definirano u konfiguracijskoj datoteci: `/usr/lib/tmpfiles.d/tmp.conf`. Nadalje, sve privremene datoteke koje su pohranjene u direktoriju `/tmp` uklanjaju se odmah nakon ponovnog pokretanja sustava. Direktorij `/tmp` koriste i sustav i pojedini programi za privremeno spremanje datoteka, koje će mehanizmi sustava obrisati: nakon restarta računala, ali i periodički.

Za tu namjenu, za ***RedHat/Centos 6.x.*** se jednom dnevno pokreće skripta `/etc/cron.daily/tmpwatch` koja čisti (briše) datoteke iz `/tmp` direktorija. Međutim za **RedHat/Centos 7.x/8.x.+** to malo drugačije jer to odrađuje ***systemd*** servis imena **`systemd-tmpfiles-clean`** čija konfiguracija rada se nalazi u datoteci: `/usr/lib/tmpfiles.d/tmp.conf`.



U direktorij `/tmp` i sami možete snimati privremene datoteke, koje će sustav s navedenim mehanizmima automatski, periodički brisati, pa toga trebate biti svjesni!

Međutim ponekad određeni korisnici neoprezno spremaju svoje privremene datoteke u `/tmp` direktorij. Stoga je dobro povremeno provjeriti sve datoteke unutra njega kojima nije pristupano duže vrijeme i prema potrebi ih obrisati kako ne bi nepotrebno zauzimali prostor na disku. Datoteke u ovom direktoriju, čini vlasnik nije ***root***, a kojima nije pristupano zadnjih pet dana, možemo pronaći na sljedeći način (a kasnije se možete odlučiti za njihovo brisanje):

```
find /tmp -type f \( ! -user root \) -atime +5
```

**Privremene datoteke i direktoriji**

U nekim slučajevima moguće je da imate potrebu sami kreirati privremenu datoteku, primjerice unutar neke *shell* skripte i slično. Privremena datoteka stvorena pomoću posebne *tmpfile* metode, upotrebom naredbe `mktemp`, automatski se briše kada program koji ju koristi završi s radom ili kada zatvori ovu datoteku. Pogledajmo kako kreirati privremenu datoteku s ovim značajkama:

```
mktemp
/tmp/tmp.BiekYeqIQV
```

Vidimo da se kreirala privremena datoteka `/tmp/tmp.BiekYeqIQV`.

**Drugi slučajevi**

Prema zadanim postavkama, mnogi programi koriste `/tmp` direktorij za privremene datoteke. U nekim slučajevima, te datoteke mogu izbrisati drugi programi s *root* privilegijom. Na primjer, administratori sustava Linux ili UNIX obično pokreću vremenske usluge za uklanjanje datoteka u `/tmp` direktoriju ili će to odraditi slični automatizmi. U svakom slučaju nedostatak ovih privremenih datoteka može spriječiti komunikaciju među programima ili njihov uredan rad. Srećom moguće je privremeno (ili trajno) definirati i neki dugi direktorij koji se može koristiti kao privremeni. Za to se koristi posebna varijabla sustava `TMPDIR`. Kod postavljanja varijable sustava `TMPDIR`, navedite direktorij koji je dostupan korisničkom ID-u koji pokreće određeni program. Pri tome je važno osigurati se da se datoteke ne mogu izbrisati s drugim korisničkim ID-ovima odnosno korisničkim računima. U primjeru ćemo eksportirati ovu varijablu, kako bi ju mogli koristiti svi programi ubuduće:

```
export TMPDIR=/home/test/tempfiles/
```

Ovime smo postavili direktorij **`/home/test/tempfiles/`** kao privremeni (*temp*) direktorij.

Drugi način njene upotrebe bi bio postavljanje ove varijable unutar skripte, trenutak prije pokretanja željenog programa. Na taj način, ona završetkom skripte više ne bi vrijedila, ali bi naš program bio pokrenut s njenim postavkama.

**Izvori informacija:(1070),(1186),(1187),(1188), `man systemd-tmpfiles`,`man tmpfiles.d`,`man mktemp`.**

## 13.9.3. Datotečni sustav `tmpfs`

**`Tmpfs`** je datotečni sustav koji pohranjuje sve datoteke u virtualnoj memoriji. On se ponaša poput RAM diska, dakle sve datoteke i direktoriji koji se kreiraju u njemu, nalaze se zapravo u RAM memoriji. Točnije on sve sprema u internu međumemoriju kernela te se sâm može proširivati (kapacitetom) ili skupljati odnosno smanjivati dinamički, prema potrebi. On je također u mogućnosti koristiti i *swap* disk, ako zatreba. S obzirom na činjenicu kako se on nalazi u *Page cache* memoriji ili prema potrebi na *swap* particiji sav njegov sadržaj je vidljiv isključivo kao *cache* međumemorija. Osim toga, moguće je vidjeti i njegov kapacitet sa standardnim naredbama poput naredbi `df` i `du`. Njega uglavnom koristi kernel, kao interni brzi disk. Dodatno, neke sistemske biblioteke mogu ga koristiti kao dijeljenu memoriju i očekuju kako će se on *montirati* kao poseban uređaj: `/dev/shm`. On nema standardni unos u datoteci `/etc/fstab` jer ga montira direktno kernel. Njegova druga varijanta je ***`devtmpfs`*** koji se koristi na novijim kernelima (od 2019.g.). *Devtmpfs* omogućuje kernelu da kreira *tmpfs* vrlo rano u tijeku inicijalizacije kernela, prije nego što se registrira bilo koji uređaj. Svaki uređaj sa svojim pripadajućim *Major/Minor* oznakama bit će kreiran unutar ovakve *tmpfs* instance. Naime nakon što kernel montira *rootfs (vršni datotečni sustav)*, popunjeni *devtmpfs* će se montirati u `/dev` direktorij. U praksi ih vjerojatno nećete sâmi koristiti, već će ih koristiti komponente operativnog sustava za svoje potrebe.



Pogledajte poglavlje:
**12.6.2. Upotreba RAM diska.**

**Izvori informacija: (862),(863),(1193), `man 5 tmpfs`,`man mount`.**

# 13.10. Provjera diskova, particija i direktorija

Za provjeru slobodnog prostora odnosno kapaciteta diskova i particija ili zauzeća direktorija, koristi se nekoliko naredbi:

- `lsblk` - prikazuje nam logičku podjelu i dostupnost blok uređaja odnosno diskova (engl. *list block [device]*).
- `df` - prikazuje nam koliko prostora imamo zauzeto odnosno slobodno na svim montiranim particijama (*engl. disk free*).
- `du` - prikazuje nam zauzeće prostora za određeni direktorij i/ili njegove pôddirektorije (engl. *disk usage*).

U sljedećim cjelinama upoznat ćemo se detaljnije s navedenim naredbama.

## 13.10.1. Naredba *lsblk*

Naredba `lsblk` daje nam informacije o svim ili samo navedenim blok uređajima odnosno praktično svim diskovima spojenim na naše računalo. Naredba `lsblk` čita ***sysfs*** datotečni sustav radi prikupljanja informacija. Ona prema zadanim postavkama ispisuje sve blok uređaje (osim RAM diskova) u formatu nalik stablu.

Pogledajmo sve diskove na sustavu odnosno logičku sliku diskova i particija uz ostale blok uređaje, pomoću naredbe `lsblk`

```
lsblk

NAME                          MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sr0                            11:0    1 1024M  0 rom
sda                             8:0    0   120G  0 disk
├─sda1                          8:1    0   100G  0 part /
└─sda2                          8:2    0    20G  0 part [SWAP]
```

U ispisu vidimo sljedeće blok uređaje spojene na naše računalo:

- `sr0` uređaj – to je naš DVD/CD-ROM.
- `sda` – ovo je naš SATA tvrdi disk, kapaciteta 120GB koji ima dvije particije:
- `sda1` – ovo je prva odnosno sistemska particija na disku `sda` koja je montirana kao *root* (`/`) kapaciteta 100 GB.
- `sda2` – ovo je druga particija na disku `sda` koja je prema vrsti *swap* particija, kapaciteta 20GB.

Sama naredba `lsblk` ima dosta opcija (prekidača), od kojih ćemo spomenuti prekidač `-f` koji će nam prikazati detalje i o datotečnom sustavu (`f`=*File System*), ali i ***UUID*** jedinstvene oznake odnosno identifikatore particija:

```
lsblk -f

NAME   FSTYPE LABEL UUID                                 MOUNTPOINT
sda
├─sda1 ext4         48edf714-2642-4cb2-82ac-b62043ec1055 /
└─sda2 swap         6c82a0a2-964a-4e94-a326-b1c75bcaec57 [SWAP]
```

Vidljivo je da je je disk `sda` podijeljen na dvije particije: `sda1` i `sda2` te kako niti jedna od njih nema nazive (`LABEL`), a vidljivi su `UUID`-i (jedinstveni identifikatori svake particije) te u što je *montirana* određena particija (pod: `MOUNTPOINT`).

Pogledajmo detaljniji opis stupaca koje smo dobili u našem ispisu:

- `NAME` - daje nam ime blok uređaja odnosno diska (u našem slučaju `sda1` znači `/dev/sda1`).
- `FSTYPE` - daje nam vrstu datotečnog sustava s kojim je particija formatirana, primjerice: ***ext2/3/4, XFS, ZFS***, …
- `sda1` - formatirana je s ***ext4*** datotečnim sustavom i ovo je sistemska particija (na nju je instaliran Linux).
- `sda2` - ovo je *swap* particija koja koristi *swap* datotečni sustav s kojim je formatirana.
- `LABEL` - ovo je oznaka particije. Naime to je proizvoljno opisno ime koje može i ne mora biti upisano.
- `UUID` - jedinstveni je identifikator svake particije koji se zapisuje na particiju, može se vidjeti i s naredbom: `blkid`. ***UUID*** se može koristiti i kao identifikator particije (umjesto ***/dev/****XY*) u datoteci `/etc/fstab`.



Pogledajte poglavlje:
**13.8. Što je datoteka /etc/fstab.**

- `MOUNTPOINT` - označava u što je *montirana* particija. Primjerice particija `sda1` je montirana u `/` koji označava vršni datotečni sustav na koji je instaliran cijeli Linux, dok je `sda2` *SWAP* particija pa nema točku montiranja.



Za više informacija o montiranim datotečnim sustavima, i opcijama kod montiranja, pogledajte naredbu: `findmnt`.

Naredba `lsblk` dolazi u softverskom (*RPM*) paketu imena: **`util-linux`**.

**Izvor informacija: (804), `man lsblk`,`man blkid`,`man findmnt`,`man fstab`.**

### 13.10.2. Naredba *df*

Naredba `df` prikazuje nam zauzeće datotečnog sustava u koji smo *montirali* neki direktorij. Točka montiranja (*mount point)* mora postojati, kako bi ova naredba prikazala podatke za određenu particiju. Ona standardno ispisuje zauzeće u blokovima podataka od 1KB, ali i to možemo promijeniti. Ova naredba u računicu uzima i metapodatke s najniže razine datotečnog sustava (pr. *disk/inode maps*, *inode*,…). Navedena naredba ima veći broj prekidača od kojih ćemo koristiti njih nekoliko:

- `-h` ili `-g` daju nam prikaz u GB, dok je prekidač `-h` takozvani „*Human Readable format*" (prikaz u ***kB***, ***MB*** i ***GB***).
- `-i` daje prikaz prema *i-node* brojevima (*iskorišteno/slobodno*), `-T` daje prikaz uz ispis o vrsti datotečnog sustava.

**Krenimo s primjerima.**
**1.Ispišimo zauzeće *montiranih* particija:**

```
df

Filesystem     1K-blocks   Used     Available Use%    Mounted on
/dev/sda1      100158332   2751504  97406828  2.7%    /
```

Vidljivo je da nam je prikaz u blokovima od 1KB pomalo nepraktičan. Ovdje vidimo particiju, zauzeće i točku montiranja.

**2. Ispišimo zauzeće u nama (ljudima) malo ljepšem formatu (`-h`).** Ovdje vidimo iskorišteno: `Used` i slobodno: `Avail`:

```
df -h

Filesystem      Size      Used  Avail   Use%   Mounted on
/dev/sda1       100G      2.7G  97.3G   2.7%   /
```

**3. Prijašnjem primjeru dodajmo i ispis vrste datotečnog sustava (`-T`).** Sada vidimo i datotečni sustav: `Type` i to: `ext4`:

```
df -Th

Filesystem  Type    Size     Used   Avail   Use%   Mounted on
/dev/sda1   ext4    100G     2.7G   97.3G   2.7%   /
```

**4. Sada provjerimo zauzeće datotečnog sustava kojem pripada određeni direktorij (`/home`).**

```
df -hT /home/

Filesystem     Type   Size   Used Avail Use% Mounted on
/dev/sda1      xfs     14G   3.6G  9.5G  28% /
```

**5. Sada provjerimo zauzeće *inode* unosa (`-i`).** Potom vidimo iskorištene: `IUsed` i slobodne: `IFree` *inode* unose:

```
df -i
Filesystem                Inodes   IUsed      IFree IUse%  Mounted on
/dev/sda1               30715904  140393   30575511    1%  /
```



*Ako je neka (velika) datoteka tek obrisana, a određeni proces (program) drži njen file deskriptor, sve dok ga on ne oslobodi, veličina te datoteka biti će prijavljena kao alocirana odnosno da se koristi i zauzima disk. To se primjerice može dogoditi tijekom nadogradnje softvera, jer određeni procesi mogu još uvijek držati stare datoteke (pr. biblioteke).*
*Takve datoteke imaju oznaku* **DEL**, *ako ih gledamo s programom* `lsof`, *odnosno možemo ih pronaći na sljedeći način:*

```
lsof | grep DEL
```



Za više detalja o ***inode*** unosima pogledajte poglavlje:
**4.5. Datotečni sustav detaljnije.**



Pogledajte i poglavlje vezano za prorijeđene (*sparse*) datoteke i ovu naredbu:
**4.6. Prorijeđene (*Sparse*) datoteke.**

Naredba `df` dolazi u softverskom (*RPM*) paketu imena: **`coreutils`**.

**6. Sada provjerimo zauzeće svih datotečnih i pseudo datotečnih sustava (`-a`).**

```
df -ah
Filesystem      Size  Used Avail Use% Mounted on
sysfs              0     0     0    - /sys
proc               0     0     0    - /proc
devtmpfs        972M     0  972M   0% /dev
securityfs         0     0     0    - /sys/kernel/security
tmpfs           990M     0  990M   0% /dev/shm
devpts             0     0     0    - /dev/pts
tmpfs           990M  8.5M  982M   1% /run
tmpfs           990M     0  990M   0% /sys/fs/cgroup
cgroup             0     0     0    - /sys/fs/cgroup/systemd
/dev/sda1        14G  3.6G  9.5G  28% /
selinuxfs          0     0     0    - /sys/fs/selinux
debugfs            0     0     0    - /sys/kernel/debug
hugetlbfs          0     0     0    - /dev/hugepages
tmpfs           198M     0  198M   0% /run/user/0
```

**Izvori informacija: (K-14), `man df`,`man lsof`.**

## 13.10.3. Naredba *du*

Naredba `du` je standardna Linux/Unix naredba koja korisniku omogućuje brzo dobivanje informacija o iskorištenju diska. Ona se najbolje primjenjuje na određene direktorije te omogućuje mnoge varijacije za prilagodbu izlaza u skladu s vašim potrebama. Kao i kod većine naredbi, korisnik može koristiti mnoge opcije ili prekidače. Također, kao i mnoge Linux naredbe, većina korisnika koristi samo dvije ili tri opcije kako bi zadovoljile svoje specifične potrebe. Mi ćemo vam predstaviti osnovne opcije koje se najčešće koriste, ali i pogledati neke koje su manje uobičajene u nadi da ćemo poboljšati našu upotrebu naredbe `du`.
Dakle naredba `du` (Engl. *Disk usage*) daje nam ispis zauzeća diskovnog prostora za određeni direktorij (mapu) odnosno sve njegove pôddirektorije rekurzivno, pa čak i za one koji su eventualno točke montiranja na drugi datotečni sustav.

Neke od opcija ove naredbe su:

- `-a` - ispisuje zauzeće svih datoteka (unutar svih poddirektorija u kojima se nalazimo).
- `-h` - format lakši za razumijevanje jer prikazuje oznake: kB, MB, GB ili TB (Engl. *Human Readable*).
- `-s` - sumiran ispis bez ispisivanje svake datoteke i poddirektorija zasebno već sumirano (Engl. *Summary*).
- `-x` - ne prelazi u drugi datotečni sustav, ako unutar trenutnog stabla direktorija imamo točku montiranja na njega.
- `--apparent-size` - prikazuje i veličinu prorijeđenih (*Sparse*) datoteka, ako se koriste to jest, ako postoje.



Vezano za prorijeđene datoteke, pogledajte poglavlje:
**4.6. Prorijeđene (*Sparse*) datoteke.**

**1.** Ispišimo samo sumirano (`-s`) zauzeće cijelog direktorija `/home/` i svih njegovih pôddirektorija zajedno. S ovime ćemo dobiti podatke o tome koliko svi korisnici imaju u svojim "*home*" direktorijima, ali lijepo formatirano (`-h`).
Ova operacija može potrajati, ovisno o količini pôddirektorija i datoteka. Pokrenimo ju na sljedeći način:

```
du -hs /home/
54M     /home/
```

Vidimo da je ukupno zauzeće svih direktorija i pôddirektorija koji se nalaze unutar vršnog direktorija `/home/` **54MB**.

**2.** Kombinacijom naredbi `find` i `du`, pronađimo najveće datoteke unutar vršnog direktorija `/usr/` i svih pôddirektorija ispod:

```
find  /usr/ -type f -exec du -Sh {} + | sort -rh | head -n 4
102M    /usr/lib/locale/locale-archive
70M     /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-3.b13.el7_5.x86_64/jre/lib/rt.jar
59M     /usr/lib64/libwireshark.so.3.1.3
```

Ispis smo sortirali s naredbom `sort`, a filtrirali smo prve četiri (4) pronađene datoteke s naredbom `head`.

Naredba `du` dolazi u softverskom (*RPM*) paketu imena: **`coreutils`**.

**Izvori informacija: (K-14), `man du`,`man find`.**

### 13.10.4. Naredba *hdparm*

Naredba `hdparm` iz naredbenog retka pruža nam pristup sučelju kernela koje podržava ATA, SATA ili SAS diskove. Točnije omogućava nam pristup Linuxovom SATA/PATA/SAS "***libata***" podsustavu te starijem IDE podsustavu upravljačkih programa. Čak i mnogi noviji diskovni USB pogoni (od 2008. godine) sada također podržavaju takozvano "***SAT***" sučelje (engl. *SCSI-ATA Command Translation*) preko kojeg se USB pogonima (diskovima) također može pristupati s programom `hdparm`. Dakle on nam osim mogućnosti dohvaćanja raznih statistika o spojenim diskovnim uređajima, daje i mogućnost njihove detaljnije konfiguracije i optimizacije parametara rada, kao što su primjerice:

- Ispis geometrije i inačice *firmware*-a diska (`-I`).
- Načina rada: PIO, DMA (`-d1`), UDMA, *Native Command Queueing* (***NCQ***), …
- Upotrebe međumemorije za čitanje i/ili zapisivanje na površinu diska [pr. za *read-ahead* (`-A1`)].
- Postavljanje vrijednosti akustičnog menadžmenta (`-M XY`)
- Testiranje brzine rada međumemorija (*cache* i *buffer*) za čitanje na disku (`-t` i `-T`).
- Definiranja broja višestrukih I/O sektora (`-m`) odnosno sposobnosti rada s više sektora odjednom, tijekom upotrebe jednog signala prekida (IRQ).
- Uključivanja štednje energije to jest gašenja diska nakon određenog vremena neaktivnosti: (`-S XY`)
- … i desetaka drugi parametara i opcija.

Pogledajmo naš SATA *SSD* disk s osnovnim prekidačem `-i`

```
hdparm -i /dev/sda

/dev/sda:
 Model=KINGSTON SHFS37A120G, FwRev=603ABBF0, SerialNo=50026B7257047ABE
 Config={ HardSect NotMFM HdSw>15uSec Fixed DTR>10Mbs RotSpdTol>.5% }
 RawCHS=16383/16/63, TrkSize=0, SectSize=0, ECCbytes=4
 MaxMultSect=1, MultSect=1
 CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAsects=234441648
 IORDY=on/off, tPIO={min:120,w/IORDY:120}, tDMA={min:120,rec:120}
 PIO modes:  pio0 pio1 pio2 pio3 pio4
 DMA modes:  mdma0 mdma1 mdma2
 UDMA modes: udma0 udma1 udma2 udma3 udma4 udma5 *udma6
 AdvancedPM=yes: WriteCache=enabled
```

U osnovnom ispisu vidimo bazične statistike poput modela diska (`KINGSTON SHFS37A120G`), inačice *firmware*-a (`603ABBF0`), informacije da je međumemorija na samom disku za zapisivanje aktivirana (`WriteCache=enabled`) i slično. Dok s upotrebom prekidača `-I` dobivamo sve već navedeno, ali i napredne statistike, poput prikaza mogućnosti našeg diska; poput:

```
hdparm -I /dev/sda

Commands/features:
        Enabled Supported:
           *    SMART feature set
                Security Mode feature set
           *    Power Management feature set
           *    Write cache
                Look-ahead
           *    Host Protected Area feature set
           *    WRITE_BUFFER command
           *    READ_BUFFER command
           *    NOP cmd
           *    DOWNLOAD_MICROCODE
           *    Advanced Power Management feature set
           *    48-bit Address feature set
           *    Gen2 signaling speed (3.0Gb/s)
           *    Gen3 signaling speed (6.0Gb/s)
           *    Native Command Queueing (NCQ)
           *    WRITE BUFFER DMA command
           *    READ BUFFER DMA command
           *    Data Set Management TRIM supported (limit 1 block)
```

**U ovom skraćenom ispisu vidimo koje značajke diska su aktivirane (`*` ispod `Enabled`) odnosno koje sve značajke uređaj podržava, ali trenutno nisu aktivirane (bez znaka `*` ispod stupca `Enabled`).**

**Izvori informacija: (1245),(1246),(1247), `man hdparm`.**

# 13.11. RAID polja diskova

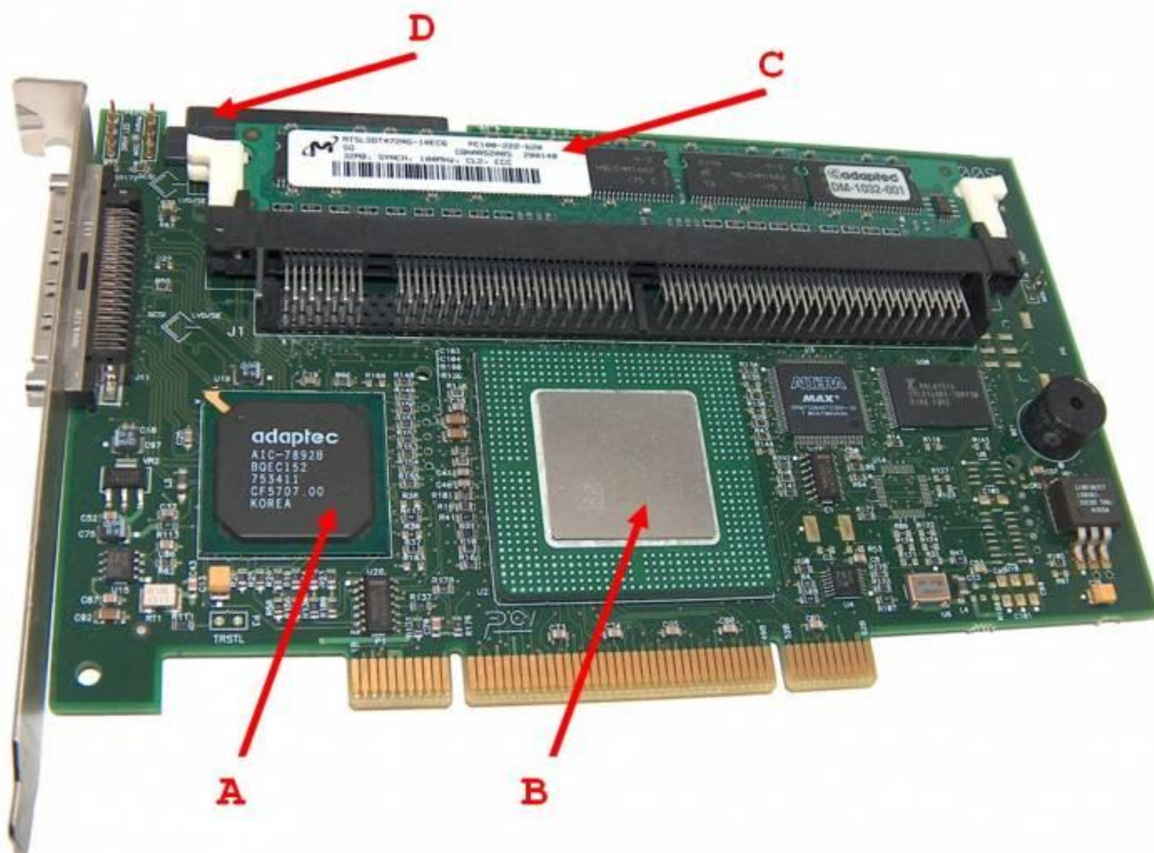
U slučajevima kada želimo postići veću sigurnost, skalabilnost ili brzinu pisanja ili čitanja na tvrdi disk ili SSD disk, potrebno je koristiti tehnologije u kojima se koristi više fizičkih diskova u istovremenom radu. Povezivanja više fizičkih diskova u neki logički disk odnosno polje diskova, moguće je pomoću takozvanih *RAID* tehnologija ili tzv. “*Volume Manager-a*”.
*RAID* je skraćenica od redundantnog polja nezavisnih diskova (Engl. *Redundant Array of Independent Disks*).

**Kako radi RAID tehnologija?**
Podaci se zapravo zapisuju (i čitaju) na više diskova paralelno odnosno istovremeno, na način koji je definiran u konkretnom RAID polju koje smo odabrali prilikom inicijalizacije RAID polja. Postoji nekoliko standardnih RAID polja od kojih svako ima neke prednosti i mane. Osim toga postoje dvije mogućnosti kreiranja RAID polja:

- Softverski, na razini operativnog sustava ili na razini upravljačkog programa (drivera).
- Hardverski, za što je potreban hardverski RAID kontroler.

Softversko RAID polje se kreira iz operativnog sustava, a hardversko iz RAID kontrolera koji je obično implementiran kao posebna kartica poput ove na slici **116**.

**Slika 116. RAID kontroler tvrtke *Adaptec* https://www.adaptec.com (Adaptec 2100S).**




Slika **116** prikazuje hardverski RAID kontroler za spajanje ***SCSI*** tvrdih diskova pomoću 68 *pinskog* konektora/kabela, preko kojeg je moguće spojiti do 15 tvrdih diskova. Ovaj RAID kontroler se na matičnu ploču spaja preko **PCI** sabirnice.
Na njemu je vidljivo sljedeće:

- (**A**) - ***SCSI*** disk kontroler (AIC-7892B).
- (**B**) - CPU i RAID ***ASIC*** koji (RAID ASIC) je zadužen za kreiranje RAID polja i druge operacije potrebne za RAID.
- (**C**) - RAM Memorija kao predmemorija za „*cache*“ podataka (u gornjem djelu kartice) (32MB PC-100 ECC - proširivo).
- (**D**) - 68*pin*ski SCSI konektor za spajanje SCSI tvrdih diskova.



Upotreba **RAID** tehnologije osigurava visoku dostupnost diskovnog polja odnosno podataka na njemu.
Za visoku dostupnost sustava, te što ona znači, pogledajte poglavlje: **26.8.4. Visoko dostupni sustavi (*High Availability*).**

**Izvori informacija: (677),(K-12)**

## 13.11.1. Hardverski RAID

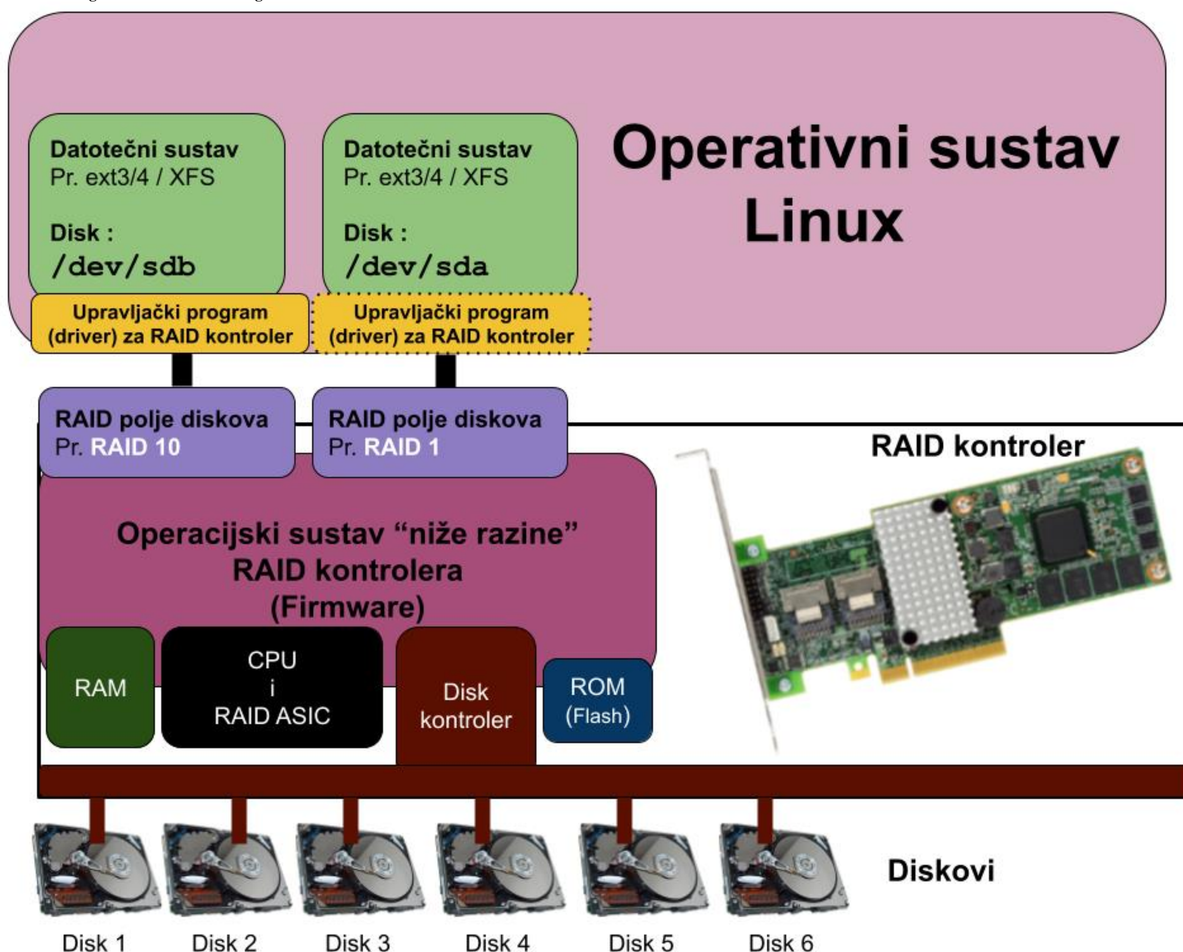
**U slučaju upotrebe hardverskog RAID kontrolera** prilikom uključivanja računala, nakon što se inicijalizira BIOS na matičnoj ploči, inicijalizira se i BIOS RAID kontrolera. Tada je potrebno ući u BIOS RAID kontrolera te kreirati željeno RAID polje (ili više njih) s dostupnim tvrdim diskovima spojenim na RAID kontroler. Ako raspolažemo s većim brojem tvrdih diskova moguće je kreirati više istovremenih RAID polja odnosno prema RAID terminologiji: ***Logičkih*** diskova.
U trenutku nakon što snimimo konfiguraciju RAID polja koja smo kreirali, RAID kontroler počinje s inicijalizacijom RAID polja i to: u potpunosti ili sâmo djelomično.
Ako je odabrana potpuna inicijalizacija RAID polja, to može trajati i satima; ovisno o RAID polju te broju i brzini tvrdih diskova.
U slučaju kada se RAID polja ne moraju potpuno inicijalizirati, a što je slučaj kod novijih RAID kontrolera, odmah nakon što snimimo konfiguraciju novog RAID polja možemo restartati računalo.
Potom će BIOS-u računala biti prijavljen po jedan logički disk za svako RAID polje koje smo kreirali.

RAID polja će se kasnije u pozadini inicijalizirati nakon pokretanja operativnog sustava, a ti logički diskovi će nam biti vidljivi kao običan fizički tvrdi disk; po jedan disk za svako RAID polje koje smo kreirali.
Svaki od tih nama vidljivih diskova možemo normalno particionirati i formatirati.

U slučaju kada se *RAID* polje mora inicijalizirati unutar *BIOSa* RAID kontrolera, potrebno je pričekati kako bi se cijeli proces završio, a što može potrajati satima ovisno o veličini i vrsti RAID polja.
U ovom slučaju, nakon inicijalizacije RAID polja, računalo se restarta, svako RAID polje (*logički disk*) se prijavljuje *BIOSu* matične ploče, te je vidljivo operativnom sustavu kao običan fizički tvrdi disk (isto kao u gornjem slučaju).

**Detaljniji pogled na RAID kontroler i RAID sustav**
Pogledajmo prvo logičku shemu hardverskog RAID kontrolera na slici 117.

**Slika 117. Logička shema hardverskog RAID kontrolera.**




Kako je vidljivo, *RAID* kontroler ima svoj centralni procesor (**CPU**) te specijalizirani procesor (**RAID** ASIC), a obično su obje funkcionalnosti integrirane u jednom integriranom sklopu (*čipu)*. Specijalizirani (dio) procesora se zove ***RAID ASIC*** i on je zadužen za RAID polja. Dakle svo zapisivanje ili čitanje na RAID polje sa strane operativnog sustava, gledano od vrha slike, odnosno zapisivanje na fizičke diskove (na slici prema dolje), kao i sve potrebne kalkulacije poput izračunavanja pariteta kod RAID 5 ili RAID 6 polja odrađuje ***RAID ASIC***. RAID kontroler ima i svoju **ROM** i **RAM** memoriju kao i disk kontroler na koji se zapravo spajaju svi diskovi. Važno je znati da RAID kontroler ima svoj minijaturni operativni sustav niže razine, a koji je pohranjen u ROM memoriji RAID kontrolera. Unutar tog minijaturnog operativnog sustava se nalaze metode za pristup svim diskovima spojenim na njega kao i sve što je potrebno za kreiranje i rad RAID polja unutar kojeg se nalaze diskovi.

Možemo promatrati RAID kontroler i kao minijaturno računalo, jer on ima:

- Svoj centralni procesor (CPU) i specijalizirani CPU za RAID polja.
- RAM memoriju.
- Disk kontroler za spajanje svih diskova.
- Flash (ROM) memoriju.
- Neku varijantu minijaturnog operativnog sustava.

Sada promatrajmo diskove koji su u konačnici spojeni na RAID kontroler kao na jednu komponentu koja komunicira s ostatkom računala preko operativnog sustava Linux i to preko upravljačkog programa (Engl. *Driver*) za naš RAID kontroler.
Operativni sustav Linux preko upravljačkog programa za naš RAID kontroler vidi sâmo polja diskova koja su kreirana od strane RAID kontrolera i to svako RAID polje odnosno logički disk, kao jedan zaseban disk. Ako smo kreirali RAID 10 polje unutar kojeg se nalaze četiri (4) tvrda diska (*disk 1* do *disk 4*) operativni sustav Linux vidjet će jedan jedini tvrdi disk, koji je zapravo cijelo RAID 10 polje diskova. U primjeru će RAID 10 polje biti vidljivo kao disk: `/dev/sdb`.
Kreirali smo još jedno polje diskova koje je RAID 1 polje, u kojem su dva diska (*disk 5* i *disk 6 na* slici).
I ovo polje diskova prezentirano je operativnom sustavu kao jedan disk, konkretno vidljivo kao disk: `/dev/sda`.
Svaki od tih (logičkih) diskova se s točke operativnog sustava sastoji od svih dijelova od kojih se sastoji bilo koji "normalan" tvrdi disk: staze (trake), cilindri, klasteri i sektori. On se nadalje ponaša kao klasičan blok uređaj odnosno tzv. *Block Device*.
Zbog toga je svaki taj logički disk potrebno particionirati te kasnije formatirati s nekim datotečnim sustavom.
Sâm sustav kao i svi programi nadalje koriste taj datotečni sustav za pohranjivanje datoteka, kao da se radi o običnom tvrdom disku. Za sve funkcionalnosti RAID polja kao i diskova koji su unutar tih polja, brine se RAID kontroler. Stoga je važno da je RAID kontroler koji ste odabrali, kao i njegov operativni sustav (*Firmware*) te pripadajući upravljački program, najbolji mogući odnosno provjeren i dokazan u dugotrajnom i stabilnom radu.

## RAM memorija na RAID kontroleru

Kao što je vidljivo iz obje slike (116 i 117) RAID kontrolera, na svakom RAID kontroleru nalazi se i brza RAM memorija koja se koristi kao brzi međuspremnik između RAID kontrolera i sâmih diskova u određenom RAID polju. Ova RAM memorija stoga mora biti vrlo pouzdana. Zbog toga se u RAID kontrolere ugrađuje isključivo vrsta memorije koji ima automatsku korekciju greške odnosno *ECC* (engl. *Error Correcting Code*) memorija. Pri tome postoji nekoliko varijanti *ECC* memorija:

- One koje dozvoljavaju grešku na bilo kojem bîtu memorije u bilo kojem memorijskom sklopu (*čipu).*
- Ili one koje dozvoljavaju ispad (kvar) bilo kojeg memorijskog sklopa (*čipa).*

Ovakve memorije se inače ugrađuju i u poslužitelje. *ECC* memorije možemo prepoznati i po tome što imaju neparan broj memorijskih sklopova (*čipova)*. *ECC* memorija na slici 118 ih ima devet, dakle stvarni ukupni kapacitet ove memorije je jednak zbroju kapaciteta osam memorijskih *čipova*.

**Slika 118. RAM memorija PC-100 ECC, kapaciteta 128MB, tvrtke Kingston: http://www.kingston.com (kompatibilnu s RAID kontrolerom Adaptec 2100S, na slici 116)**




Ova brza memorija, osim što drastično ubrzava operacije prema RAID kontroleru odnosno RAID polju diskova, uvodi i jedan problem. Što će se dogoditi, ako je naš operativni sustav poslao na RAID kontroler na zapisivanje određene podatke koje je RAID kontroler prvo spremio u ovu brzu RAM memoriju, ali trenutak prije nego je podatke stvarno počeo zapisivati na diskove, nestalo je električne energije. U ovom slučaju ostati ćemo bez tih podataka koji su bili u RAM memoriji RAID kontrolera. Ima li pomoći?.
Naravno, rješenje je opcija koja je dostupa za sve novije RAID kontrolere, a to je upotreba vrlo slične *ECC* RAM memorije, ali koja na sebi ima punjivu (akumulatorsku) bateriju ili posebne kondenzatore. U slučaju nestanka električne energije ova baterija čuva stanje RAM memorije satima, sve dok se računalo/poslužitelj ponovno ne pokrene.

Tada RAID kontroler sve podatke iz ove ECC RAM memorije, a zahvaljujući njenoj bateriji, uredno zapisuje na tvrde diskove u RAID polju na koje su se podaci inicijalno i trebali zapisati. Ova opcija za RAM memoriju RAID kontrolera se zove ***BBU*** (Engl. *Battery Backup Unit*). Ovakva memorija s baterijom je vidljiva na slici 119.

**Slika 119. Izgled ECC RAM memorije sa BBU baterijom, za RAID kontroler tvrtke *LSI Logic:* http://www.lsi.com.**



**Slijedi napredna cjelina!**
**Linux i hardverski RAID kontroleri.**
Inicijalna konfiguracija i inicijalizacija RAID polja se u pravilu radi iz *BIOSa* sâmog RAID kontrolera, ali je moguće neke od opcija konfigurirati i naknadno iz posebnog softvera koji je specifičan za pojedini RAID kontroler ili za proizvođača. Tako recimo za RAID kontrolere koje nalazimo u poslužiteljima tvrtke ***Hewlett Packard*** (**HP**) možemo instalirati softverski paket s kojim dobivamo program s kojim možemo konfigurirati njihove RAID kontrolere, ali i čitati razne statistike rada kontrolera ili spojenih diskova. Taj softverski paket se naziva: *HP Smart Storage Administrator (HP SSA) CLI* (2014.g.+) odnosno sâma naredbe je `ssacli`. Mi ćemo za primjer instalirati trenutnu inačicu ovog softvera na sljedeći način (za trenutnu aktivnu inačicu **4.15**). U jednom koraku ćemo kopirati ovaj softverski paket i instalirati ga (naredba koja slijedi je u jednom retku):

```
rpm -ivh https://downloads.hpe.com/pub/softlib2/software1/pubsw-linux/p1944765023/v165907/ssa-4.15-6.0.x86_64.rpm
```

U ovoj cjelini pokazat ćemo vam sâmo nekoliko mogućnosti ovog alata koji direktno pristupa i konfigurira sâm RAID kontroler. Pošto smo govorili o RAM memoriji na RAID kontroleru odnosno ***BBU*** vrsti memorije s baterijom, pogledajmo prvo kako provjeriti je li ona uključena odnosno aktivirana za upotrebu (filtrirali smo ispis i dodali svoje komentare →):

```
ssacli ctrl all show config detail | grep -i -e cache -e battery -e slot
```

```
Slot: 0                                  → identifikator utora u kojem se nalazi RAID kontroler
 Cache Status: OK                        → Status BBU RAM memorije je dobar (OK)
 Drive Write Cache: Disabled             → Ova memorija se trenutno ne koristi (Disabled) za zapisivanje (Write)
 Total Cache Size: 512 MB                → Veličina BBU RAM memorije je 512MB
 No-Battery Write Cache: Disabled        → U slučaju da BBU baterija ne radi sustav će isključiti upotrebu ove memorije
 Cache Backup Power Source: Capacitors   → Vrsta baterije na memoriji je kondenzatorska (Capacitors)
Battery/Capacitor Status: OK             → Status baterije/kondenzatora je dobar (OK)
```

Status baterije na **BBU** memoriji je dobar i imamo zaštitni mehanizam, koji u slučaju da baterija ne radi, isključuje upotrebu ove memorije za ubrzavanje operacija zapisivanja na disk. Stoga ju možemo aktivirati, na sljedeći način, pri čemu `slot=0` označava utor u kojem se koristi naš RAID kontroler, a `dwc` (*Drive Write Cache)* označava memoriju za operacije zapisivanja:

```
ssacli ctrl slot=0 modify dwc=enable
```

I potom ćemo morati potvrditi ovu radnju sa (`y`). Nakon toga imamo uključenu upotrebu ove brze RAM memorije s baterijom na RAID kontroleru, za sve operacije zapisivanja podataka na diskove u RAID polju (poljima) našeg RAID kontrolera.

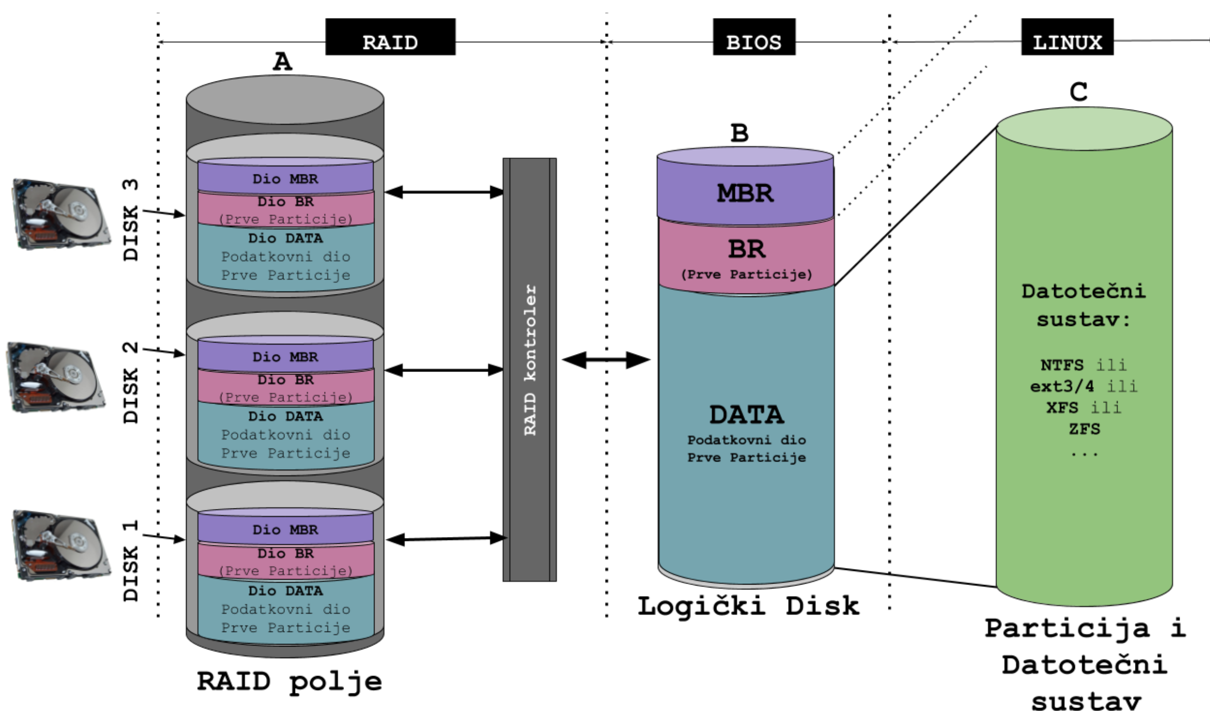
**Izvori informacija: (677),(850),(851), (K-12), `man ssacli`.**

---

## 13.11.1.1. Pogled na nižu razinu diska

**Slijedi napredna cjelina!**
Sada ćemo se spustiti jednu razinu niže. Pogledajmo stanje u kojem koristimo RAID kontroler s jednim RAID poljem koje predstavlja jedan logički disk, na kojem je kreirana jedna particija s ***MBR*** shemom. Ova ***MBR*** shema je vidljiva na slici **120.**

**Slika 120. Pogled na logičke razine: od RAID kontrolera, preko logičkog diska do datotečnog sustava.**




Sa strane RAID kontrolera (pogled **A**), vidljivi su svi diskovi u RAID polju. U RAID polju se podaci raspoređuju po svim diskovima, ovisno o sâmom RAID polju. Tako je i sâm ***MBR*** raspoređen (kopiran) na više diskova u RAID polju. Isto se događa i s ***BR*** i svim podacima (DATA dio) odnosno dijelom diska koji se formatira sa željenim datotečnim sustavom.

Nadalje RAID kontroler prijavljuje *BIOS*u računala cijelo RAID polje kao jedan (običan), ali s točke RAID kontrolera je to logički disk (pogled **B**). Ako sve pogledamo s točke tog logičkog diska (pogled **B**) on se ponaša kao običan disk koji se sastoji od ***MBR*a**, ***BR*a** i dijela za podatke (DATA dio) odnosno podataka na disku koji čini datotečni sustav s datotekama i direktorijima. Pogledamo li ipak sve iz točke operativnog sustava, sa strane korisnika (pogled **C**), dostupna nam je particija diska (DATA dio), koju trebamo formatirati s datotečnim sustavom po želji: ***NTFS***, **ext2**/**3**/***4***, ***XFS,…*** koji potom trebamo montirati (*mount*).

---

### 13.11.1.2. Softverski RAID

**Osim upotrebe hardverskih RAID kontrolera moguća je i upotreba softverskih RAID polja unutar Linuxa**.
U slučaju upotrebe softverskog RAID polja koje u Linuxu odrađuje takozvani *LVM2* odnosno *Logical Volume Manager* inačice dva, nakon pokretanja operativnog sustava, pomoću alata koji dolaze sa ***LVM2*** paketom, kreiramo željeno RAID polje na cijelom tvrdom disku ili željenim **particijama**. Nakon što se to polje inicijalizira, možemo ga normalno formatirati sa željenim datotečnim sustavom te dalje koristiti kao da je u pitanju normalna particija na običnom tvrdom, SSD ili nekom drugom disku.
*LVM2* donosi i druge mogućnosti osim sâmog RAID polja, a neke od njih su:

- Mogućnost dodavanja i vađenja diskova u radu; ako to podržava hardver.
- Povećavanje veličine bilo kojeg logičkog diska (engl. *Resize*).
- Izrada sigurnosne kopije (engl. *Backup*) cijelog logičkog diska u radu, bez zaustavljanja, korištenjem metode snimanja stanja u vremenu tzv. *Snapshot* metode.

Dodatna preporuka je definiranje zasebnog tvrdog diska čija je jedina namjena promjena neispravnog diska, kada greška nastupi. Tvrdi disk koji namijenimo za taj zadatak naziva se engl. "***Hot Spare***" disk. Naime u slučaju greške na nekom od diskova automatiziran proces rekreiranja (engl. *Rebuild*) RAID polja kreće s upotrebom ovog *Hot Spare* diska koji nakon završetka procesa postaje funkcionalan disk koji je član RAID polja u kojem je bio neispravan disk. Tada se neispravan disk može izvaditi i zamijeniti s novim diskom. Taj novi zamijenjeni disk (uglavnom[konfigurabilno]) tada postaje novi *Hot Spare* disk.



Upotreba odnosno preporuka upotrebe tzv. ***Hot Spare*** diska vrijedi i za hardverske ***RAID*** kontrolere!.



O softverskoj RAID funkcionalnosti koja je u linuxu implementirana pod nazivom **LVM2**, pogledajte poglavlje: **13.11.2. Softverski RAID (LVM2).**

---

### 13.11.1.3. Koja su najčešća RAID polja u upotrebi i koje su im prednosti i mane?

Danas u upotrebi imamo nekoliko RAID polja od kojih svako ima određene prednosti, ali i neke mane.
U ovom poglavlju upoznat ćemo se s nekim od RAID polja koja su najčešće u upotrebi.

### 13.11.1.4. Standardna RAID polja

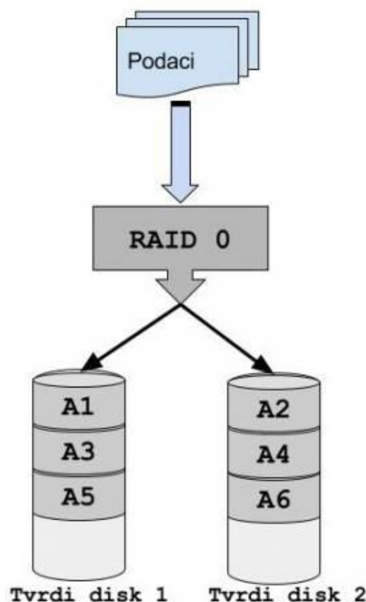
Važno je znati da hardverski ili softverski RAID kontroleri tijekom inicijalizacije RAID polja logičku površinu svih tvrdih diskova u polju dijele na male blokove podataka, pri čemu je veličina tih blokova ovisna o RAID kontroleru i/ili softveru.
Ove male logičke cjeline odnosno blokovi postaju najmanje jedinice za zapis podataka, praktično kao što su *clusteri* tj. blokovi na datotečnom sustavu. Veličina ovih blokova podataka u RAID terminologiji se zove ***Stripe size***. Ovo je važno jer svi diskovi u određenom RAID polju moraju imati istu podjelu na najmanje blokove podataka koji se mogu čitati ili pisati tj. nad njima provoditi operacije zapisivanja ili čitanja.

Najčešća veličina *Stripe size-a* je 64kB, ali ju je moguće i mijenjati, što se ne preporučuje, ako ne znate zašto to radite. Ovu veličinu je moguće mijenjati sâmo prije nego se krene u proces inicijalizacije RAID polja. Standardna RAID polja ćemo upoznati ubrzo.

### 13.11.1.4.1. RAID 0

**RAID 0** polje ili ***Stripe*** polje sastoji se od minimalno dva diska. Ukupan kapacitet ovog polja jednak je zbroju kapaciteta svih diskova u polju. U ovo polje često je moguće dodavati diskove različitih veličina, ali polje će se obično kreirati tako, da će na svim diskovima biti iskorišten sâmo kapacitet od najmanjeg diska. Ako imamo diskove od 120 GB, 320 GB i 500 GB, ukupni kapacitet će biti 120 GB x 3 = 360 GB. Zapisivanje podataka u **RAID 0** polje radi se tako, da se podaci koji se trebaju zapisati, dijele na blokove (pr. 64kB) i pri tome se ti blokovi podataka pravilno raspoređuju te zapisuju na sve diskove u polju.

**Slika 121. RAID 0 polje diskova**




Zamislimo da imamo dva diska kao na slici 121. Podaci se tada raspoređuju i zapisuju: polovina na jedan, a pola na drugi disk i to redom kako dolaze. Ako su recimo blokovi (*Stripe size*) od 64kB na svakom disku kao najmanje jedinice za zapisivanje i imamo 384kB podataka za zapisati, dogodit će se sljedeće:

- Prvih 64kB će biti zapisano na prvi disk (u blok ***A1***).
- Drugih 64kB na drugi disk (u blok ***A2***).
- Trećih 64kB na prvi disk (u blok ***A3***).
- Četvrtih 64kB na drugi disk (u blok ***A4***).
- Petih 64kB na prvi disk (u blok ***A5***).
- Šestih 64kB na drugi disk (u blok ***A6***).

Ovo polje ne daje nam nikakvu sigurnost odnosno redundanciju.
Ako se pokvari bio koji disk u polju, **SVI** podaci će biti izgubljeni.
Najveća prednost ovog polja je u njegovoj brzini i kapacitetu. Tijekom zapisivanja se podaci snimaju na sve diskove paralelno, a isto tako se i čitaju sa svih diskova paralelno. Zbog toga je brzina RAID 0 polja najveća od svih RAID polja: i za čitanje i zapisivanje.
U slučaju kvara bilo kojeg od diskova u ovom polju, gube se **SVI** podaci, pa je prema tome ovo RAID polje najmanje pouzdano od svih.

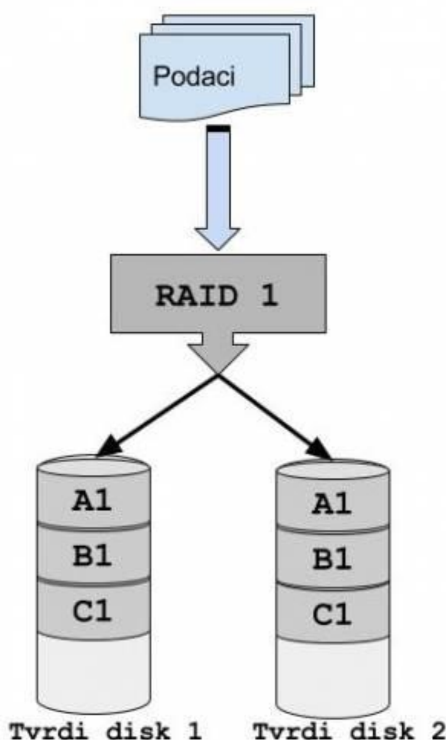
**Izvor informacija: (678)**

---

### 13.11.1.4.2. RAID 1

**RAID 1** ili zrcalno odnosno ***Mirror*** polje sastoji se od minimalno dva diska.

**Slika 122. RAID 1 polje diskova.**




U ovom polju moguće je imati i više od dva diska uz isto ograničenje za kapacitet kao za ***RAID 0*** polje, a kapacitet najmanjeg diska se koristi na svim diskovima. Ovo polje radi identičnu (zrcalnu) kopiju svih podataka s prvog diska na drugi (ili na više njih) i to također kako podaci dolaze, razbijanjem podataka u blokove (engl. *Stripe*).
Ako imamo dva diska u ***RAID 1*** polju i zapisujemo pr. 192kB podataka, događa se sljedeće:

1. Prvih 64kB se zapisuje na prvi i drugi disk u blok **A1**.
2. U drugom koraku se drugih 64kB zapisuje na prvi i drugi disk u blok **B1**.
3. U trećem koraku se trećih 64kB zapisuje na prvi i drugi disk u blok **C1**.

Ovime imamo zapisano svih 192kB na oba diska (pogledajte sliku [122] lijevo).
***RAID 1*** polje daje nam najveću sigurnost jer su svi podaci zapisani na prvi disk zapisani i na drugi disk (ili koliko ih već ima u polju). Brzina čitanja je veća jer se podaci mogu čitati u parovima (polovina podatka s prvog, a pola s drugog diska). Na brzinu čitanja najviše utječe implementirana funkcionalnost čitanja s više diskova paralelno, što ovisi o RAID kontroleru tj. softveru koji to odrađuje. Pri tome lošije implementacije procesa čitanja daju i lošije rezultate.

Brzine tijekom zapisivanje su obično jednake brzini jednog diska jer se podaci moraju istovremeno zapisivati na sve diskove u polju. RAM međumemorija može ubrzati zapisivanje, ovisno koliko memorije je dostupno i u odnosu na veličinu podataka koji se zapisuju.

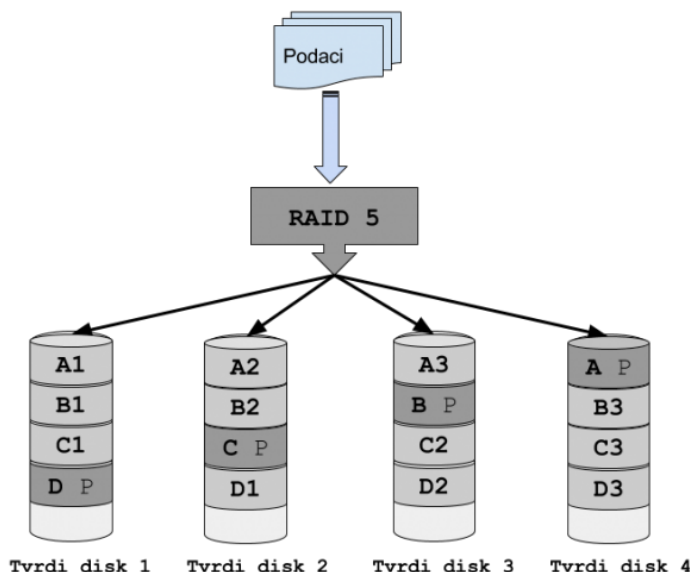
**Izvor informacija: (678)**

---

### 13.11.1.4.3. RAID 5

Za ovo RAID polje potrebna su nam minimalno tri tvrda diska. Ovo je prvo specifično polje koje koristi računanje *pariteta* nîza podataka, koje je procesorski zahtjevnije od ***RAID 0*** i ***RAID 1*** polja. Ovdje do izražaja dolaze hardverski RAID kontroleri koji imaju poseban procesor (CPU) koji izračunava paritet za svaki nîz podataka. Zamislimo ***RAID 5*** polje s četiri tvrda diska kao na slici 123.

**Slika 123. Pogled na RAID 5 polje diskova.**




Uzmimo standardni *Stripe size* od 64kB. U ovom polju, s četiri tvrda diska svaki nîz podataka se lomi na tri dijela odnosno ukupan broj diskova umanjen za jedan. Zamislimo da moramo zapisati 768kB podataka. U prvom nîzu se zapisuje sljedeće:

- Prvih 64kB na prvi disk (u blok **A1**).
- Drugih 64kB na drugi disk (u blok **A2**).
- Trećih 64kB na treći disk (u blok **A3**).
- Potom se radi logička ***XOR*** operacija prvih, drugih i trećih 64kB podataka i dobije se rezultat koji se naziva paritet (engl. *Parity*), a koji je isto veličine 64 kB i koji se zapisuje na četvrti disk (u blok **A P**). U sljedećem nîzu podataka razlomljeni podaci se zapisuju na prvi (blok **B1**), drugi (blok **B2**) i četvrti disk (blok **B3**), a paritet se izračunava i zapisuje na treći disk (blok **B P**). Kod svakog sljedećeg nîza podataka paritet se izračunava i zapisuje na neki drugi disk, tako da, ako nam se pokvari bilo koji disk u polju, od preostalih diskova se pomoću paritetnih podataka mogu izračunati (restaurirati) podaci koji nedostaju.

**Kako se radi računanje pariteta odnosno XOR funkcija?**
Prvo se podsjetimo *XOR* logičke operacije, pa pogledajte tablicu s ulaznim podacima i *XOR* rezultatom.

| Ulaz A | Ulaz B | *XOR* rezultat |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Dakle ako su oba ulaza jednaka (**A** i **B**), rezultat je nula (`0`), a ako su ulazi različiti, rezultat je jedan (`1`). Vratimo se na naša četiri diska i "*Stripe*"-ove od 64kB. Pošto nam je 64kB preveliki broj, za razumijevanje logike rada, zamislit ćemo da je "*Stripe size*" sâmo četiri bîta. Zamislimo da moramo zapisati sljedeći podatak: `101111100110`.
Pošto se on mora razlomiti na tri dijela po četiri bîta jer nam je četiri bîta *Stripe size*, to će izgledati ovako: prvi dio (`1011`), drugi dio (`1110`) i treći dio (`0110`).

Sada svaki od ovih razlomljenih dijelova ide na zapisivanje prema diskovima:
Podaci za disk **1**: `1011`
Podaci za disk **2**: `1110`
Podaci za disk **3**: `0110`
Radi se *XOR operacija*: prvi disk: `1011` **XOR** drugi disk: `1110` i kao rezultat dobivamo: **`0101`**

Sada se radi *XOR* prvog rezultata: (**`0101`**) s trećim diskom i dobivamo paritetne podatke: **`0011`**.
Dakle na četvrti disk se zapisuju paritetni podaci: **`0011`**

Sada na diskovima imamo sljedeće zapise u prvom nîzu zapisivanja:

| Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|---|---|---|---|
| 1011 | 1110 | 0110 | 0011 |

I tako redom, drugi nîz, pa paritet na disku tri, te sljedeći nîz s paritetom na disku dva, pa sljedeći nîz s paritetom na disku jedan i tako u krug.

Zamislimo da nam se pokvario "**Disk 2**" pa imamo sljedeće stanje:

| Disk 1 | **Disk 2** | Disk 3 | Disk 4 |
|---|---|---|---|
| 1011 | ~~XXXX~~ | 0110 | 0011 |

Kada je kvar otkriven, ponovno se radi *XOR* operacija, redom (Disk1, Disk 3, Disk 4):
Disk 1: `1011` *XOR* Disk 3: `0110` rezultat je: **`1101`**
Rezultat prvog *XOR*: `1101` *XOR* Disk 4: `0011` te je rezultat: **`1110`**
Dobili smo rezultat odnosno sadržaj bloka koji je bio neispravan: **`1110`**, što je točno.

**Što se događa kada imamo neispravan disk?**
Ako nemamo *Hot Spare* disk ili nismo zamijenili neispravan disk, obično će RAID kontroler raditi *XOR* u radu (u letu) za svaki nîz podataka, tako da će sve raditi i dalje (uz djelomično usporenije), ali ako nam se istovremeno pokvari još jedan disk, izgubit ćemo **SVE** podatke. Ako smo ipak imali *Hot Spare* disk, RAID kontroler će početi s procesom restauriranja podataka (engl. *Rebuild*) na *Hot Spare* disk. Ako nismo imali *Hot Spare* disk, kada izvadimo neispravni disk i ubacimo ispravni disk, proces restauriranja će se isto pokrenuti. Ovaj proces kreće uglavnom automatski, u radu. Pošto sada RAID kontroler mora preračunati podatke za svaki nîz podataka na cijeloj površini svih tvrdih diskova, to može potrajati satima ili čak danima, u slučaju kada imamo velike količine podataka i naravno velike diskove.

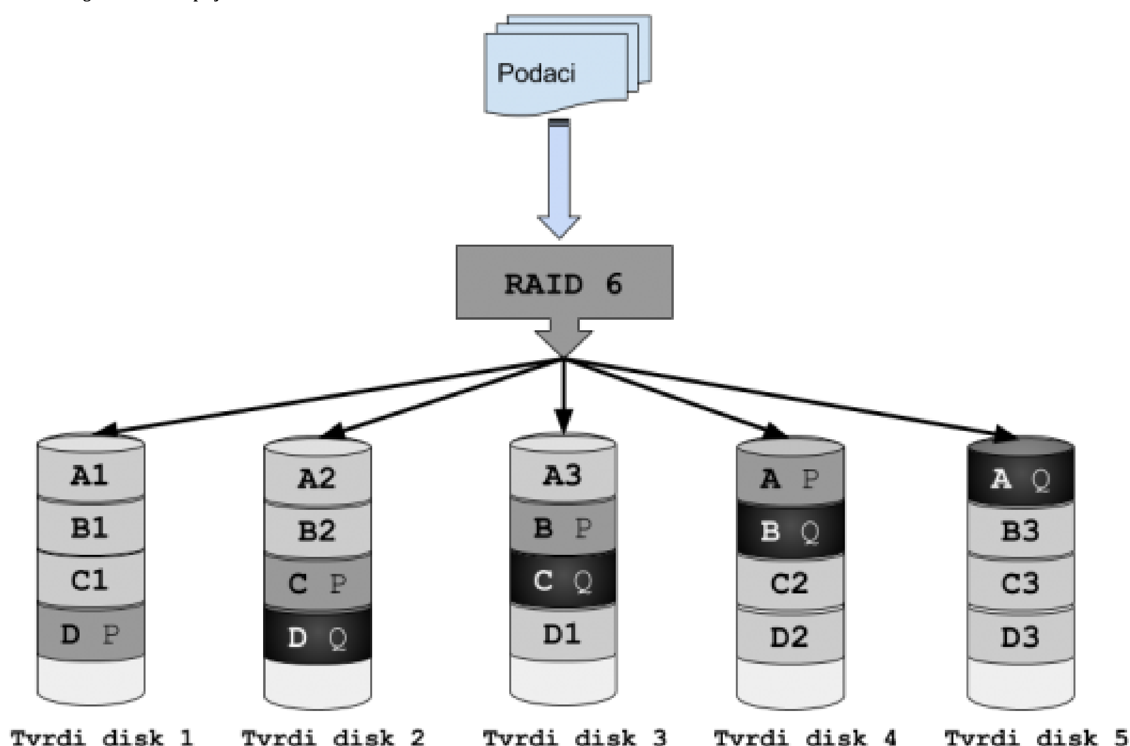
**Karakteristike RAID 5 polja**
Brzina čitanja je prilično velika jer se podaci mogu čitati s više diskova istovremeno. Brzina pisanja je manja od ***RAID 0*** i ***RAID 1*** polja jer se podaci zapisuju na više diskova paralelno, ali se za svaki nîz podataka mora izračunavati i paritet.
Ukupni kapacitet cijelog polja je jednak zbroju svih diskova umanjen za jedan (-1) disk.
***RAID 5*** polje dozvoljava ispad sâmo jednog tvrdog diska iz polja i to bilo kojeg diska u polju.

**Izvor informacija: (678).**

### 13.11.1.4.4. RAID 6

Za ovo RAID polje potrebna su nam minimalno četiri tvrda diska.
Ovo polje je slično ***RAID 5*** polju s time kako se ovdje računaju dva (2) pariteta koja se razmještaju na različite tvrde diskove u polju. Zamislimo osnovno ***RAID 6*** polje poput ovoga na slici **124**.

**Slika 124. Pogled na RAID 6 polje diskova.**




Paritet se razmješta prema istom principu kao kod ***RAID 5*** polja, svaki puta se za svaki novi nîz podataka pomiče na druge diskove. S razlikom što ovdje imamo dva pariteta od kojih se svaki zapisuje na svoj disk.
Za prvi nîz podataka, prema tome imamo (ako je *Stripe size* 64kB):

1. Prvih 64kB se zapisuje na prvi disk (u blok **A1**).
2. Drugih 64kB se zapisuje na drugi disk (u blok **A2**).
3. Trećih 64kB se zapisuje na treći disk (u blok **A3**).
4. Prvi paritet od 64kB se zapisuje na četvrti disk (u blok **A P**).
5. Drugi paritet od 64kB se zapisuje na peti disk (u blok **A Q**).

Dalje se sve mijenja slično kao kod ***RAID 5*** polja, s ciljem kako bi se za svaki slijedeći nîz podataka, pariteti zapisivali na druge diskove. Ovo polje diskova stoga omogućava ispad do dva, i to bilo koja tvrda diska u polju.
Ukupni kapacitet polja jednak je zbroju kapaciteta svih diskova umanjen za kapacitet od dva diska (-2).
Brzina čitanja je slična ili nešto sporija kao kod ***RAID 5*** polja, ovisno o implementaciji i brzini hardvera.
Brzina pisanja također ovisi o implementaciji: (hardver [i to koji?] ili softver), a također je slična ili nešto lošija od ***RAID 5*** polja, zbog dvostrukog izračunavanja pariteta koje je procesorski dvostruko zahtjevnije.

**Izvor informacija: (678)**

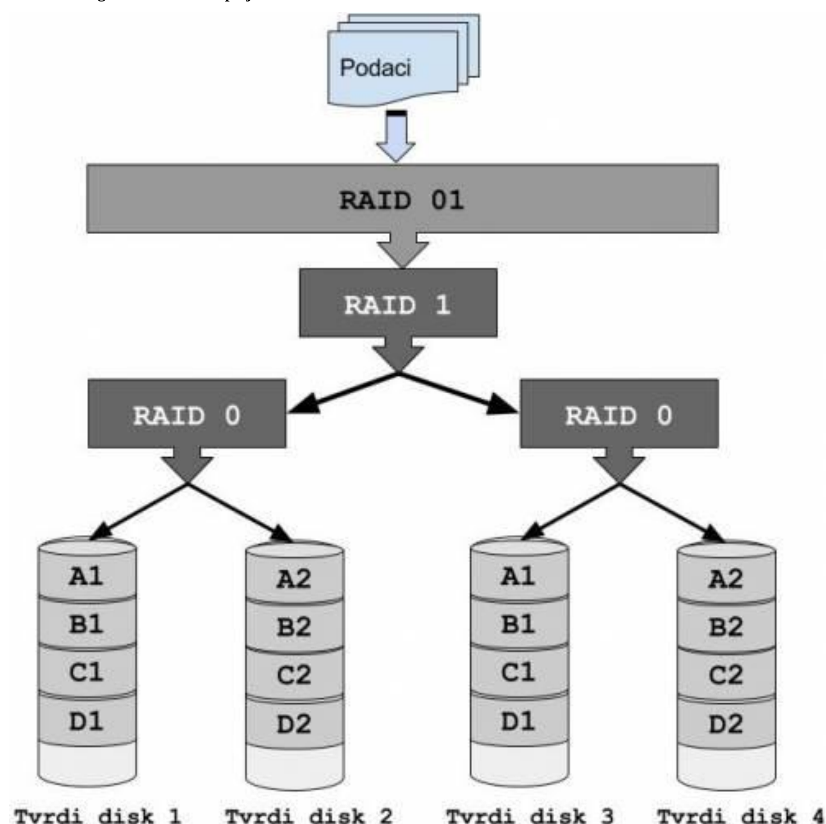
### 13.11.1.5. Ugniježđena RAID polja

U praksi se često koriste i kombinacije ***RAID 1,5*** i ***6*** s ***RAID 0*** poljem.
Ovakva polja nazivamo ugniježđenim RAID poljima (Engl. *Nested*) odnosno hibridnim RAID poljima.

#### 13.11.1.5.1. RAID 0+1

Ovo RAID polje često se naziva i **RAID 01**. Ovo polje (Engl. *Mirror of Stripes*) nastaje kreiranjem dva ili više **RAID 0** polja na dva ili više diskova koja se stavljaju unutar većeg ***RAID 1*** polja. Konačno **RAID 1** polje se prijavljuje kao logički disk *BIOS*u matične ploče računala, kako ga vidi i operativni sustav.

**Slika 125. Pogled na RAID 0+1 polje diskova.**




Ukupni kapacitet je jednak polovici kapaciteta svih diskova. Ovo polje otporno je na ispad cijelog jednog **RAID 0** polja (pošto **RAID 0** ispada uslijed kvara sâmo jednog diska). Međutim **RAID1** (engl. *Mirror*) zrcaljenje štiti s identičnom kopijom, koja se nalazi na drugom **RAID 0** polju.

U slučaju ispada dva diska iz različitih **RAID 0** polja, cijelo **RAID 01** polje se ruši i gube se **SVI** podaci. Proširivanje ovog polja (kapacitetom) je nepraktično (i skoro neizvedivo u praksi) jer zahtjeva manipulacije na svim **RAID 0** poljima.

Brzina čitanja je prilično velika pošto se podaci mogu čitati sa svih diskova istovremeno odnosno dio podataka sa svakog diska.

Brzina pisanja je također prilično velika pošto se podaci mogu čitati sa svih diskova istovremeno, isto dio podataka sa svakog diska.

Za rad ovog polja potrebno je minimalno četiri diska, kao što je vidljivo na slici 125.

**Izvor informacija: (679).**

---

#### 13.11.1.5.2. RAID 10

Ovo RAID polje često se naziva i **RAID 10**. Za rad ovog polja potrebna su minimalno četiri diska Ovo polje je slično **RAID 01**, ali prema logici rada potpuno obrnuto. Po dva diska se dodaju u **RAID 1** polje (engl. *Mirror*) odnosno zrcalno se kopiraju jedan na drugi. Druga dva diska također se dodaju u svoje **RAID 1** polje, a isto tako i treća, četvrta, peta ili n-ta dva diska.
Potom se kreira **RAID 0** (engl. *Stripe*) koji povezuje sve zrcaljene parove diskova u jedan veliki logički disk.

Ispadom (kvarom) od po jednog diska iz svakog ***RAID 1*** polja, nemamo problem. Jedino ispadom oba diska unutar pojedinog ***RAID1*** polja gubimo **sve** podatke. Dakle ovo polje, otporno je na ispad bilo kojeg diska unutar bilo kojeg ***RAID 1*** (engl. *Mirror*) zrcalnog para diskova, na bilo kojem ***RAID 1*** paru diskova.

Ukupni kapacitet ovog polja je jednak polovici kapaciteta svih diskova. Brzina čitanja je prilično velika pošto se podaci mogu čitati sa svih diskova istovremeno (dio podataka sa svakog diska). Brzina pisanja je također prilično velika pošto se podaci isto mogu čitati sa svih diskova istovremeno (isto dio podataka sa svakog diska). Dodatna prednost je i u tome što neki RAID kontroleri omogućavaju proširenje kapaciteta vršnog ***RAID 0*** polja (koje je vidljivo kao logički disk) dodavanjem novog para ***RAID 1*** (engl. *Mirror*) diskova. Proširenje polja je obično vrlo jednostavno i unutar ***LVM2*** i drugih menadžera (pr. ***ZFS***) ako koristimo softversko polje diskova.

Primjer rada ***RAID 10*** polja je vidljiv na slici 126.

**Slika 126. Pogled na RAID 10 polje diskova.**

Podaci

RAID 10

RAID 0

RAID 1 RAID 1

A1 A1 B1 B1
C1 C1 D1 D1
E1 E1 F1 F1
G1 G1 H1 H1

Tvrdi disk 1 Tvrdi disk 2 Tvrdi disk 3 Tvrdi disk 4

**Kako se zapisuju podaci u ovo polje?**

Ako imamo 512kB podataka koji se trebaju zapisati (uz standardni *stripe size* od 64 kB), događa se sljedeće:

- U prvom koraku se prva 64kB kopiraju i paralelno zapisuju na oba diska (diskovi **1** i **2**) u prvom ***RAID 1*** polju (blokovi **A1** i **A1**).
- U drugom koraku se druga 64kB kopiraju i paralelno zapisuju na oba diska (diskovi **3** i **4**) u drugom ***RAID 1*** polju (blokovi **B1** i **B1**) jer je ovo drugo ***RAID 1*** polje zapravo *Stripe* odnosno ***RAID 0*** polje, prvom ***RAID 1*** polju.
- U trećem koraku se treća 64kB kopiraju i paralelno zapisuju na oba diska (diskovi **1** i **2**) u prvom ***RAID 1*** polju (blokovi **C1** i **C1**).
- U četvrtom koraku se četvrta 64kB kopiraju i paralelno zapisuju na oba diska (diskovi **3** i **4**) u drugom ***RAID 1*** polju (blokovi **D1** i **D1**) jer je ovo drugo ***RAID 1*** polje zapravo *Stripe* odnosno ***RAID 0*** polje, prvom ***RAID 1*** polju.
- … i tako redom dok se ne zapišu svi podaci.



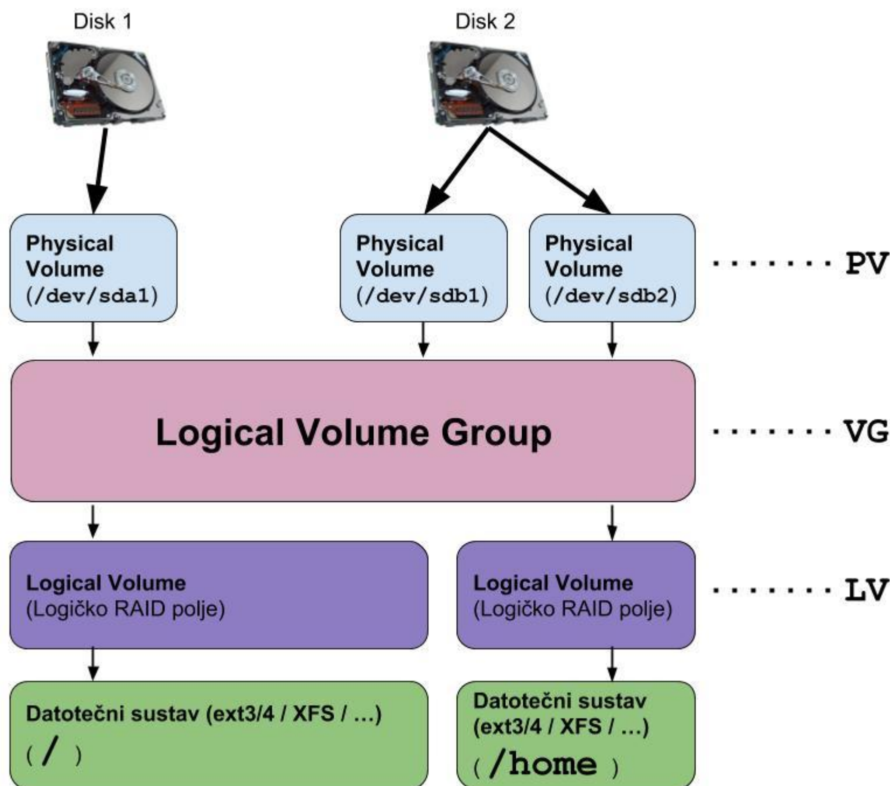
Osim navedenih ugniježđenih RAID polja, moguća su i polja **RAID 100**, **RAID 50** i **RAID 60.**

**Izvori informacija: (677),(678),(679),(680),(681),(K-12).**

## 13.11.2. Softverski RAID (LVM2)

**Slijedi napredno poglavlje (13.11.2.x)!**
Softversko RAID polje u Linuxu odrađuje takozvani **LVM2** odnosno *Logical Volume Manager* inačice dva *(2)*.
Nakon pokretanja operativnog sustava pomoću alata koji dolaze u *LVM2* paketu, kreiramo željeno RAID polje na particijama diskova. Nakon što se to polje inicijalizira možemo ga normalno formatirati i dalje koristiti kao da je u pitanju normalna particija na običnom tvrdom disku (ili neko RAID polje na hardverskom RAID kontroleru). **LVM2** je kao što mu i sâmo ime kaže: logički menadžment za pohranu podataka. Odnosno on predstavlja dinamičko, softversko RAID polje unutar operativnog sustava Linux.

**Slika 127. Pogled na LVM2 softversko RAID polje diskova.**




**Važno je znati sljedeće:**

- Pomoću *LVMa* logičke *LVM* particije se mogu rasprostirati preko više fizičkih tvrdih diskova te se mogu vrlo lako proširivati prema potrebi.
- Prema *LVMu*, takozvani *Physical Volume* (**PV**) se preslikava na particiju fizičkog diska. Prema tome moguće je na jednom tvrdom disku imati jedan ili više ***PV**-ova*.
- Logička *Volume* grupa (**VG**) odnosno *Volume Group*, se nalazi ispod ***PV**a*, kao logički disk (logički *Volume)* koji predstavlja logičko polje dostupnih diskova (tzv. ***pool***). Ovo je ekvivalent fizičkim tvrdim diskovima dostupnim hardverskom RAID kontroleru i budućem RAID polju.
- Ispod *Volume Grupe* (**VG**) se nalazi *Logical Volume* (***LV***) na koji se primjenjuje kreiranje nekog od ekvivalenata RAID polja i on prema tome predstavlja RAID polje, kao logički disk.
- Nadalje svaki *Logical Volume* (**LV**), a može ih biti jedan ili više, potrebno je formatirati s nekim datotečnim sustavom poput: ***ext3, ext4***, ***xfs*** ili nekim drugim. Na kraju je standardno potrebno montirati upravo kreirani ***LV***.

***LVM*** sustav tijekom inicijalizacije učitava postavke i opcije iz datoteke: `/etc/lvm/lvm.conf`.

**Izvori informacija: (1009),(K-12)**,**`man 7 lvmraid`**,**`man 5 lvm.conf`**.

### 13.11.2.1. Kreiranje LVM polja

Krenimo s kreiranjem *LVM2* polja. Sada ćemo dodati dva tvrda diska (ili dva virtualna diska unutar naše *virtualke*).
Pri tome će diskovi (u našem slučaju dodavanja stvarnih tvrdih diskova) biti vidljivi kao:

- `/dev/sdb` ← ovo je drugi SATA (SAS ili SCSI) disk.
- `/dev/sdc` ← ovo je treći SATA (SAS ili SCSI) disk.

Potrebno je kreirati po jednu particiju na svakom disku.



Za kreiranje particija i particije pogledajte poglavlje:
**13.4 Logička shema diska**.

Razlika kod kreiranja particije, u odnosu na normalnu particiju s kojom smo radili, je u tome što ćemo za vrstu (engl. *Type*) particije odabrati `LVM` odnosno kôd: `8E`. U primjeru za particioniranje diska ćemo koristiti program: `cfdisk` koji koristi *MBR* shemu diska, mada smo mogli koristiti i njegovu inačicu za *GPT* shemu, program imena `cgdisk` koji dolazi u softverskom paketu imena: `gdisk`. Po potrebi (ako već nije instaliran softverski paket *LVM2*), možete ga instalirati ovako:

```
yum -y install lvm2
```

Dakle kreirat ćemo jednu primarnu particiju na disku `/dev/sdb` i jednu primarnu particiju na disku `/dev/sdc`.
**1.** Odaberimo disk `/dev/sdb` pomoću programa `cfdisk` (*možemo koristiti i* `fdisk` *ili* `gdisk`) na sljedeći način:

```
cfdisk /dev/sdb
```

**2.** Označite `NEW` s kursorskim tipkama, te potvrdite s tipkom *ENTER*.
**3.** Potom odaberite `PRIMARY` jer želimo da to bude primarna particija. Pojaviti će se pitanje do koje veličine želimo da se particija proteže. Pošto želimo particiju do kraja diska, samo stisnite tipku *ENTER*.
**4.** Sada kada je particija kreirana, moramo promijeniti njenu vrstu u `LVM`.
Dakle odabrat ćemo `TYPE` te upisati `8E` jer to označava *LVM* kao vrstu particije, te potvrditi sa tipkom *ENTER*.
**5.** Konačno sve možemo snimiti, odabirom iz menija: `WRITE` pa potvrdom s tipkom *ENTER* i tek sada sve moramo potvrditi, tako što ćemo upisati: `yes` pa ponovno potvrditi tipkom *ENTER*.

**Ponovimo sve korake 1-4 i za drugi disk, s time da je u točki 1. potrebno koristiti disk:** `/dev/sdc`

Sada pogledajmo oba diska i nove *LVM* particije pomoću naredbe `lsblk`:

```
lsblk /dev/sdb /dev/sdc

NAME    MAJ:MIN RM  SIZE RO  TYPE  MOUNTPOINT
sdb       8:16   0  500M  0  disk
└─sdb1    8:17   0  500M  0  part
sdc       8:32   0  500M  0  disk
└─sdc1    8:33   0  500M  0  part
```

Vidimo uredno kreirane particije (`/dev/sdb1` i `/dev/sdc1`) na oba diska.

Za kreiranje particija, mogli smo koristiti i programe: `parted`, `fdisk`, `gdisk` ili `cgdsk`:

### 13.11.2.2. Kreiranje *LVM Physical Storage Device*a

U ovom koraku moramo kreirati takozvani *LVM* fizički uređaj. Dakle sada moramo kreirati *LVM* fizički uređaj (Engl. *Physical Storage Device*) koji će *LVM2* moći koristiti i to za svaku particiju od koje ćemo kreirati *LVM* polje. U našem primjeru dodajemo prvu particiju s prvog diska: `/dev/sdb1` i prvu particiju s drugog diska: `/dev/sdc1`, pomoću naredbe `pvcreate`:

```
pvcreate /dev/sdb1 /dev/sdc1

Physical volume "/dev/sdb1" successfully created
Physical volume "/dev/sdc1" successfully created
```

Sada možemo provjeriti jesu li fizički LVM uređaji (*Physical Volume*) kreirani pomoću naredbe `pvscan` na sljedeći način:

```
pvscan

  PV /dev/sdb1                       lvm2 [499.97 MiB]
  PV /dev/sdc1                       lvm2 [499.97 MiB]
Total: 2 [1 GB] / in use: 0 [0   ] / in no VG: 2 [1 GB]
```

Vidimo da je sve u redu (*imamo dva* `PV`*-a*). Više informacija o *PV-ovima LVM* sustava, možemo vidjeti s naredbom: `pvdisplay`. Slika prikazuje ispis naredbe `pvdisplay` koja će nam dati ove željene informacije:

```
pvdisplay

"/dev/sdb1" is a new physical volume of "499.97 MiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdb1
  VG Name
  PV Size               499.97 MiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               hH87ee-9so2-X5QW-WPKg-q6Bk-SDkm-hKlb2L

"/dev/sdc1" is a new physical volume of "499.97 MiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdc1
  VG Name
  PV Size               499.97 MiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               kP1uw8-jYyR-tcBq-o2AE-T9Z6-PnHr-T00Sxi
```

Dakle vidimo obje particije dostupne kao ***PV***-ove (Engl. *Physical volume*). Više detalja možete vidjeti s prekidačem `-m`, dakle `pvdisplay -m`, pri čemu se vide i veze između *Logical volume-a* i *device* uređaja (u `/dev/` direktoriju).

**Izvori informacija: (K-12),** **`man pvcreate`**, **`man pvscan`**, **`man pvdisplay`**, **`man cfdisk`**, **`man fdisk`**.

### 13.11.2.3. Kreiranje *LVM Volume grupe*

Sada slijedi kreiranje *LVM* grupe (*Volume Group*). Ova *LVM Volume Group-a* je polje diskova unutar kojega se nalazi minimalno jedan ili više fizičkih *LVM* uređaja odnosno *Physical Volume-a*. Ovo još uvijek nije ekvivalent RAID polju već više kao spremište diskova dostupnih kasnije za kreiranje ekvivalenta RAID polju (što će biti sljedeća točka). Kreirat ćemo *Volume Group-u* u koju ćemo dodati obje particije (`/dev/sdb1` i `/dev/sdc1`) koje smo prethodno pripremili.

Kreirajmo *Volume grupu* imena: `VolGroup01` u koju ćemo dodati navedene prethodno pripremljene particije: `/dev/sdb1` i `/dev/sdc1` pomoću naredbe `vgcreate`, na sljedeći način:

```
vgcreate VolGroup01 /dev/sdb1 /dev/sdc1

Volume  group "VolGroup01" successfully created
```

*Volume Grupu* (***VG***) možemo obrisati s naredbom: `vgremove` što bi u konkretnom slučaju, ako želimo obrisati ovu (`VolGroup01`) ***VG*** bilo sljedeće:

```
vgremove VolGroup01
```

← Ovo (brisanje) nemojte raditi, ako želite nastaviti dalje.

Krenimo dalje. Postojeće odnosno sve kreirane *Volume grupe*, možete vidjeti s naredbom `vgdisplay`:

```
vgdisplay
```

I dobit ćemo nešto poput:

```
--- Volume group ---
  VG Name               VolGroup01
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               992.00 MiB
  PE Size               4.00 MiB
  Total PE              248
  Alloc PE / Size       0 / 0
  Free  PE / Size       248 / 992.00 MiB
  VG UUID               uTSjOm-qULQ-OhT8-xLBH-FW7Y-Mr9z-9zBkYi
```

Sada kada imamo *Volume grupu* (**VG**): `VolGroup01` od nje možemo kreirati neko od ekvivalenata *RAID* poljima, a koje se prema **LVM** terminologiji zove *Logical Volume*.
Mogući izbor RAID polja je poprilično velik, mi ćemo spomenuti sâmo neke:

- `striped` – naziva se i *Stripe*. Ovo je praktično klasično ***RAID 0*** polje.
- `linear` – podaci se popunjavaju linearno u nîzu odnosno kada se popuni prvi disk popunjava se drugi. Ovo je standardni odabir ako nije ništa navedeno i ovo je linearno ***RAID 0*** polje, za razliku od klasičnog *striped* polja.
- `raid1` - ovo je standardno ***RAID 1*** polje znano kao *Mirror* odnosno zrcalno polje.

Moguće je i kreirati ekvivalente klasičnih RAID polja, o kojima smo pričali u poglavljima prije. Stoga imamo:

- `raid5_ls` je klasično ***RAID 5*** polje, a postoji još nekoliko njegovih podvrsta: `raid5_la`, `raid5_ra`, kao i polje naziva: `raid5_rs`.
- `raid6_zr` je klasično ***RAID 6*** polje, koje također ima još nekoliko podvrsta.
- `raid10` je klasično ***RAID 10*** polje.

Možemo se odlučiti za bilo koji ekvivalent RAID polju, a mi ćemo opisati samo dva (**1.** i **2.**) a odlučiti se za prvo (**1.**).
**1.** Mi ćemo se odlučiti za ***RAID 1*** pošto imamo dvije particije od 500MB. Inicijalno ćemo iskoristiti samo 400MB.
*Logical Volume grupu* ćemo nazvati: `RAID`. Kreirajmo ovo polje na sljedeći način, upotrebom naredbe `lvcreate`:

```
lvcreate --type raid1 -L400M -n RAID VolGroup01

Logical volume "RAID" created
```

**2.** Eventualno smo umjesto ***RAID 1*** polja mogli kreirati i *Stripe* polje (ekvivalent *RAID0* polju), a sve ovisi prvenstveno o broju diskova i potrebama/željama. Definirati ćemo da će ukupni iskoristivi kapacitet stoga biti 800MB jer ćemo od svakog od dodanih diskova uzeti samo 400MB. U ovom slučaju bi pokrenuli sljedeću naredbu:

```
lvcreate --type striped -L800M -n RAID0 VolGroup01

Logical volume "RAID" created
```

Na kraju pogledajmo što, odnosno koje RAID polje smo kreirali (**1.**), ali pomoću naredbe `lvscan` na sljedeći način:

```
lvscan

ACTIVE            '/dev/VolGroup01/RAID'       [400.00 MiB]  inherit
```

Vidimo RAID polje imena `RAID` i ukupnog kapaciteta `400`MB, ali ne znamo detalje o tome koja vrsta RAID polja je u pitanju.

Sada pogledajmo i što smo dobili na razini logičkog volumena, uz više detalja s naredbom: `lvdisplay`

```
lvdisplay

--- Logical volume ---
  LV Path                /dev/VolGroup01/RAID
  LV Name                RAID
  VG Name                VolGroup01
  LV UUID                OuzcXc-hydx-x627-Pmjp-K3j1-uAFV-G02LB6
  LV Write Access        read/write
  LV Creation host, time localhost.localdomain, 2018-09-06 12:30:30 +0200
  LV Status              available
  # open                 0
  LV Size                400.00 MiB
  Current LE             100
  Mirrored volumes       2
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:6
```

Vidimo da se radi o **RAID 1** polju jer se polje sastoji od dva zrcalna logička „*volumena*“: (`Mirrored volumes 2`).

**Izvori informacija: (K-12), `man vgcreate`, `man vgremove`, `man vgdisplay`, `man lvcreate`, `man lvscan`.**

#### 13.11.2.4. Formatiranje LVM polja

Ovo je zadnji korak kod kreiranja *LVM* polja diskova, koji su sada vidljivi kao jedan logički disk. Za formatiranje *LVM RAID* polja diskova je potreban isti postupak kao da formatiramo bilo koju particiju na bilo kojem "normalnom" disku. Mi ćemo formatiranje odraditi s ***ext4*** Linux datotečnim sustavom. Stoga ćemo koristiti naredbu `mkfs.ext4` na sljedeći način:

```
mkfs.ext4 /dev/VolGroup01/RAID
```

Nakon toga na razini oba blok uređaja (diskova `/dev/sdb` i `/dev/sdc`) je vidljivo sljedeće:

```
lsblk /dev/sdb /dev/sdc

NAME                                 MAJ:MIN RM  SIZE RO  TYPE   MOUNTPOINT
sdb                                    8:16   0  500M  0  disk
└─sdb1                                 8:17   0  500M  0  part
  ├─VolGroup01-RAID_rmeta_0  (dm-2)  253:2    0    4M  0  lvm
  │ └─VolGroup01-RAID  (dm-6)        253:6    0  400M  0  lvm
  └─VolGroup01-RAID_rimage_0  (dm-3) 253:3    0  400M  0  lvm
    └─VolGroup01-RAID  (dm-6)        253:6    0  400M  0  lvm
sdc                                    8:32   0  500M  0  disk
└─sdc1                                 8:33   0  500M  0  part
  ├─VolGroup01-RAID_rmeta_1  (dm-4)  253:4    0    4M  0  lvm
  │ └─VolGroup01-RAID  (dm-6)        253:6    0  400M  0  lvm
  └─VolGroup01-RAID_rimage_1  (dm-5) 253:5    0  400M  0  lvm
    └─VolGroup01-RAID  (dm-6)        253:6    0  400M  0  lvm
```

Nakon formatiranja možemo i ručno ili automatski *montirati* ovu particiju, primjerice u `/mnt/LVM` direktorij:

```
mkdir /mnt/LVM
mount /dev/VolGroup01/RAID /mnt/LVM
```

Od tog trenutka povezuje se posebni logički uređaj imena: `/dev/mapper/VolGroup01-RAID` s našim poljem.

I sada pogledajmo novo *montirani* disk koji koristi *LVM2* polje diskova:

```
df -h /dev/mapper/VolGroup01-RAID

Filesystem                       Size  Used Avail Use%  Mounted on
/dev/mapper/VolGroup01-RAID      380M  2.3M  358M   1%  /mnt/LVM
```

Dakle vidimo naše *LVM RAID* polje formatirano s ***ext4*** datotečnim sustavom, *montirano* kao da se radi o bilo kojoj drugoj formatiranoj i *montiranoj* particiji. Posebni logički uređaj imena: `/dev/mapper/VolGroup01-RAID` pokazuje na takozvani *Device Mapper* uređaj: `/dev/dm-6`.

Za više informacije o ***DM*** uređajima možemo koristiti naredbu `dmsetup`:

```
dmsetup status
```

Možemo koristiti i sljedeću naredbu:

```
dmsetup ls
```

Odnosno ljepši prikaz logičkog stabla dobivamo sa:

```
dmsetup ls --tree

VolGroup01-RAID (253:6)
 ├─VolGroup01-RAID_rimage_1 (253:5)
 │  └─ (8:33)
 ├─VolGroup01-RAID_rmeta_1 (253:4)
 │  └─ (8:33)
 ├─VolGroup01-RAID_rimage_0 (253:3)
 │  └─ (8:17)
 └─VolGroup01-RAID_rmeta_0 (253:2)
    └─ (8:17)
```

I ovdje se konačno vidi struktura logičkih uređaja uključenih u *LVM2 RAID* polje koje smo kreirali.

Struktura u *device* datotekama koja pokazuje na to *LVM2* RAID polje je vidljiva u direktoriju: `/dev/mapper/` sa:

```
ls -l /dev/mapper/

crw-rw---- 1 root root 10, 58 Sep  6 12:10 control
lrwxrwxrwx 1 root root       7 Sep  6 12:32 VolGroup01-RAID -> ../dm-6
lrwxrwxrwx 1 root root       7 Sep  6 12:30 VolGroup01-RAID_rimage_0 -> ../dm-3
lrwxrwxrwx 1 root root       7 Sep  6 12:30 VolGroup01-RAID_rimage_1 -> ../dm-5
lrwxrwxrwx 1 root root       7 Sep  6 12:30 VolGroup01-RAID_rmeta_0 -> ../dm-2
lrwxrwxrwx 1 root root       7 Sep  6 12:30 VolGroup01-RAID_rmeta_1 -> ../dm-4
lrwxrwxrwx 1 root root       7 Sep  6 12:10 VolGroup-lv_root -> ../dm-0
lrwxrwxrwx 1 root root       7 Sep  6 12:10 VolGroup-lv_swap -> ../dm-1
```



*Device mapper* je sučelje koji nam nudi linux kernel za preslikavanje fizičkih blok uređaja u logičke blok uređaje. Njega osim ***LVM2*** koriste i druge komponente linuxa, poput ***dm-crypt*** alata za kriptiranje/dekriptiranje, a on nam nudi i mogućnost izrade snimanja stanja diska ili polja diskova, u vremenu (Engl. *Snapshot*).

**Izvori informacija: (K-12), `man mkfs`, `man lsblk`, `man dmsetup`, `man df`, `man mount`.**

## 13.11.2.5. Proširivanje kapaciteta LVM polja

Važno je znati kako prema potrebi možemo i proširivati kapacitet *LVM* polja. Međutim prije bilo kakvog proširivanja moramo paziti koje RAID polje smo kreirali i kako se može proširivati. Tako se primjerice ***RAID 10*** polje mora isključivo proširivati s parovima diskova odnosno s po dva diska: jer su po dva para diskova u ***RAID 1*** polju (*mirror*), a oba ***RAID 1*** polja su u ***RAID0*** polju, što zajedno čini ***RAID10*** polje. Stoga treba dodavati po još jedan pâr diskova kojim će se proširivati postojeće ***RAID10*** polje. Vezano za sâmo proširenje kapaciteta polja, moguće je u bilo kojem trenutku proširiti kapacitet postojećeg *LVM* polja, koje smo nekada kreirali. I to na vrlo jednostavan način. Ako smo se primjerice odlučili proširiti kapacitet sa 400MB na recimo još 450MB ili želimo dodati još jedan disk te proširiti kapacitet na njega. Pri tome imamo dva izbora (**1.** i **2.**).



U primjerima koji slijede proširujemo RAID0 polje vrste ***striped***: `--type striped`

**1.** Proširenje kapaciteta na postojećim diskovima kada nam je ostalo mjesta koje smo ipak odlučili iskoristiti.
Konkretno želimo proširiti kapacitet sa 400MB na 450MB (+50 MB) jer nam još toliko treba, a toliko i imamo slobodno:
**a.)** Proširimo veličinu logičkog volumena pomoću naredbe: `lvextend` na sljedeći način:

```
lvextend -L+50 /dev/VolGroup01/RAID
```

**b.)** Sada moramo proširiti i datotečni sustav kako bi prepoznao novu veličinu onoga što on vidi kao particiju na fizičkom disku formatiranu sa *ext4* datotečnim sustavom. To ćemo napraviti pomoću naredbe `resize2fs` na sljedeći način:

```
resize2fs /dev/VolGroup01/RAID
```

Ovdje smo koristili *ext4* naredbu `resize2fs` za proširivanje postojećeg *ext4* datotečnog sustava na upravo proširenu *LVM* particiju: `/dev/VolGroup01/RAID`

**2.** Proširenje kapaciteta dodavanjem novog diska. Proširivanje kapaciteta dodavanjem novog diska moguće je, ali potrebno je paziti koje RAID polje smo prethodno kreirali. Primjerice, ako imamo ***RAID1*** polje s dva diska bilo bi preporučljivo dodati još dva diska u novo ***RAID1*** polje te proširiti postojeće polje. Ovdje je važno shvatiti da proširenje radi bez brisanja postojećih podataka, mada je preporuka uvijek prethodno napraviti sigurnosnu kopiju svojih podataka.

**a.)** Prvo pogledajmo sve od najniže razine ***LVM***-a. Pogledajmo koje sve fizičke volumene (*Physical Volumes/**PV***) imamo:

```
pvs /dev/sdb1 /dev/sdc1

  PV         VG         Fmt  Attr PSize   PFree
  /dev/sdb1  VolGroup01 lvm2 a--u 496.00m 92.00m
  /dev/sdc1  VolGroup01 lvm2 a--u 496.00m 92.00m
```

Vidimo kako smo dodali dva diska odnosno njihove ***lvm2*** particije: `/dev/sdb1` i `/dev/sdc1` u ***PV*** (*Physical Volume).*

**b.)** Sada pogledajmo našu *Volume Grupu (VG)* imena: `VolGroup01`

```
vgs VolGroup01

  VG          #PV  #LV #SN  Attr    VSize     VFree
  VolGroup01    2    1   0  wz--n-  992.00m   184.00m
```

Naša ***VG*** je imena: `VolGroup01` te ima dva (2) **`PV`** (diska) koja čine jedan (1) logički disk (**`LV`**).

Sada pogledajmo naš ***LV***, prema ***VG*** imenu, isto pomoću naredbe `lvs`:

```
lvs VolGroup01

LV   VG         Attr       LSize   Pool Origin Data% Meta%  Move Log Cpy%Sync Convert
RAID VolGroup01 rwi-a-r--- 400.00m                                   100.00
```

Vidimo kako imamo ***LV*** imena ***RAID***, čiji je dostupni kapacitet 400MB.

**U primjeru koji slijedi ćemo dodati jedan novi disk (`/dev/sdd`) i na njemu kreirati LVM vrstu particije koja će potom biti vidljiva kao: `/dev/sdd1`.**

**c.) Sada moramo proširiti našu *Volume Grupu* odnosno *VG*.**

Dakle u našu postojeću ***VG*** imena: `VolGroup01` dodajemo novu particiju s kojom ju proširujemo, sve sa naredbom `vgextend` na sljedeći način:

```
vgextend VolGroup01 /dev/sdd1
```

Ako je sve u redu, dobivamo poruku:

```
  Physical volume "/dev/sdd1" successfully created
  Volume group "VolGroup01" successfully extended
```

Pogledajmo sada našu ***VG*** pomoću naredbe `vgs`:

```
vgs VolGroup01

  VG          #PV #LV #SN Attr   VSize VFree
  VolGroup01    3   1   0 wz--n- 1.45g 680.00m
```

Sada imamo tri (3) **`PV`**-a, dakle tri diska/particije unutar našeg ***`LV`***-a, a vidimo i kako je narastao ukupni kapacitet na 1.45GB.


U slučaju kada ipak trebamo izbaciti jedan PV (LVM particiju/disk) iz VG-a, to možemo napraviti na sljedeći način:

```
vgreduce VolGroup01 /dev/sdd1
```

← Ovdje smo iz naše ***VG*** izbacili particiju: `/dev/sdd1`.

Ovo je samo primjer, mi ovu particiju nećemo izbaciti iz *Volume grupe* (***VG***), dakle ne pokrećite ovu naredbu.

Vratimo se na primjer u kojem smo dodali novu particiju u **VG** i pogledajmo kako naša **VG**:`VolGroup01` sada izgleda.

To ćemo napraviti s naredbom `pvscan` i to ovako:

```
pvscan

  PV /dev/sdb1   VG VolGroup01      lvm2 [496.00 MiB / 92.00 MiB free]
  PV /dev/sdc1   VG VolGroup01      lvm2 [496.00 MiB / 92.00 MiB free]
  PV /dev/sdd1   VG VolGroup01      lvm2 [496.00 MiB / 496.00 MiB free]
```

Kako bismo jednostavno vidjeli koliko ukupno možemo proširiti naše LVM RAID polje možemo pobrojati sve vrijednosti ili koristiti naredbu `vgdisplay` uz jedno filtriranje s naredbom `grep` ovako:

```
vgdisplay VolGroup01 | grep "Free  PE"

  Free  PE / Size       170 / 680.00 MiB
```

Dakle ukupno je moguće proširiti se do 680MB. Mi ćemo se ipak proširiti nešto manje od ovog limita, tj. na +500MB.

I sada možemo napraviti operaciju proširenja našeg ***LVM*** polja (kao i u slučaju **1.**):

```
lvextend -L+500 /dev/VolGroup01/RAID
```

I tek sada je moguće proširiti i datotečni sustav jer on nije „*svjestan*“ nove veličine particije na kojoj se nalazi (RAID polja).

Na našem *LVM2* RAID polju je potrebno napraviti sljedeće:

```
resize2fs /dev/VolGroup01/RAID
```

Kada se ovaj proces završi, imat ćemo datotečni sustav, proširen na cijelo *LVM2* RAID polje.

**Izvori informacija: `man lvextend`, `man resize2fs`, `man pvs`, `man vgs`, `man lvs`, `man vgextend`, `man vgreduce`, `man pvscan`, `man vgdisplay`, `man lvextend`.**

### 13.11.2.6. Brisanje LVM polja

Ponekada imamo potrebu i obrisati cijelo *LVM* polje diskova.



OPREZ: Ako brišemo *LVM* polje, svi podatci će biti izgubljeni.

Brisanje *Logical Volume* grupe je praktično brisanje cijelog *LVM2* RAID polja, koje je u našem slučaju imena **VolGroup01**, a koje pokazuje na RAID polje imena **RAID**. Obrišimo ga pomoću naredbe `lvremove` na sljedeći način:

```
lvremove VolGroup01
```

I to je to, obrisali smo cijelo *LVM2 RAID* polje i sve podatke na njemu.

**Izvori informacija: (K-12), `man lvremove`,`man 5 lvm.conf`.**

### 13.11.2.7. LVM thin

Prisjetimo se: *Logical Volume Manager* (**LVM**) softverski je sustav sličan RAID poljima odnosno načinu rada RAID kontrolera. To radi tako da se nakon dodavanja željenih particija diskova u tzv. fizičke volumene (***PV*** [*Physical Volume*]), oni mogu dodati u *grupe volumena* (**VG** [*Volume Group*]) odnosno takozvani *pool* koji zapravo čini skupinu dostupnih diskova za pohranu. Tek tada se dostupni diskovi iz ove skupine (*poola/**VG***) koriste za kreiranje ekvivalenta RAID polja, kreiranjem logičkog volumena (**LV** [*Logical Volume*]), koji predstavlja logički disk, koji se potom može formatirati i koristiti, kao da se radi o stvarnom (fizičkom) disku.



Za detalje o radu LVM(2) sustava podsjetite se poglavlja:
**13.11.2. Softverski RAID (LVM2).**

Blokovi (podataka) na **LVM** sustavu na standardnom logičkom volumenu (***LV***) dodjeljuju (alociraju) se kada je **LV** stvoren, ali blokovi u takozvanom *thin provision LVM-u* dodjeljuju se tek kada se na njih nešto stvarno i zapisuje.
Zbog toga što *thin provision logički volumen* (***LV***) dobiva virtualnu veličinu, bez zauzimanja stvarnog prostora na disku, on može biti mnogo veći od fizički dostupnog prostora za pohranu. Nadalje, veličina fizičkog prostora za pohranu predviđena za *thin provision LV-ove* može se kasnije i povećati prema potrebi.
Blokovi (podataka) u standardnom **LV**-u (**LVM** sustava) se dodjeljuju tijekom stvaranja, iz grupa volumena (***VG*** [*Volume Group*]), ali blokovi u *thin provision LV-u* se dodjeljuju tek tijekom korištenja, iz posebnog *thin provision pool LV-a*.
*Thin provision pool LV* sadrži blokove fizičkog sustava za pohranu, a blokovi u *thin provision LV-ovima* sadrže samo reference blokova u *thin provision LV-u*. Snimke stanja (*snapshot*) *thin provision LV-ova* su dodatno vrlo učinkovite jer su podaci u blokovima zajednički za *thin provision LV* i sve njegove snimke, dijeljeni među sobom.

**Za primjer ćemo koristiti dva diska od 2GB i to: `/dev/sda` i `/dev/sdc`.**
Prvo instalirajmo program `parted` iako smo sve mogli napraviti i s programima: `cfdisk`, `fdisk`, `gdisk` ili `cgdsk`:

```
yum -y install parted
```

Pošto koristimo testne diskove od 2GB: `/dev/sda` i `/dev/sdc`, na njima treba kreirati po jednu particiju:

```
parted /dev/sda mklabel gpt
parted /dev/sda mkpart prvi ext4 0% 100%
parted /dev/sdc mklabel gpt
parted /dev/sdc mkpart prvi ext4 0% 100%
```

S gornjim korakom smo kreirali particije: `/dev/sda1` i `/dev/sdc1` koje ćemo potom koristiti za LVM.
Provjerimo ih najjednostavnije moguće, s naredbom `fdisk`:

```
fdisk -l /dev/sda /dev/sdc | egrep "^/dev/sda|^/dev/sdc"
/dev/sda1   2048 4192255 4190208   2G Linux filesystem
/dev/sdc1   2048 4192255 4190208   2G Linux filesystem
```

Vidimo kreirane obije particije od 2GB, kako smo i očekivali.

Zatim dodajemo ove dvije particije kao fizičke volumene (**PV** [*physical volume*]):

```
pvcreate /dev/sda1 /dev/sdc1
  Physical volume "/dev/sda1" successfully created.
  Physical volume "/dev/sdc1" successfully created.
```

Sada ćemo kreirati *Volume Grupu* (**VG**) koja predstavlja tzv. *pool*:

```
vgcreate VolGroupa-1 /dev/sda1 /dev/sdc1
  Volume group "VolGroupa-1" successfully created
  Volume group "VolGroupa-1" successfully created
```

I tek sada smo došli do *LVM thin* dijela, pa ćemo kreirati *thin pool* (**LV**) pomoću prekidača `-T`:

```
lvcreate -L 2GB -T VolGroupa-1/THIN
```

Dakle unutar *pool-a* (to jest ***VG-a***) imena `VolGroupa-1` kreirali smo *thin pool LV-a* naziva `THIN`.
Osnovne podatke možemo vidjeti s naredbom `lvdisplay`, a više detalja s naredbom `lvs`:

```
lvs -a
LV                 VG          Attr       LSize  Pool Origin Data%  Meta%  Move Log
  THIN             VolGroupa-1 twi-a-tz--  2.00g             0.00   11.04
  [THIN_tdata]     VolGroupa-1 Twi-ao----  2.00g
  [THIN_tmeta]     VolGroupa-1 ewi-ao----  4.00m
  [lvol0_pmspare]  VolGroupa-1 ewi-------  4.00m
```

Vezano za *thin volume*, iz atributa (stupac `Attr`) vidimo da se radi o:

- (`t`)*hin pool* → *za LV – `THIN`.*
- (`T`)*hin pool data* → *za LV – `THIN_tdata` koji sadrži stvarne podatke (ne metapodatke).*

A sada kreirajmo još dva *thin volume-a* veličine 1GB iz postojećeg *thin poola* `VolGroupa-1` (koji je trenutno **2GB**):

```
lvcreate --thinpool VolGroupa-1/THIN --name Thin-vol-1 --virtualsize 1G
lvcreate --thinpool VolGroupa-1/THIN --name Thin-vol-2 --virtualsize 1G
```

Pogledajmo sada novo stanje:

```
lvs -a
 LV                    VG            Attr          LSize    Pool  Origin Data%   Meta%
Move Log
  THIN           VolGroupa-1 twi-aotz-- 2.00g              0.00   11.23
  [THIN_tdata]   VolGroupa-1 Twi-ao---- 2.00g
  [THIN_tmeta]   VolGroupa-1 ewi-ao---- 4.00m
  Thin-vol-1     VolGroupa-1 Vwi-a-tz-- 1.00g   THIN       0.00
  Thin-vol-2     VolGroupa-1 Vwi-a-tz-- 1.00g   THIN       0.00
  [lvol0_pmspare]    VolGroupa-1 ewi------- 4.00m
```

Ovdje vezano za *thin volume*, iz atributa (stupac `Attr`), vidimo sljedeće:

- (`t`)*hin pool* → *dakle za `THIN` LV vidimo da je on thin pool.*
- (`T`) *hin pool dana* → *za LV – `THIN_tdata` koji sadrži stvarne podatke (ne i metapodatke)*
- *Thin* (`V`)*olume* → *za `Thin-vol-1` i `Thin-vol-2` vidimo da su oni također thin volume-i.*

U ovom primjeru jasno je da imamo *LVM pool* naziva `THIN` koji koristimo za naša dva *LVM thin volume-a* naziva: `Thin-vol-1` i `Thin-vol-2`.

Osim atributa (stupac `Attr`) imamo i druge korisne stupce statistika (navest ćemo samo neke od njih), poput:

- `LSize` → označava zadanu veličinu *pool-a*.
- `Origin data%` → označava koliko se stvarnih podataka nalazi unutar *LVM (thin) volume-a*.



Ovakva upotreba ***LVM thin volume-a*** se često koristi kod virtualizacije i/ili Linux kontejnera.
Pri tome primjerice svaki Linux kontejner dobiva svoj *thin volume* ograničene veličine (kapaciteta).



Za detalje primjene diskovnih sustava u virtualizaciji, pogledajte poglavlje:
**27.1.2.4. Disk image formati.**

Za ***SSD*** diskove možete postaviti opciju `issue_discards=1` u datoteku `/etc/lvm/lvm.conf` za *LVM thin*.
To je za slučajeve kada želite da se aktivira *TRIM/Discard* opciju u slučaju, ako se unutar ***LV***-a (*Logical Volume*) više ne koristi određeni **PV** (*physical volume*). Dakle u slučajevima kada primjerice izbacujemo određeni disk (particiju diska) iz ***LV***-a ili kada smanjujemo kapacitet **LV**-a, to jest kada se primjerice koriste naredbe: `lvremove` i `lvreduce` na ***SSD*** diskovima.

**Izvori informacija: (1409),(1410),(1411),(1412),(1413),(K-8), `man 7 lvmthin`,`man 8 lvs`,`man 7 lvmraid`, `man parted`,`man pvcreate`,`man vgcreate`,`man lvcreate`,`man 5 lvm.conf`.**

## 13.11.3. Device Mapper (DM) i md (RAID)

***Slijedi napredna cjelina!***

### *Device mapper* (DM)

*Device mapper* (**DM**) je podkomponenta za upravljanje logičkim diskovnim uređajima u Linuxu. On omogućava stvaranje, upravljanje i manipuliranje logičkim diskovnim uređajima, kao što su logički volumeni, na način koji je transparentan za aplikacije (i korisnike). *Device Mapper* podržava različite podsustave i funkcionalnosti koje omogućavaju napredne značajke poput dinamičkog alociranja prostora, kreiranja snimaka (*snapshots*), kriptiranja sadržaja diskova (šifriranja) i još mnogo toga. Neke od funkcionalnosti *Device Mappera*, implementirane kao moduli su:

- `dm-mirror` – koristi se za zrcaljenje podataka na više fizičkih diskova.
- `dm-snapshot` – koristi se za stvaranje snimaka (engl. *snapshot*) logičkih volumena.
- `dm-thin` – koristi se za *thin provisioning*, gdje je fizička pohrana alocirana samo kada je potrebna.
- `dm-writecache` – koristi se za ubrzanje zapisa podataka na disk koristeći brzi NVMe ili SSD uređaj kao predmemoriju za operacije zapisivanja.
- `dm-cache` – koristi se kao predmemorija (za *caching*), omogućujući korištenje bržih uređaja kao cache za sporije uređaje, univerzalniji je i može se koristiti za predmemoriranje operacije čitanja i zapisivanja.
- `dm-stripe` – koristi se za *striping* podataka preko više diskova radi povećanja performansi.
- `dm-integrity` – osigurava integritet podataka korištenjem provjera sažetaka (hash) za detekciju i ispravak grešaka.
- `dm-crypt` – koristi se za implementaciju transparentnog kriptiranja (šifriranja) blok uređaja, …

Kao što je vidljivo iz liste nekih od ***Device Mapper*** modula, *Device Mapper* je vrlo fleksibilna i moćna komponenta koja je temelj mnogih naprednih funkcionalnosti upravljanja diskovima u Linuxu. Naime korištenjem različitih gore navedenih modula, omogućava stvaranje složenih diskovnih konfiguracija koje zadovoljavaju različite potrebe za pohranom podataka. Važno je razumjeti da se i neke druge tehnologije oslanjaju na *Device Mapper*, poput:

- *Logical Volume Manager* (**LVM**):
  - Koristi ***Device Mapper*** za implementaciju logičkih volumena, uključujući *mirror*, *snapshot* i *thin provisioning* funkcionalnosti, koje koristi od *Device Mappera*.
- **Dm-crypt** i *Linux Unified Key Setup* (**LUKS**):
  - ***dm-crypt*** koristi *Device Mapper* za implementaciju transparentnog kriptiranja blok uređaja.
  - ***LUKS*** također koristi dm-crypt za kriptiranje i upravljanje ključevima
- ***STRATIS* –** alat za rad s logičkim volumenima i LVM2.
- **Md** (**MDADM** [*RAID*]) – on je zadužen za RAID polja diskova, te može raditi preko *Device Mappera*, ali i bez njega odnosno direktno. Također LVM2 se može oslanjati na njega, ali i raditi RAID polja bez njega (**MD**).
- ***Multipath I/O*** (**MPIO**):
  - Koristi *Device Mapper* za implementaciju višestrukih fizičkih putanja do istog diskovnog uređaja, pružajući redundanciju i balansiranje opterećenja.

Aplikacije/podkomponente poput **LVM2** koje trebaju kreirati nove logičke uređaje komuniciraju s *Device Mapperom* koji kreira posebne *device* datoteke. Uobičajeno poput `/dev/dm-0`, `/dev/dm-1` i sličnih, s kojima se povezuje preko simboličkog linka: `/dev/mapper/`*XX (pr.* `/dev/mapper/vgdata-lvdata`*)* i primjerice za LVM2 preko simboličkog linka: `/dev/vgdata/lvdata`. U slučaju upotrebe LVM2 ove poveznice sa simboličkim linkovima se zapisuju u *metadata* dio LVM2 logičkog uređaja odnosno njihovi simbolički linkovi pokazuju na primjerice `/dev/dm-0`.

Ako koristimo *Device Mapper*, tada imamo slijedeću logičku vezu:

***Disk 🡐 Device Mapper 🡐 Logički diskovni uređaj 🡐 🡐 (LVM, LUKS, Multipath, …)***

Iako je moguć i ovakav način rada:

***Disk 🡐 MD [MDADM] 🡐 Device Mapper 🡐 Logički diskovni uređaj 🡐 🡐 (LVM, LUKS, Multipath, …)***

Trenutna konfiguracija *Device Mappera* se može ispisati ili konfigurirati i interaktivno ili iz skripti, upotrebom uslužnog programa `dmsetup`. S ovom naredbom najčešće ispisujemo status *Device Mapper* logičkih volumena, koji su primjerice kreirani s **LVM2**, stoga što se (iako je i to moguće) ručno kreiranje *Device Mapper* logičkih volumena ne preporučuje. Neke od korisnih naredbi su:

- `dmsetup ls` – koristi se za ispis *Device Mapper* logičkih volumena.
- `dmsetup status` – koristi se za ispis statusnih informacija o *Device Mapper* uređajima, primjerice:

```
dmsetup status
vg_main-lv_main: 0 20971520 linear
```

  - U ispisu nakon imena uređaja slijede: početni sektor logičkog uređaja (pr. `0`) te broj sektora koje uređaj ima.
  - Slijedi vrsta logičkog volumena koja može biti: `linear` - linearno mapiranje (standardno), `striped` – podijeljen u *stripe*, `mirror` – zrcaljen, `snapshot` – *snapshot*, `thin-pool` – *thin pool*, `thin` – *thin provisioning*, `raid` – RAID polje, `multipath` – *Multipath* (*MPIO*) ili `crypt` – *DM-Crypt ili LUKS*.
- `dmsetup info` – koristi se za ispis općih informacija o *Device Mapper* uređajima.
- `dmsetup info /dev/mapper/dm-0` – koristi se za ispis informacija o određenom uređaju (pr. `dm-0`).



Za detalje oko **LVM2** pogledajte cjelo poglavlje:
**13.11.2. Softverski RAID (LVM2).**

## Md (MDADM [*RAID*])

*Multiple Device Admin* (**MDADM**) je alat za administraciju softverskih RAID uređaja u operativnom sustavu Linux.
On se koristi za RAID (*Redundant Array of Independent Disks*) tehnologiju, koja omogućava kombiniranje više fizičkih diskova u jednu ili više logičkih jedinica za različite ciljeve, uključujući poboljšanje performansi i/ili povećanje pouzdanosti i otpornosti na kvarove.

Trenutno *mdadm* podržava sljedeće *md* uređaje/mehanizme:

- **LINEAR** – ovo polje spaja više diskova u jedan veliki virtualni disk bez ikakve RAID razine. Podaci se zapisuju na prvi disk dok se ne popuni, a zatim na drugi, i tako dalje.
- **RAID0** (*Striping*) – raspoređuje podatke preko dva ili više diskova koristeći tehniku stripinga. Podaci se dijele na blokove i zapisuju se paralelno na sve diskove.
- **RAID1** (*Mirroring*) – zrcali (mirroring) podatke na dva ili više diskova. Svi podaci su identični na svim diskovima u polju.
- **RAID4** – koristi *striping* s jednim diskom posvećenim za pohranu pariteta. Paritet omogućava rekonstrukciju podataka u slučaju kvara samo jednog diska.
- **RAID5** - koristi *striping* s distribuiranim paritetom. Paritet se raspodjeljuje preko svih diskova, čime se eliminira usko grlo jednog paritetnog diska.
- **RAID6** – sličan je RAID5, ali koristi dva skupa pariteta distribuirana po svim diskovima. Time omogućava oporavak podataka čak i ako dva diska otkažu.
- **RAID10** - kombinira *striping* (RAID0) i *mirroring* (RAID1). Polje je organizirano u skupine zrcalnih diskova koji su potom spojeni pomoću *stripinga*.
- **MULTIPATH** – omogućava korištenje više putanja (konekcija) do istog diska radi poboljšanja performansi i redundancije. To je korisno u *Storage Area Network* (**SAN**) okruženjima.



U kontekstu korištenja s **MDADM**, ***Device Mapper*** može biti korišten za dodatne slojeve apstrakcije preko RAID polja, omogućujući, na primjer, kreiranje logičkih volumena unutar RAID polja.

Međutim **MDADM** je alat specifičan za upravljanje softverskim RAID poljima u Linuxu i ne ovisi o *Device Mapperu*. *MDADM* upravlja softverskim RAID poljima direktno, bez potrebe za dodatnim slojem apstrakcije koji pruža *Device Mapper*. To znači da možete koristiti samo *Multiple Device Admin* (**MDADM**) ili *Device Mapper* ili *Device Mapper* u kombinaciji s *Multiple Device Admin* (**MDADM**).
Dakle upotrebom **md** (*MDADM*) i *Device Mapper* moguć je i ovakav načini rada:

***Disk ← MD [MDADM {RAID}] ← Device Mapper ← Logički diskovni uređaj ← ← (LVM, LUKS, Multipath, …)***

**Kreiranje RAID polja**

Naredba `mdadm` koristi niz opcija za inicijalizaciju novog RAID polja.
Primjerice, za stvaranje RAID 1 polja s dva diska (`/dev/sda` i `/dev/sdb`), koristi se naredba:

```
mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sda /dev/sdb
```

**Uklanjanje diska iz RAID polja**

Ako iz prethodno kreiranog RAID 1 polja želimo ukloniti disk `/dev/sda`, to ćemo učiniti ovako:

```
mdadm --manage /dev/md0 --fail /dev/sda
mdadm --manage /dev/md0 --remove /dev/sda
```

MDADM omogućava praćenje statusa RAID polja te automatsku obnovu podataka u slučaju kvara diska.
Praćenje prethodno kreiranog RAD polja koje vidljivo kao `/dev/md0` se može izvršiti s naredbom:

```
mdadm --detail /dev/md0
```

Ako se jedan disk pokvari, MDADM će automatski pokušati obnoviti podatke koristeći redundantne informacije sa preostalih diskova. Da bi osigurao automatsko prepoznavanje i montiranje RAID polja prilikom pokretanja, MDADM koristi konfiguracijsku datoteku `/etc/mdadm/mdadm.conf`. Ova datoteka sadrži detalje o RAID poljima koja su kreirana na sustavu.

**Izvori informacija: (1706),(1707),(1708), `man mdadm`, `man mdadm.conf`, `man dmsetup`.**

### 13.11.3.1. Device Mapper Multipath (Multipath I/O)

*Device Mapper Multipath* (**Multipath I/O** odnosno **MPIO**) je još jedan podsustav koji se oslanja na *Device Mapper* u Linuxu. *Multipath I/O* omogućava korištenje više fizičkih putanja za pristup istom diskovnom uređaju, čime se poboljšava pouzdanost, redundancija i performanse diskovnih (I/O) operacija. Ovo je posebno korisno u okruženjima gdje su diskovni uređaji povezani putem više fizičkih puteva, poput *Storage Area Network* (**SAN**) sustava.
U takvom radu, ako jedna putanja nije dostupna, **MPIO** može automatski preusmjeriti I/O operacije na drugu dostupnu putanju odnosno diskovni (*blok*) sustav. Pri tome *Device Mapper* upravlja logikom preusmjeravanja I/O operacija između dostupnih putanja. ***MPIO*** je ključan za kritične produkcijske sustave gdje je pouzdanost i dostupnost podataka od najveće važnosti.



Za više detalja o **SAN** sustavima, pogledajte poglavlje:
**13.12.1. Storage Area Network (SAN).**

**Način rada Multipath I/O (MPIO)**
Kada su diskovni uređaji dostupni preko više putanja, ***MPIO*** prvo detektira sve dostupne putanje prema svakom uređaju. *Device Mapper* koristi informacije o putanjama za stvaranje jedinstvenog logičkog uređaja koji predstavlja sve dostupne putanje do fizičkog uređaja. *MPIO* može balansirati I/O opterećenje između različitih putanja kako bi optimizirao performanse. Postoji nekoliko mehanizama balansiranja opterećenja, uključujući: *round-robin, least queue depth* i druge.
Ako jedna putanja postane nedostupna zbog kvara hardvera ili mreže, MPIO automatski preusmjerava I/O operacije na preostale dostupne putanje, osiguravajući kontinuirani pristup podacima.

Ključne logičke komponente MPIO-a su:

- *Multipath Device* (multipath uređaj) – ovo je virtualni uređaj koji predstavlja sve fizičke putanje do uređaja za pohranu. Korisnici i aplikacije komuniciraju s ovim uređajem, a ne s pojedinačnim fizičkim putanjama.
- *Path* (putanja) – fizička veza između poslužitelja i uređaja za pohranu. Svaki uređaj može imati više putanja, npr. putem različitih adaptera, switcha ili kablova.
- *Path Grouping* (grupiranje putanja) – putanje se grupiraju prema karakteristikama kao što su latencija ili propusnost, omogućujući balansiranje opterećenja i prebacivanje u slučaju kvara.
- *Load Balancing* (balansiranje opterećenja) – distribucija I/O zahtjeva preko više putanja kako bi se maksimizirale performanse i smanjilo opterećenje pojedinačne putanje.
- *Failover* - automatsko preusmjeravanje I/O zahtjeva s neispravne putanje na dostupnu putanju.

**Pogledajmo primjer konfiguracije upotrebom Fibre Channel mreže i Fibre Channel SAN sustava**
Konfiguracija MPIO na Linux sustavima obično uključuje instalaciju programskog paketa `multipath-tools`. Pri tome konfiguracijska datoteka `/etc/multipath.conf` sadrži pravila za prepoznavanje i upravljanje *multipath* uređajima.
Ova datoteka ne opisuje direktno sve ključne komponente MPIO-a, već sadrži specifične postavke i pravila za konfiguraciju tih komponenti. Zamislimo ovakvu vezu na ***Fibre Channel SAN*** sustav za pohranu:

***Poslužitelj (ovo računalo) ← Device Mapper ← Multipath IO = [Fibre Channel (HBA)] ← Fibre Channel SAN***

Struktura i sadržaj datoteke `/etc/multipath.conf` bio bi sljedeći:

1. Sekcija `Defaults`, u kojoj se definiraju postavke koje definiraju globalne zadane vrijednosti za MPIO; primjerice:

```
defaults {
    user_friendly_names yes
    find_multipaths yes
}
```

- `user_friendly_names yes` – ova postavka omogućuje korištenje prijateljskih (user-friendly) imena za multipath uređaje umjesto njihovih podrazumijevanih identifikatora (npr. **WWID** - *World Wide Identifier*). Kada je ova opcija omogućena (`yes`), multipath uređaji će biti imenovani na način koji je lakše čitljiv i prepoznatljiv za korisnike, obično u obliku imena, poput: `mpath1`, `mpatha`, itd. Prijateljska imena se pohranjuju u `/etc/multipath/bindings` datoteku, što osigurava dosljednost imena preko restarta računala ili ponovnog pokretanja `multipathd` servisa.

2. Sekcija `Blacklist`, u kojoj se definiraju pravila za isključivanje određenih uređaja iz MPIO; primjerice:

```
blacklist {
    devnode "^sda"
}
```

- `devnode "^sda"` - koristimo kada postoje uređaji koje ne želimo da se tretiraju kao multipath uređaji, bilo zato što su lokalni diskovi (što je ovdje slučaj za disk `/dev/sda`) ili iz drugih razloga.

3. Sekcija `Devices`, sa specifičnim postavkama za uređaje, uključujući proizvođača, proizvod (uređaj), politiku grupiranja putanja, prioritete, itd.; primjerice:

```
devices {
    device {
                vendor "3PARdata"
                product "VV"
                path_grouping_policy "group_by_prio"
                hardware_handler "1 alua"
                prio "alua"
                failback "immediate"
                no_path_retry 18
                fast_io_fail_tmo 10
                dev_loss_tmo "infinity"
                vpd_vendor "hp3par"
           }
}
```

Opis:

- `vendor "3PARdata"` i `product "VV"` – označavaju proizvođača i model ***SAN*** uređaja (radi se o ***HP Enterprise 3PAR*** **SAN** sustavu)
- `path_grouping_policy "group_by_prio"` – definira politiku grupiranja putanja. Vrijednost `group_by_prio` grupira putanje prema prioritetu.
- `hardware_handler "1 alua"` – ova postavka definira upravljač za hardver koji se koristi za prebacivanje putanja. Vrijednost `1 alua` specificira korištenje **ALUA** (*Asymmetric Logical Unit Access*).
- `prio "alua"` – ova postavka definira metodu za određivanje prioriteta putanja. U ovom primjeru, koristi se **ALUA** metoda.
- `failback "immediate"` – označava da se putanje vraćaju u upotrebu odmah nakon što postanu dostupne.
- `no_path_retry 18` – definira broj pokušaja prije nego što se odustane od neuspjele putanje.
- `fast_io_fail_tmo 10` – definira da se I/O operacije otkazuju nakon 10 sekundi u slučaju kvara putanje.
- `dev_loss_tmo "infinity"` – definira da uređaj ostaje u sustavu zauvijek, čak i nakon nakon gubitka putanje.
- `vpd_vendor "hp3par"` – ovo su dodatne informacije o proizvođaču uređaja na temelju VPD podataka.

4. Sekcija `Multipaths`, u kojoj se definiraju specifične postavke za individualne multipath uređaje, poput:

```
multipaths {
    multipath {
        wwid 3600508b400105e210000900000490000
        alias mpath1
    }
}
```

Opis:

- `wwid 3600508b400105e210000900000490000` – ovdje se definira jedinstveni identifikator za uređaj za pohranu. Odnosno u ovom primjeru, **WWID** je `3600508b400105e210000900000490000`. **WWID** osigurava da je uređaj ispravno identificiran bez obzira na promjene u fizičkim putanjama. On omogućuje preciznu kontrolu i konfiguraciju specifičnih uređaja.
- `alias mpath1` - ovdje definiramo prijateljsko ime (alias) za uređaj. U ovom primjeru, uređaj će biti poznat kao `mpath1`. Alias povećava čitljivost i upravljivost uređajima u sustavu. On olakšava administraciju i smanjuje mogućnost grešaka prilikom rada s uređajima.



Za više detalja o **SAN** sustavima i ***Fibre channel***, pogledajte poglavlja:
**13.12.1. Storage Area Network (SAN).**
**13.12.1.2. Fibre Channel (FC).**

**Izvori informacija: (1706),(1707),(1708),(1709),(1714), `man multipath.conf`.**

# 13.12. NAS/SAN sustavi i druge tehnologije

## NAS i SAN sustavi

Mrežno spojena spremišta podataka odnosno ***NAS*** (engl. *Network Attached Storage*) osiguravaju nam prostor za spremanje podataka preko mreže. Ovo su zapravo mrežni dijeljeni sustavi za pohranu (i dohvaćanje) podataka. Oni rade na razini datoteka (i naravno direktorija), koje pohranjujemo na njih, preko protokola za mrežno dijeljenje datoteka. Možemo reći da je ***NAS*** sustav, sustav odnosno računalo ili poslužitelj, koji preko mreže daje pristup svom diskovnom prostoru, na razini datoteka i direktorija. Ovakav sustav je, prema tome, specijaliziran za dijeljenje datoteka s klijentima odnosno drugim računalima na mreži.
***NAS*** sustav može biti izveden hardverski ili softverski, a uglavnom je kombinacija jednog i drugog. Dodatno, ovakvi sustavi obično imaju ugrađeno po nekoliko tvrdih diskova (ili SSD/NVMe diskova u bržoj varijanti) koji se po mogućnosti nalaze u odgovarajućem RAID polju. Kako smo naučili, to RAID polje je prvo potrebno particionirati te formatirati sa željenim datotečnim sustavom. Taj datotečni sustav se dalje koristi za pohranu a potom i dijeljenje podataka, preko nekog od protokola za mrežno dijeljenje datoteka, te tako postaje ***NAS*** sustav.



Svako dijeljenje datoteka preko mreže (engl. *Network Share*), korištenjem nekog od mrežnih protokola koji postoje za tu namjenu, možemo nazvati upotrebom kao ***NAS*** sustava.

Svaki ***NAS*** sustav (na računalu ili poslužitelju) dijeli pristup svojim datotekama i direktorijima pomoću mrežnih protokola za tu namjenu. Navest ćemo protokole za dijeljenje datoteka preko mreže, koji su najčešće u primjeni:

- ***NFS*** (*Network File System*) - koristi se uglavnom na operativnim sustavim Linux/Unix ili ponekad u Windows okruženju. **Za više detalja o *NFSu* pogledajte poglavlje: 25.8.7. NFS.**
- ***SMB/CIFS*** (*Server Message Block / Common Internet File System*) - koristi se uglavnom na Windows ili Linux okruženjima. Osim za dijeljenje datoteka koristi se i za dijeljenje pisača i drugih uređaja preko mreže te dodatne funkcionalnosti.
- ***AFP*** (Apple Filing Protocol) - koristi se za mrežno dijeljenje datoteka na ***Mac OS*** računalima.
- Ovoj kategoriji pripadaju i ***FTP*** (*File Transfer Protocol*) i ***TFTP*** (*Trivial File Transfer Protocol*) protokoli, s time da su oni jednostavniji i nemaju napredne mogućnosti kao gore navedeni protokoli.
  **Za više detalja pogledajte poglavlje: 25.8.2. TFTP i FTP protokoli**.
- Osim navedenih, često se koristi i ***WebDAV*** (*Web Distributed Authoring and Versioning*) koji je što se tiče funkcionalnosti negdje između ***FTP*** protokola i gore navedenih protokola.

Dakle ***NAS*** sustavi koriste diskove koji su particionirani i formatirani odnosno na kojima se nalazi neki datotečni sustav poput primjerice: ***NTFS***, ***XFS***, ***ext3*** ili **ext4**, **ZFS** i slično. Tom datotečnom sustavu se zatim preko protokola za dijeljenje datoteka preko mreže (primjerice: ***NFS***, ***SMB/CIFS***, ***FTP***, …) i pristupa preko mreže, kao da se radi o lokalnom disku spojenom na računalo.

S druge strane **SAN** sustavi rade na drugačiji način. Naime ***SAN*** (*Storage Area Network*) sustavi ne nude mrežno dijeljenje datoteka, već nam osiguravaju mrežni pristup takozvanom blok uređaju odnosno praktično *površini diska*. Naime ***SAN*** sustavi preko mreže dijele svoje diskove ili polja diskova vidljiva klijentskoj (drugoj) strani kao običan tvrdi disk, odnosno površinu tog diska. Takva površina diska je prazna, nema niti particija, niti datotečni sustav i prema tome se ponaša kao stvarna prazna površina (novo kupljenog diska). Nadalje takav disk se sastoji od blokova (podataka) kao i bilo koji lokalni ATA, SATA, SAS ili neki drugi disk. Za ovakvo mrežno dijeljenje praktično sirove površine diska, potrebi su mrežni protokoli koji nam osiguravaju ovakav pristup.
Neki od ***SAN*** protokola koji su često u upotrebi su:

- ***iSCSI*** *- za eksportiranje SCSI [diskova] preko TCP/IP protokola.*
- ***Fibre Channel over Ethernet*** *- za eksportiranje Fiber Channel [diskova] diskova preko ethernet mreže.*
- ***ATA over Ethernet*** *- za eksportiranje ATA [diskova] preko ethernet mreža.*

Nakon što se klijent preko nekog od gore navedenih protokola spoji (na površinu diska), takav disk se mora prvo particionirati te formatirati, kao da se radi o lokalnom disku odnosno disku spojenom na vaše lokalno računalo.

**Za SAN sustave pogledajte sljedeće poglavlje!**

***NAS*** i ***SAN*** sustavi dolaze ili kao komercijalni samostojeći uređaji tvrtki poput: ***EMC²/Dell, IBM, NetApp, Synology*** i sličnih ili kao specijalizirani (operativni) sustavi poput: ***NexentaStor***, ***TrueNAS***, ***Open-E*** i drugi.
Osim navedenih postoje i sustavi otvorenog programskog kôda (*open source*), kao što su: ***OpenFiler***, ***Open Media Vault***, ***FreeNAS***, ***Nas4Free*** i slični, koji se mogu instalirati na bilo koje računalo koje zadovoljava određene hardverske preduvjete.
***NAS*** i ***SAN*** sustavi obično imaju i cijeli niz naprednih mogućnosti poput: snimanja stanja u vremenu (*snapshot*), provjeru konzistencije podataka na površini diskova, sinkronizaciju podataka na udaljeni NAS/SAN sustav, upotrebu VLAN mreža i drugo. U konačnici ***NAS*** ili ***SAN*** sustavi, s obzirom na to da na nižoj razini implementiraju neku od RAID tehnologija s diskovima, osiguravaju nam:

- Bolje performanse diskovnog sustava, uz preduvjet odabira dovoljno brze računalne mreže [*10/100/200/400Gbps*].
- Otpornost na ispad (kvar) određenog broja diskova u RAID polju, ovisno koje je RAID polje odabrano.
- Centralizirano mjesto za pohranu podataka, koje se još eventualno može dodatno kopirati na drugi sustav.

## Hibridni sustavi za pohranu

Hibridni datotečni sustav **ZFS** (Zettabyte file system) inicijalno je razvija tvrtka ***Sun Microsystems***, a danas je u vlasništvu tvrtke ***Oracle***. Srećom ***ZFS*** je objavljen pod posebnom licencom otvorenog kôda koja je kompatibilna s primjerice operativnom sustavom **FreeBSD**, ali nažalost nije s Linuxom. Ipak, cijela funkcionalnost ZFS-a, razvijena je od početka i danas se održava i razvija u projektu imena **OpenZFS** pa je stoga ***ZFS*** podržan i u upotrebi pod Linuxom, već duži niz godina.

## Što je *ZFS* i zbog čega je uopće razvijen?

***ZFS*** je razvijen kao kombinacija RAID kontrolera odnosno takozvanog *volume managera* (poput Linuxovog **LVM2**) i datotečnog sustava u jednom. Dakle unutar njega se kreiraju ekvivalenti RAID polja diskova, na koja se automatski instalira napredni ***ZFS*** datotečni sustav. Ekvivalenti RAID polja koje ***ZFS*** standardno podržava su:

- Takozvani *single device*, odnosno instalacija na jedan jedini disk, pa prema tome nema redundancije (zalihosti).
- **RAID-Z** polje, znano i kao **RAID-Z1**, ono je ekvivalent RAID-5 tehnologiji s jednim *partitetnim* diskom.
- **RAID-Z2** - ono je ekvivalent RAID-6 tehnologiji s dva *paritetna* diska.
- **RAID-Z3** - ono je slično RAID-6 tehnologiji, ali s tri *paritetna* diska.
- Takozvani *mirror device*, odnosno zrcalna kopija, koja je ekvivalent RAID-1 polju diskova.
- Ekvivalent RAID-10 polju.

***ZFS*** je razvijen kako bi riješio poznate probleme s RAID-5 i RAID-6 poljima *[***(1172)***]* te pojednostavio zamjenu neispravnog diska i proširenje postojećeg polja diskova. Nadalje, on nam u slučaju kvara cijelog računala/poslužitelja omogućuje bezbolno prebacivanje diskova u novo računalo, na kojem je iznimno jednostavno i pouzdano vratiti staro polje diskova i što je još važnije sačuvati sve podatke u tom procesu.

Također je važno da ZFS ne uopće ne treba hardverski RAID kontroler, štoviše ne bi ga niti smjeli koristiti jer **ZFS** diskovima zbog optimizacija u radu pristupa na najnižoj mogućoj razini. On sve operacije zapisivanja radi kao takozvane transakcije (tzv. *Copy-On-Write (COW)*), dakle na siguran i optimiziran način, koji između ostalog drastično ubrzava izradu kopija podataka u vremenu (tzv. *snapshot*).

Osim toga on podržava i mnoge druge napredne funkcije poput navedene izrade kopije stanja podataka u vremenu (tzv. *snapshot*), periodičku provjeru konzistencije podataka (tzv. *data scrubbing*) na polju diskova, replikaciju stanja podataka (*snapshota*) na udaljeni **ZFS** poslužitelj (tzv. `zfs send– zfs receive`), ubrzavanje operacija snimanja (tzv. ***ZIL***) ili čitanja (tzv. ***ARC*** ili ***L2ARC***) na polje diskova, upotrebom SSD ili NVMe diska ili polja diskova, te mnoge druge stvari.

Važna značajka ovog sustava je i ta da se nakon kreiranja polja diskova (ekvivalenta nekog od RAID polja), polje može automatski formatirati sa ***ZFS*** datotečnim sustavom i koristiti kao bilo koji drugi datotečni sustav i prema potrebi kasnije eksportirati preko mrežnog dijeljenog sustava odnosno protokola poput: ***NFS***, ***SMB/CIFS*** i drugih. Dakle jednostavno se može koristiti kao ***NAS*** sustav ili je moguće eksportirati ga kao ***SAN*** sustav preko nekog od protokola za tu namjenu (pr. ***iSCSI***).

Na slikama 127.1., 127.1.A i 127.2. vidimo poslužitelje tvrtki ***HP*** i ***Dell***, koji su namijenjeni za pohranu podataka (tzv. *storage* sustavi).

**Slika 127.1. Dell EMC SCv3000 poslužitelj koji može prihvatiti do 30 diskova.**



**Slika 127.1.A. Dell PowerStore 3200Q**



Slika 127.2. HP ProLiant SL4540 poslužitelj koji može prihvatiti do 48 diskova.



## Redundantni i klasterski sustavi

### Mrežno redundantni sustavi

Klasični mrežni redundantni sustavi su obično takvog (jednostavnijeg) dizajna u kojemu se ispred njih logički nalazi sustav koji osigurava pristup jednoj jedinoj (virtualnoj) IP adresi kojoj klijenti i pristupaju, mada postoje i drugačije implementacije, ali ova je najčešća. Ovakvi sustavi su često izvedeni sa samo dva ***NAS*** sustava koji rade prema principu aktivnog i pričuvnog uređaja (engl. *Active-Standby*). Pri tome je prvi ***NAS*** uređaj stalno aktivan i na njega i s njega se zapisuju podaci dok se svaki novo zapisan podatak u pozadini sinkronizira na pričuvni uređaj. Dakle u pozadini se takav sustav brine da se svi podaci uredno kopiraju s aktivnog na pričuvni ***NAS*** sustav. U slučaju kvara aktivnog sustava pričuvni sustav preuzima njegovu funkciju i svi podaci su sačuvani. Ovakvi sustavi mogu biti implementirani u hardveru, a mogu biti ***NAS*** ili ***SAN*** sustavi.

Međutim postoje i softverska rješenja, čak i otvorenog programskog kôda, poput ***DRBD*** sustava (engl. *Distributed Replicated Block Device*). ***DRBD*** sustav praktično radi tako da se aktivni odnosno primarni sustav (***NAS*** poslužitelj) stalno sinkronizira (replicira) s pričuvnim sustavom, naravno preko mreže. ***DRBD*** funkcionira kao logički blok uređaj, vidljiv kao `/dev/drbdXY`. Prema logici rada on se ponaša poput RAID-1 zrcalnog polja diskova, ali koje se sinkronizira preko mreže.

### Klasterski sustavi

Sljedeća kategorija mrežnih dijeljenih sustava za pohranu i dohvaćanje podataka su sustavi koji se nalaze u takozvanom *klasteru* (*grozd*), a oni su znatno kompleksnijeg dizajna koji uključuje i razne dodatne softverske, ali često i hardverske komponente.
Svi klijenti u pravilu pristupaju vršnoj klasterskoj komponenti, koja je često izvedena samo u softveru, a koja se brine za raspodjelu podataka unutar klastera, na pojedine uređaje (poslužitelje) za pohranu podataka. Sama replikacija svih podataka između pojedinih poslužiteljskih sustava se često izvodi i u softveru i na specijaliziranom hardveru.
Zbog ovakvog, složenog dizajna i potrebe za posebnim hardverom i njihova cijena je poprilično veća od redundantnih **NAS** sustava.

### Softverski klasterski sustavi

Osim sustava koji zahtijevaju i poseban hardver, postoje i softverska rješenja, od kojih ih postoji i nekoliko otvorenog programskog kôda. Neki od njih su:

- **GlusterFS** (*Red Hat Gluster Storage*) – on omogućuje više mogućih načina redundancije s upotrebom većeg broja poslužitelja. On podržava mnoge ekvivalente RAID polja: RAID0 ili RAID1 (min. 2. poslužitelja), RAID10 (min. 4. poslužitelja i slično. Dakle on podržava sinkronizaciju i replikaciju između više poslužitelja.
- **GFS2** (*Global Filesystem* v.2.) **i OCFS2** (*Oracle Cluster File System*) podržavaju replikaciju podataka i donekle su slični prema klasterskom načinu rada.
- **Lustre** – on nudi rad s još većim sustavima, a prema načinu rada je distribuirani klasterski datotečni sustav.
- U primjeni su još i: **MooseFS** (*Moose File System*), **BeeGFS** (*Bee File System*) i nekolicina drugih.

**Problemi klasterskih sustava**
Čak i klasterski ***NAS*** i ***SAN*** sustavi imaju svoje limitirajuće faktore. Kod većine je to cijena, ali i ograničenja skalabilnosti. Naime većina informatički sustava s vremenom treba sve veći i veći kapacitet pohrane podataka, koji postaje ili preskup u početku ili zahtjeva vrlo velika ulaganja tijekom proširenja. I na kraju krajeva svi oni opet imaju svoja ograničenja, najviše po pitanju proširenja sustava. Tvrtkama koje pružaju usluge i servise u *oblaku*, pružanje usluge pohrane velike količine podataka kao što je primjerice pohrana i rad s virtualnim računalima ili raznim servisima i uslugama, svakodnevni je posao. Proširivost ovakvih sustava je stoga krucijalna. Rani odgovor na ovu problematiku je bio razvoj i kasnija upotreba, sustava koji rade na potpuno drugačiji način u odnosu na tradicionalni klasterske ***NAS*** ili ***SAN*** sustave.

## I rodio se objektni sustav za pohranu podataka (engl. *object storage*)
Dakle sustav koji podatke promatra i pohranjuje kao objekte, za razliku od tradicionalnih sustava kod kojih postoji neka struktura datoteka i direktorija odnosno klasičan datotečni sustav (kod ***NAS*** sustava) ili blokovi diskovnog prostora (kod ***SAN*** sustava). Dakle objektni sustav je drugačiji i od ***SAN*** sustava koji rade s blokovima podataka koji se spremaju u sektore na disku, bilo logičkom ili fizičkom. Kao što RAID kontroler razlama neku datoteku na male blokove podataka koje dalje raspoređuje na diskove, ovisno o RAID polju, tako i objektni sustavi razlamaju podatke na takozvane objekte (uz pripadajuće metapodatke), koje onda raspodjeljuju na poslužitelje u *klasteru*.
Objektni sustav za pohranu trebao bi nam nuditi, skalabilni (proširivi) sustav otporan na greške.
Ovakvi sustavi su se počeli znatnije razvijati od 1995. godine iako su neki radovi i ideje nastali i znatno ranije.
Prvo komercijalno rješenje je razvila tvrtka *Centera Technology* koju je odmah kupila tvrtka **EMC²** te ga je 2002. godine izbacila na tržište pod tržišnim nazivom "***EMC Centera***". Ova linija proizvoda se i danas razvija.

Smatra se da se u razvoj ove tehnologije od strane neovisnih investitora u prvim godinama uložilo oko 300 milijuna dolara, a ova brojka je rasla sve više. Ne računajući ulaganja tvrtki poput: ***DataDirect Networks***, ***Centera***, ***Atmos***, ***HDS***, ***EMC²*** i kasnije ***Dell***, ***HP***, ***IBM***, ***NetApp***, ***Red Hat*** i drugih. Kasnije su se ovakvi sustavi razvijali i od strane tvrtki koje pružaju usluge i servise u oblaku, poput: ***Amazon AWS, Microsoft (Microsoft Azure)***, ***Google (Google Cloud Storage)*** i drugih.
Pogledajmo listu nekoliko visoko skalabilnih, redundantnih objektnih sustava dostupnih pod licencama otvorenog kôda:
- Lustre, LizardFS, *Hadoop distributed file system* (***HDFS***).
- Moose File System, OpenAFS, Quantcast File System, CEPH i drugi.

Kod većih sustava, kao i kod sustava kod kojih korisnici ne žele kupovati ekstremno skupi hardver i softver za objektne sustave za pohranu, jedno od rješenja otvorenog kôda je ***CEPH*** o kojem ćemo govoriti dalje u tekstu.

## CEPH
***CEPH*** je distribuirani objektni sustav za pohranu podataka koji je dizajniran za postizanje vrhunskih performansi, te sustav koji je visoko dostupan i pouzdan. Osim toga on je krajnje skalabilan odnosno proširiv, čak do razine *egzabajta* (EB).
1 EB = 1.000 PB = 1.000.000 TB.

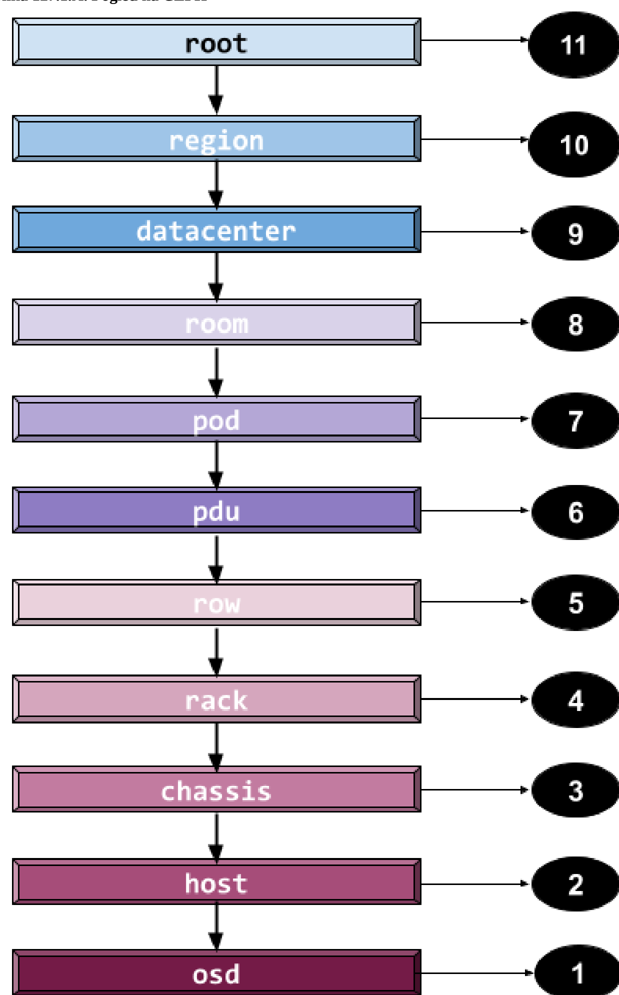
Ovo je sustav koji je zbog svog dizajna otporan na greške i kvarove cijelih poslužitelja i/ili pojedinačnih diskova ili grupe diskova, a u većim implementacijama, cijelih ormara punih poslužitelja pa čak i cijelih podatkovnih centara, a samim time i stotinama ili desecima tisuća diskova. Sve ovisno o konfiguraciji i raspoloživoj opremi.
***CEPH*** je razvio ***Sage Weil*** kao temu za doktorski rad na sveučilištu *University of California, Santa Cruz*.
Razvoj se nastavio u tvrtki ***Inktank***. Navedenu tvrtku je kupio ***Red Hat***, 2014. godine za 175 milijuna U$ u gotovini.
Tvrtka ***Red Hat*** ga nastavlja razvijati do danas (kao i zajednica koja ga koristi). Projekt je i dalje, i ostat će otvorenog kôda, a njegovom razvoju su se pridružile tvrtke: ***Intel***, ***Cisco***, ***Fujitsu***, ***SanDisk***, ***Canonical*** i ***SuSe*** te ***CERN*** i mnogi pojedinci.
S obzirom na njegovu sve veću prihvaćenost, a vrlo brzo nakon učlanjenja u obitelj **Red Hat**, za njega postoji i podrška od strane proizvođača hardvera. Naravno, svi važniji proizvođači hardvera počeli su nuditi sustave koji su certificirani za *CEPH*, poput tvrtki ***HP***, ***Dell***, ***Supermicro*** i drugih. Osim navedenog hardvera, ***CEPH*** se može koristiti i na bilo kojem hardveru koji posjedujete, a na kojem se može pokretati bilo koja ***Red Hat*** ili ***Debian*** bazirana distribucija Linuxa, novije generacije.

Dakle za njegov rad su dostupni i ***RPM*** i ***Debian*** (***deb***) softverski paketi.
Osim toga dostupan je i izvorni kôd *CEPH*a, pa je sve moguće kompilirati i za druge distribucije Linuxa, ako za to postoji potreba.
Nadalje, CEPH je odlično integriran s barem dvije platforme za virtualizaciju: ***OpenStack*** i ***Proxmox VE***. Primjerice instalacija, konfiguracija i upotreba ***CEPHa*** danas (2021.g.), iz platforme ***Proxmox VE*** je trivijalna i u potpunosti iz grafičkog sučelja.

**Tko ga sve koristi?**
- ***Amazon AWS*** - prema nekim informacijama koristi se za neke dijelove ***S3 Storage*** sustava.
- ***Facebook*** – za određene dijelove sustava.
- ***CERN*** **–** koriste ga za pohranu podataka: 2017.g. instaliran je na 1.500 poslužitelja s 55.000 diskova i ukupnim kapacitetom od 220PB [220.000 TB]. Koriste ga i mnoge druge korporacije i tvrtke.

***CEPH*** je zamišljen da se može koristiti u sljedećim scenarijima:
- Kao blok uređaj i to ako se koristi kao *Rados Block Device* (**RBD**), koji je tada vidljiv kao blok uređaj (disk) ili logički disk koji se koristi za opću primjenu: primjerice za spremanje diskova virtualnih računala, Linux kontejnera i slično.
- Kao klasični objektni sustav za pohranu, upotrebom takozvanog **RADOSGWa**, koji je **S3** i **Swift** kompatibilan, a obično se koristi za pristupanje datotekama bilo koje vrste preko weba: pomoću *HTTP PUT* ili *HTTP GET* metoda.
- Kao datotečni sustav upotrebom **CephFS**, a tada se može montirati kao klasičan (***POSIX***) datotečni sustav.

Slika 127.1.A. Pogled na CEPH




**Distribuiranje podataka unutar CEPH klastera**
Za distribuiranje odnosno raspodjeljivanje podataka unutar *CEPH klastera* koristi se takozvani **CRUSH** algoritam i pripadajuća CRUSH tablica koja je distribuirana na više poslužitelja zbog redundancije (zalihosti).

Ova tablica i samim time mehanizam su zaduženi za distribuciju, replikaciju i redistribuciju podataka unutar *CEPH klastera*. CRUSH je dizajniran da omogućava raznoliku upotrebu, ovisno o veličini implementacije. Prema tome postoje CRUSH tipovi koji opisuju fizičku poziciju *CEPH* poslužitelja i diskova (*OSD)* unutar cijelog *CEPH klastera*.

Drugim riječima definira se fizička hijerarhijska strukturu svakog elementa unutar hijerarhije, kako je vidljivo na slici 127.1.A. Dakle logika iza ovoga je da sustav treba biti svjestan hijerarhijske pozicije, od diskova u poslužiteljima, smještaja poslužitelja u informatičke ormare, nizove ormara, prema napajanjima, dijelovima podatkovnih centara, podatkovnim centrima, regijama (praktično kontinentima) i u konačnici vrhu hijerarhije.

Naime nije svejedno nalaze li se dva CEPH poslužitelja s diskovima u istom informatičkom ormaru povezani s brzom mrežom ili su u različitim ormarima ili drugim dijelovima podatkovnog centra ili su recimo pozicionirani u dva različita podatkovna centra ili se pak nalaze u različitim regijama (pr. državama ili čak kontinentima).

***Pogledajmo što sve CEPH razlikuje u hijerarhiji:***

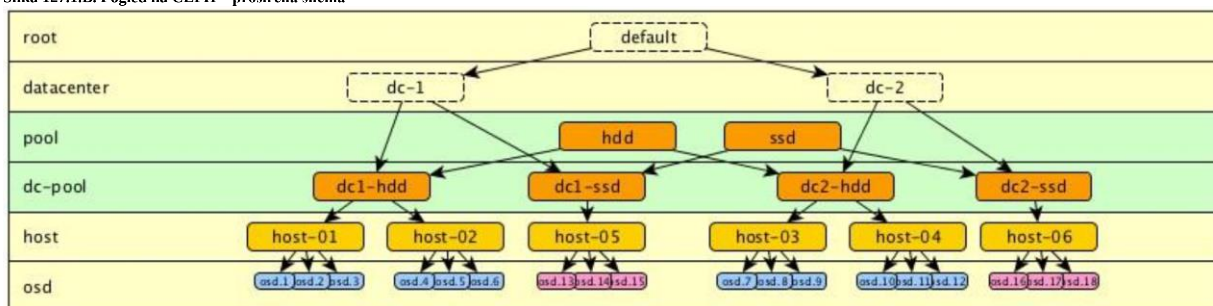
**11.** Na vrhu hijerarhije je takozvani „**root**" (*korijen stabla*).
**10. Region** - predstavlja regiju (pr. grad, državu i sl.).
**9. Datacenter** je podatkovni centar s CEPH poslužiteljima.
**8. Room** – predstavlja sobu unutar podatkovnog centra.
**7. Pod** – predstavlja logičku podjelu unutar jedne sobe u podatkovnom centru (pr. vezu na druge preklopnike, s drugim napajanjem, agregatima za napajanje ili UPS uređajima).
**6. PDU** – ovo je podjela unutar podatkovnog centra prema izvorima napajanja (napajanje, agregati i UPS uređaji) jer i na to treba paziti, ako primjerice jedan dio ostane bez napajanja, drugi dio podatkovnog centra treba uredno raditi.
**5. Row** – predstavlja jedan red s informatičkim ormarima unutar kojih se nalaze *CEPH* poslužitelji s diskovima.
**4. Rack** – predstavlja jedan informatički ormar s *CEPH* poslužiteljima.
**3. Chassis** – predstavlja jedno kućište – primjerice ako se koriste ***Blade*** poslužitelji koji se mogu nalaziti unutar zajedničkog kućišta.
**2. Host** – predstavlja poslužitelj s diskovima za *CEPH*.
**1. OSD** – predstavlja svaki pojedini disk koji će se koristiti u *CEPH* klasteru.

U konačnici ovakav hijerarhijski model omogućava nam raznolike scenarije upotrebe, kako veličinom: od minimalno tri [3] CEPH poslužitelja s diskovima, do tisuća poslužitelja s diskovima, čak raspoređenim u više podatkovnih centara, raštrkanih u više gradova ili (geografski) šire.

Osim toga ova hijerarhija može biti i proširena za različite namjene i logike rada, kao što je vidljivo na slici 127.1.B.
Prema logičkoj shemi na slici 127.1.B vidimo primjer u kojem smo u definiciji ***Pool-a*** kreirali dvije strukture: jednu za mehaničke diskove (hdd), a drugu (ssd) za SSD diskove. U konačnici su diskovi (osd) spojeni da poslužitelje (host), tako definirani u hijerarhiji, da su mehanički diskovi spojeni na (hdd), a SSD diskovi na (ssd) ***Pool***, koji predstavljaju ekvivalent polja diskova.

Dakle hijerarhija u primjeru je definirana za određene vrste diskova.

**Slika 127.1.B. Pogled na CEPH – proširena shema**




Izvor fotografije: https://ceph.io/geen-categorie/crushmap-example-of-a-hierarchical-cluster-map/

**Zapisivanje podataka u CEPH klaster**

Nakon što je definirana hijerarhijska struktura za *CEPH* klaster (***CRUSH***) te kreiran ekvivalent RAID polja koji se prema *CEPH* terminologiji naziva **Pool**, sve je spremno za rad. Pojednostavljeno svaka datoteka koja se zapisuje, lomi se na manje blokove, koji se onda u konačnici zapisuju odnosno distribuiraju na dostupne poslužitelje i njihove diskove.
Primjerice ako smo za određeni **Pool** na kojem radimo, tijekom njegovog kreiranja odabrali da je broj **replika** odnosno prema *CEPH* terminologiji **CEPH Pool Size** jednak tri (3), to znači da se podaci zapisuju na odredišni poslužitelj, a potom na još druga dva (2) poslužitelja. Tako da ćemo u ovom slučaju isti podatak imati sveukupno na tri (3) mjesta.
Veličina bloka je standardno 4 MB, ali se može promijeniti do razine više MB - ovisno o vrsti podataka koje zapisujemo ili čitamo.

To znači da je za neke primjene ova veličina zadovoljavajuća, a za neke je ova veličine premalena jer se zapisuju ili čitaju podaci koji zahtijevaju dohvaćanje većih blokova podataka odjednom. Promjenom veličina bloka možemo poboljšati performanse i smanjiti opterećenje sustava; zbog smanjenja broja operacija dohvaćanja velikog broja malih objekata.
Ulazno/izlazne operacije prema diskovnom sustavu tijekom operacija pisanja ili čitanja se zovu **IOPS**.
Klasični mehanički diskovi su znatnije pogođeni ovim operacijama od SSD diskova.

Dakle SSD diskovi u prosjeku mogu podnijeti desetke, stotine i tisuće puta više ulazno/izlaznih operacija u sekundi, od mehaničkih/magnetskih diskova, pa pri dizajnu treba paziti na pravilan odabir i kapaciteta i vrste diskova: mehanički ili SSD/NVMe odnosno moguća je i razdioba i njihova istovremena upotreba, kako smo vidjeli u primjeru na slici **127.1.B.**
Dakle moguće je primjerice imati dio ***CEPHa*** s mehaničkim diskovima, za pohranu veće količine podataka, a za one podatke kojih ima manje, ali zahtijevaju brži pristup, SSD ili NVMe diskove.

**Distribucija podataka**

Prilikom zapisivanja podaci se distribuiraju na cijeli *CEPH* klaster, sve njegove poslužitelje i njima dostupne tvrde diskove, te se istovremeno radi replikacija svakog bloka podataka na drugi poslužitelj odnosno disk na njemu. Sve prema tome kako je konfigurirana hijerarhija za CRUSH te koliko replika smo odabrali za određeno *CEPH* polje odnosno **CEPH Pool**.
Proces zapisivanja i dodatno replikacije, radi se transakcijski (poput **ZFS** transakcijskog modela), sve zbog konzistentnosti podataka.

Kod procesa čitanja se također prema klasterskoj tablici i CRUSH algoritmu određuje to jest izračunava koji blok podataka je završio na kojem poslužitelju, i na kojem disku na njemu. Zatim se počinje s čitanjem blokova podataka sa svih poslužitelja i svih diskova. U konačnici sve se svodi na to da se podaci zapisuju na sve *CEPH* poslužitelje i njihove dostupne diskove te se kod čitanja također čitaju sa svih njih. Ovime su se znatno povećavaju performanse: što je više poslužitelja u *CEPH* klasteru, to je brže zapisivanje ili čitanje jer se ono odrađuje paralelno.

**Redistribucija podataka**

U određenim slučajevima, kada se pojedini *CEPH* poslužitelj gasi (sa svim svojim diskovima [**OSD**ovima]) ili se dodaje novi ili se dodaju ili vade diskovi iz nekih poslužitelja, tada *CEPH* radi redistribuciju podataka.
Naime kod svakog dobro balansiranog sustava, postotak zauzeća i upotrebe svih diskova mora biti podjednak. Za to su zaduženi automatizmi o kojima ćemo govoriti. Dodavanjem novog diska, vađenjem jednog od njih ili dodavanjem ili izbacivanjem cijelog poslužitelja sa svim diskovima *CEPH* kreće u redistribuciju svih podataka. To znači da ako smo recimo dodali novi poslužitelj s diskovima (detaljnije: radi se i o koeficijentu svakog diska ovisno o njegovom kapacitetu i drugim parametrima) podaci se preraspodjeljuju unutar cijelog klastera odnosno na svim diskovima, tako da svi diskovi na svim poslužiteljima budu podjednako zauzeti.
Ovo je vrlo važno jer se nakon dovršetka redistribucije podaci tada počinju zapisivati ili čitati i s novih poslužitelja ili novih diskova, ravnomjerno koristeći sve resurse (poslužitelje i diskove) *klastera* te poboljšavajući brzinu i propusnost sustava.

Za redistribuciju kao i za replikaciju podataka, koristi se (preporučuje) zasebna mreža, kako se ne bi opteretila "radna" mreža. Prema *CEPH* preporukama, potrebno je imati dvije zasebne mreže dovoljne propusnosti:

- ***Public Network** – [javna mreža]* - preko nje čitamo i zapisujemo podatke na *CEPH* (kao *CEPH* klijenti).
- ***Cluster Network** – [klasterska mreža]* - preko nje se odrađuju sve ostale radnje poput redistribucije i replikacije podataka.

**Sve Ceph uloge i detalji**

Za svaki od načina upotrebe ***CEPHa***, potrebne su određene funkcionalnosti odnosno uloge *CEPH* poslužitelja:

Uloga *Ceph Monito*ra (***MON***) zaslužna je za održavanje karte stanja klastera, uključujući mapu svih *monitora* (**MON**), **OSD** mapu, **MDS** mapu i *CRUSH* tablicu. Ove karte su kritične za stanje klastera i potrebne su da se servisi *Cepha* međusobno koordiniraju. ***Monitori*** su također odgovorni za upravljanje provjerom autentičnosti između servisa i klijenata. Za redundantnost i visoku dostupnost obično su potrebna najmanje tri *monitora*.

**OSD** (engl. *object storage daemon*) uloga zaslužna je za pohranjivanje podataka, upravljanje replikacijom podataka, obnavljanjem, ponovnim balansiranjem to jest redistribucijom podataka te pružanjem informacija o nadzoru *Ceph* monitorima i upraviteljima (*Ceph menadžerima*) provjeravanjem dostupnosti drugih *Ceph OSD* servisa. Za redundantnost i visoku dostupnost obično su potrebna najmanje tri *Ceph OSD-a*. *OSDovi* predstavljaju i diskove.

***Ceph menadžer*** (*ceph-mgr*) odgovoran je za praćenje metrike vremena izvođenja i trenutnog stanja *Ceph* klastera, uključujući korištenje pohrane, trenutne metrike performansi i opterećenja sustava. Servisi *Ceph Menadžera* koriste module obično temeljene na *Pythonu* za upravljanje i izlaganje informacija *Ceph* klastera, uključujući web-bazirani „*Ceph Dashboard*" te ***REST API***. Za visoku dostupnost obično su potrebna najmanje dva upravitelja to jest *Ceph menadžera*.

Kod uporabe CephFSa odnosno, ako imamo potrebu koristiti *CEPH* datotečni sustav (***CephFS***), za to postoji uloga koja se zove ***Metadata server*** (***MDS***), koja je potrebna za spremanje metapodataka za ***CephFS*** datotečni sustav. *Ceph Metadata Serveri* omogućuju korisnicima (**POSIX)** datotečnog sustava da izvršavaju osnovne naredbe nad datotekama ili direktorijima, poput naredbi: **ls**, **find** i drugih, bez nepotrebnog opterećivanja *Ceph* klastera.

**Zbog čega je za najmanji mogući sustav potrebno barem tri poslužitelja?**

Većina klastera u radu rade prema principu *kvoruma*; dakle tri je najmanji broj poslužitelja u kojemu minimalna većina (dva) poslužitelja sudjeluju u dogovaranju i provjerama rada klastera. Ovdje se radi o sustavu glasovanja i izbora što znači da svaki poslužitelj ima jedan glas za glasovanje. Ako su samo dva poslužitelja u sustavu glasovanja, izbori su nemogući. Prema tome za sustav glasovanja je potrebno minimalno tri poslužitelja. U ovakvim minimalnim klasterima s tri poslužitelja, u svakom trenutku moraju biti aktivna i funkcionalna dva poslužitelja.
Ovo čak ne mora značiti da je jedan poslužitelj ugašen već možda ne radi kako treba, pa primjerice daje krive rezultate (ili ih ne daje uopće), a tada se zadnja dva poslužitelja pokušavaju sustavom glasovanja dogovoriti o radu *CEPH* klastera, njegovoj funkcionalnosti i pouzdanosti.



**Ponovimo:** uloga monitor (**MON**) poslužitelja je praćenje svih dostupnih poslužitelja i diskova u *klasteru* i pohrana nekoliko tablica: tablice sa svim **MON** poslužiteljima, tablice sa svim **OSD** poslužiteljima i diskovima, tablice s grupama za pohranu podataka (*Placement Group*) te CRUSH tablice s hijerarhijom u CEPH klasteru. **OSD** uloga je zadužena za pohranu podataka na diskove te replikacijom i redistribucijom podataka.

**Minimalne potrebne funkcionalnosti za CEPH kao blok uređaj (*Rados Block Device [RBD]*)**

Primjerice za upotrebu *CEPH*a kao blok uređaja to jest kao **RBD**a, potrebne su nam dvije funkcionalnosti odnosno uloge poslužitelja. To prema definiciji znači da moramo imati poslužitelje od kojih je svaki zadužen samo i isključivo za jednu ulogu:

- Ulogu takozvanog monitor poslužitelja (**MON**), na minimalno tri poslužitelja.
- Ulogu takozvanog **OSD** poslužitelja s diskovima za CEPH klaster, na minimalno tri poslužitelja.

U najgorem slučaju **OSD** poslužitelji mogu imati i **MON** ulogu, ako nemamo više od sveukupno tri poslužitelja.
Kod procesa čitanja ili pisanja na blok uređaj (***CEPH RBD***) klijent koji želi nešto zapisati ili pročitati iz *CEPH* klastera koji koristi kao blok uređaj, prvo se kontaktira *CEPH* klaster i to **MONitor** poslužitelje te se od njih traži tablica s konfiguracijom klastera. *CEPH* klaster ***MONitor*** poslužitelj(i) mu šalju traženu tablicu to jest konfiguraciju *CEPH* klastera s hijerarhijom.

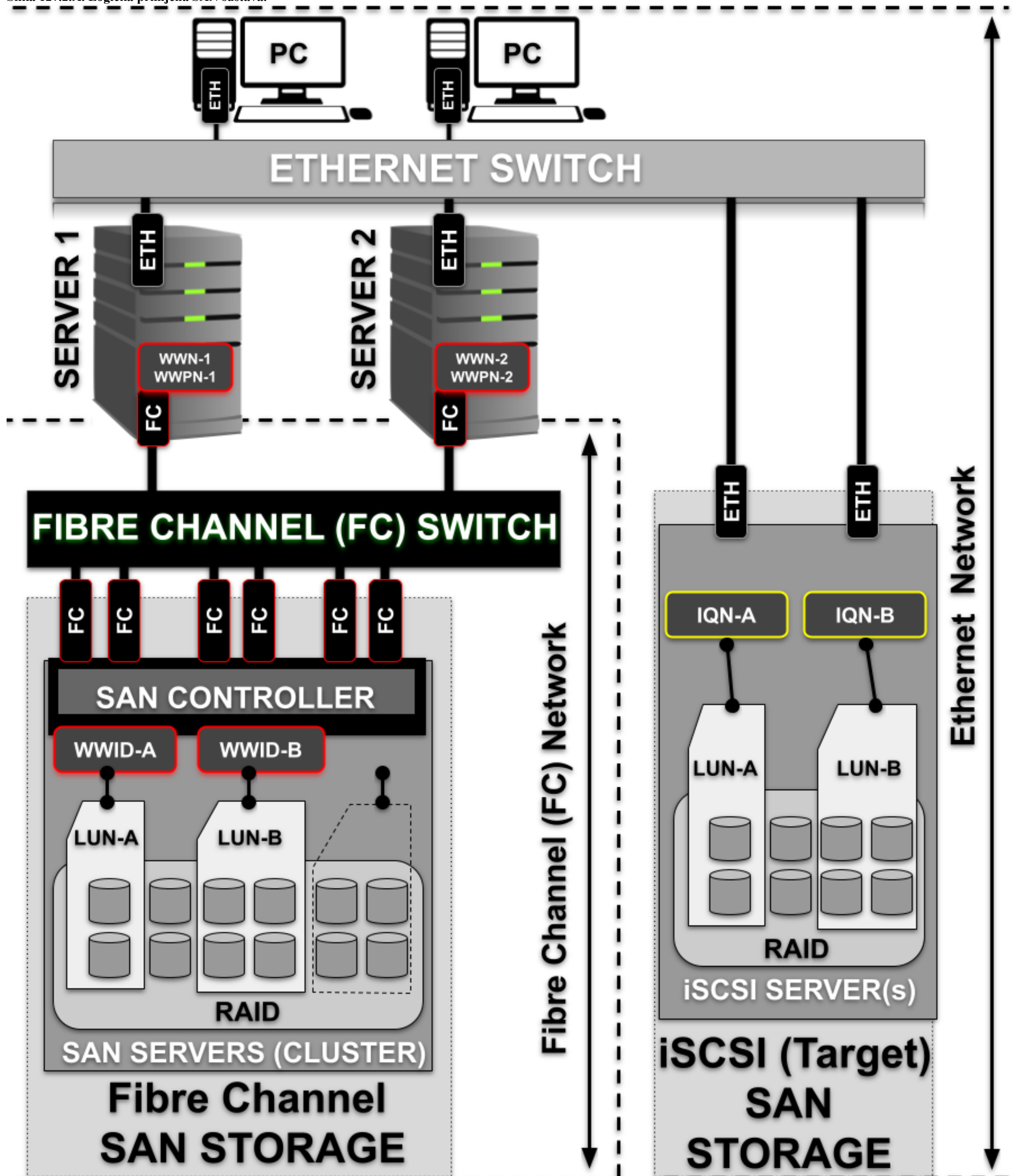
Na osnovi tablice odnosno konfiguracije koju je dobio, klijent pomoću *CRUSH* algoritma, on traži od **OSD** poslužitelja i **OSD** diskova zahtjev za čitanje ili traži zapisivanje. On potom s **OSD**ova dobiva odgovor na traženi zahtjev (čitanje ili pisanje).

**Izvori informacija:(K15),(451),(1162),(1163),(1164),(1165),(1166),(1167),(1168),(1169),(1170),(1171),(1172), (1173),(1174),(1175),(1176), (1177),(1178),(1179).**

## 13.12.1. Storage Area Network (SAN)

***Storage area network*** (*SAN*) uobičajeno je specijalizirana mreža koja omogućuje pristup sustavu za pohranu podataka na razini blokova, a ne datoteka. *SAN*-ovi se primarno koriste za pristup uređajima za pohranu podataka, kao što su poljda diskova ili tračni uređaji, tako da se ti uređaji operativnom sustavu prikazuju kao izravno priključen sustav za pohranu odnosno disk to jest kao blok uređaj. *SAN* je obično namjenska mreža kojoj se ne može pristupiti putem lokalne (LAN) ***Ethernet*** mreže već preko specijalizirane ***Fibre Channel*** mreže. Mada postoje i varijante u kojima se za pristup ***SAN*** sustavu za pohranu može koristiti i klasična ***Ethernet*** mreža i TCP/IP skup protokola, a sve pomoću **iSCSI** protokola.

**Slika 127.2.A. Logička primjena SAN sustava.**




Na slici su vidljive dvije najčešće tehnologije za primjenu **SAN** sustava:

- **Fibre Channel (FC)** – specijalizirane *Fibre Channel* mreže (*FC* preklopnika i *FC* mrežnih adaptera) te specijaliziranih ***Fibre Channel SAN*** sustava za pohranu (*SAN* poslužitelja i kontrolera kao krajnje točke). Ovdje je vidljivo i to da se **LUN**-ovima preko **FC**, pristupa pomoću **WWID** oznaka odnosno adresa.
- **iSCSI** – upotrebom standardnih (klasičnih) *Ethernet* preklopnika i **iSCSI** protokola koji koristi TCP/IP protokol te krajnjih poslužitelja s diskovima kao iSCSI odredištem (***iSCSI Target***). Dok je ovdje vidljivo da se **LUN**-ovima na *iSCSI* sustavu za pohranu prostupa preko **IQN** oznaka odnosno adresa.

Iako ***SAN*** sustav pruža samo pristup na razini blokova, datotečni sustavi izgrađeni na SAN-ovima pružaju pristup na razini datoteka. To znači da pristupom na ***SAN*** sustav dobivamo pristup diskovnom blok uređaju (ili na više njih), koji možemo normalno particionirati i formatirati, kao da se radi o lokalnom disku. Pogledajmo ispis diskova na sustavu koji ima jedan lokalni disk (`/dev/sda`) i diskove `/dev/sdb` i `/dev/sdc` dostupne preko *Fibre Channel* mreže.

```
lsscsi
[0:0:0:0]    storage  HP         P220i           8.32  -
[0:1:0:0]    disk     HP         LOGICAL VOLUME  8.32  /dev/sda
[1:0:0:0]    disk     3PARdata   VV              3226  /dev/sdb
[1:0:0:254]  enclosu  3PARdata   SES             3226  -
[1:0:1:0]    disk     3PARdata   VV              3226  /dev/sdc
[1:0:1:254]  enclosu  3PARdata   SES             3226  -
```

← Diskovi `/dev/sdb` i `/dev/sdc` su spojeni preko *Fibre Channel kartice* (***HBA***) na ***SAN*** sustav za pohranu.



Za primjere upotrebe **SAN** sustava pogledajte poglavlja:
**13.11.3.1. Device Mapper Multipath (Multipath I/O).**
**13.12.1.1. iSCSI.**
**13.12.1.2. Fibre Channel (FC).**

Moguće su i napredne i kompleksnije primjene ***SAN*** sustava, primjerice povezivanjem na više dostupnih puteva do ***SAN*** sustava; primjerice upotrebom ***Fibre Channel*** ili ***iSCSI*** sustava.
Takav pristup se može koristiti upotrebom **Multipath I/O (MPIO)** konfiguracije (hardverske i softverske).



Za više detalja o **Multipath I/O (MPIO)** sustavima, pogledajte poglavlje:
**13.11.3.1. Device Mapper Multipath (Multipath I/O).**

Pogledajmo usporednu tablicu sa značajkama **NAS** i **SAN** sustava:

| Kriterij | NAS (Network Attached Storage) sustav | SAN (Storage Area Network) sustav |
|---|---|---|
| **Definicija** | NAS je datotečni sustav za pohranu koji se povezuje s (*Ethernet*) mrežom i TCP/IP protokolom te omogućuje pristup datotekama preko standardnih mrežnih protokola poput: **NFS**, **SMB/CIFS**, ... | SAN je diskovni blok-sustav za pohranu koji se povezuje s poslužiteljima uobičajeno putem namjenskih mreža (*Fibre Channel*), omogućujući brzi pristup blokovima podataka kao da su lokalni diskovi. |
| **Protokoli** | **NFS, SMB/CIFS, HTTP, FTP** | **iSCSI, Fibre Channel (FC), FCoE** |
| **Tip podataka** | Datotečni | Blokovi |
| **Performanse** | Manje performanse u usporedbi sa SAN-om zbog mrežnih protokola i mogućih zagušenja mreže. | Visoke performanse zbog namjenskih mreža (pr. ***Fibre Channel***) i izravnog pristupa blokovima podataka. |
| **Skalabilnost** | Skalabilnost je ograničena u usporedbi sa SAN-om. | Vrlo visoka skalabilnost, omogućuje jednostavno povećanje kapaciteta i performansi. |
| **Redundancija (zalihost)** | Može imati redundanciju putem RAID-a, ali ovisi o konkretnom uređaju. | Visok stupanj redundancije i pouzdanosti, često koristi napredne tehnologije za osiguranje dostupnosti podataka. |
| **Sigurnost** | Dobra sigurnost, ali ovisi o mrežnoj sigurnosti i pravilnim postavkama dozvola prava pristupa. | Visoka sigurnost zbog namjenskih mreža i naprednih sigurnosnih mjera. |



**SAN** sustav čini kombinacija specijaliziranog hardvera i softvera; primjerice:

- ***Fibre Channel*** adaptera (tzv. **HBA**) na strani poslužitelja.
- ***Fibre Channel*** preklopnika i mreže te ***Fibre Channel SAN*** sustava za pohranu.

Međutim, **SAN** sustav može koristiti i ***Ethernet*** mrežu i TCP/IP skup protokola bez potrebe za specijaliziranim hardverom, i to samo i isključivo ako se koristi **iSCSI** protokol i (***SAN*** sustav koji ga podržava).

**Za detalje, pogledajte i sljedeća dva poglavlja (*iSCSI* i *Fibre Channel*) !**

**Izvori informacija: (1165),(1167), K-17, `man lsscsi`.**

### 13.12.1.1. iSCSI

Prije nego se upoznamo s ***iSCSI*** protokolom, moramo se malo detaljnije upoznati sa ***SCSI*** standardom.

## ***Small Computer System Interface*** (SCSI)

**SCSI** je skup standarda za fizičko povezivanje i prijenos podataka između računala i perifernih uređaja, najpoznatijih po svojoj upotrebi s uređajima za pohranu kao što su diskovi, iako se pomoću ovog sučelja spajanju i tračni uređaji, skeneri i druga periferija. SCSI standard se mijenjao s vremenom:

- **SCSI-1**:
  - Godina definiranja standarda: 1986.
  - Prva inačica SCSI standarda definirala je osnovni skup naredbi i fizičko sučelje za komunikaciju između računala i perifernih uređaja.
- **SCSI-2**:
  - Godina definiranja standarda: 1990.
  - SCSI-2 je bila nadogradnja na SCSI-1 s poboljšanjima u performansama, dodatnim naredbama i podrškom za šire i brže sabirnice.
- **SCSI-3**:
  - Godina definiranja standarda: 1994.
  - SCSI-3 je značajno proširenje SCSI-2 standarda, uvodeći modularnu arhitekturu s različitim slojevima za transport, fizičko sučelje i naredbe. SCSI-3 primjerice koristi *Fibre Channel Protocol* (**FCP**), koji omogućuje prenošenje SCSI naredbi preko *Fibre Channel* mreže. Isto tako i **iSCSI** prenosi SCSI-3 naredbe u *Ethernet* mrežama preko IP protokola.



Za više detalja o fizičkom **SCSI** sučelju i diskovima, pogledajte poglavlje:
**13.1.1. Podjela prema vrsti diskova.**

## iSCSI

*iSCSI* standard je razvijen početkom 2000-ih. Prva specifikacija ***iSCSI*** protokola definirana je 2003. godine od strane IETF (*Internet Engineering Task Force*) kao **RFC 3720**.

Neke od prednost **iSCSI** su sljedeće:

- ***iSCSI*** koristi standardne *Ethernet* mreže i mrežnu opremu (preklopnici, usmjerivači) i IP protokol, smanjujući potrebu za specijaliziranom *Fibre Channel* infrastrukturom, što može značajno smanjiti troškove.
- S obzirom da ***iSCSI*** može biti implementiran na postojećim mrežama, to olakšava postavljanje i upravljanje SAN-om bez potrebe za dodatnim *Fibre Channel (**FC**)* preklopnicima, FC modulima i kablovima.



Za više detalja o ***Fibre Channel*** tehnologiji pogledajte sljedeće poglavlje:
**13.12.1.2. Fibre Channel (FC).**

**Osnovni koncepti iSCSI**

- **iSCSI inicijator** - klijent (obično poslužitelj) inicira komunikaciju i šalje SCSI naredbe preko IP mreže. Inicijator može biti fizički poslužitelj ili računalo, virtualno računalo ili aplikacija.
- **iSCSI cilj (target)** – sustav za pohranu (npr. *SAN* poslužitelj s poljem diskova) prima i obrađuje SCSI naredbe, pružajući pristup fizičkim ili virtualnim diskovima. To su obično **SAN** (*Storage Area Network*) sustavi za pohranu.



Linux može raditi i kao ***iSCSI inicijator*** i/ili kao ***iSCSI Target.***

**Kako radi iSCSI**

1. **Enkapsulacija podataka –** SCSI aredbe koje generira aplikacija ili operativni sustav *inicijatora* se enkapsuliraju (ugnježđuju) u iSCSI protokole. Ove enkapsulirane naredbe se zatim prenose kao IP paketi preko mreže.
2. **Prijenos preko IP mreže –** *iSCSI* koristi TCP/IP protokol za prijenos podataka.To znači da se iSCSI paketi prenose preko standardne *Ethernet* mreže, omogućujući korištenje postojećih mrežnih infrastrukturnih komponenata (prekolopnici i usmjerivači).
3. **Dekapsulacija na odredištu –** kada *iSCSI* paketi stignu na cilj (*target*), *iSCSI* sloj na cilju dekapsulira SCSI naredbe iz *iSCSI* paketa. Dekapsulirane SCSI naredbe se zatim obrađuju od strane ciljanog uređaja/sustava za pohranu (***SAN***).
4. **Izvršavanje SCSI naredbi –** ciljani (odredišni) uređaj za pohranu (**SAN**) izvršava SCSI naredbe (npr. čitanje ili pisanje podataka na disk) i generira odgovarajuće odgovore. Odgovori se ponovno enkapsuliraju u *iSCSI* pakete i šalju natrag *inicijatoru* preko IP mreže.
5. **Dekapsulacija na inicijatoru** – kada iSCSI paketi s odgovorima stignu natrag na *inicijator*, iSCSI sloj na *inicijatoru* dekapsulira odgovore i šalje ih natrag operativnom sustavu.

**iSCSI Adrese i imena (IQN)**
iSCSI koristi TCP protokol kao transportni protokol, i to obično **TCP** portove **860** i **3260**.
*iSCSI Qualified Name* (**IQN**) je jedinstveno ime koje se koristi za identifikaciju iSCSI ciljeva (***targets***) i inicijatora (***initiators***).
Na kraju konfiguracije ***iSCSI Target*** (***iSCSI SAN***) povezuje određeni ***LUN*** s ***IQN***-om (***IQN→ LUN***).
***IQN*** format definiran je tako da osigura globalnu jedinstvenost.
***IQN*** ime ima sljedeći format: **iqn.yyyy-mm.<domena>:<jedinstveni_identifikator>**, a pri tome:

- **yyyy-mm** - predstavlja datum registracije domene (godina i mjesec).
- **<reverse domena>** je registrirana domena (*obrnutim redoslijedom*) organizacije koja upravlja iSCSI uređajem.
- **<jedinstveni_identifikator>** je jedinstveni identifikator koji razlikuje svaki iSCSI uređaj unutar domene.

**Logical unit number (LUN)**
U SCSI terminologiji, **LU** označava logičku jedinicu, koja je određena jedinstvenim brojem logičke jedinice.
Dok **LUN** predstavlja pojedinačno adresabilni (logički) SCSI uređaj koji je dio fizičkog SCSI uređaja (*targeta*).
**U iSCSI okruženju, LUN-ovi mogu biti diskovi ili segmenti određenog RAID polja odnosno logičke particije RAID polja SAN sustava (Slika 127.2.A.).** U radu s *iSCSI, inicijator* pregovara s ciljem (*target*) za uspostavljanje veze s LUN-om, pomoću njegove **IQN** adrese; a rezultat je iSCSI veza koja emulira vezu na SCSI tvrdi disk. Inicijatori tretiraju iSCSI LUN-ove na isti način kao što bi tretirali SCSI ili IDE tvrdi disk (kao blok uređaj). Na primjer, umjesto montiranja udaljenih direktorija kao što bi se radilo na NAS sustavima; primjerice pomoću **NFS** ili **CIFS** protokola, *iSCSI* sustavi pristupaju diskovima kao blok uređajima koje je potrebno particionirati i formatirati, a na kraju i upravljati datotečnim sustavom.

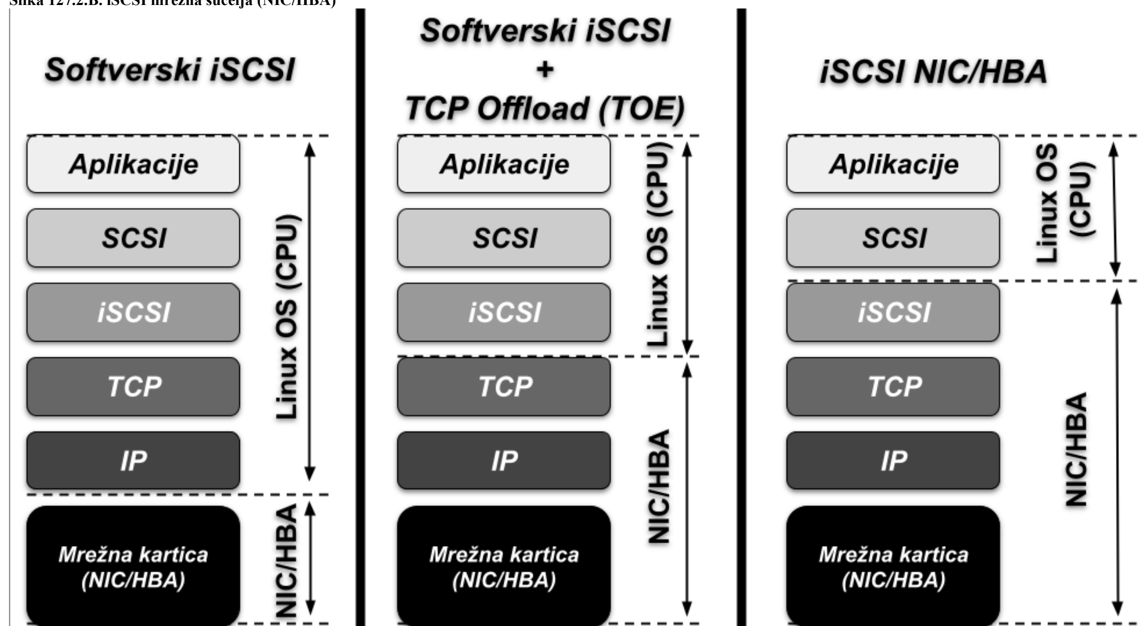
**Proces Povezivanja**

- **Discovery** (otkrivanje) – inicijator koristi iSCSI protokol za otkrivanje iSCSI ciljeva (*targeta*) na mreži. To može biti jednostavna metoda otkrivanja ciljeva na poznatim IP adresama ili korištenje *Internet Storage Name Service* (**iSNS**) za otkrivanje dostupnih ciljeva.
- **Autentifikacija** – inicijatori i ciljevi koriste *Challenge Handshake Authentication Protocol* (**CHAP**) za međusobnu autentifikaciju, osiguravajući sigurno povezivanje (što nije nužno).
- **Session Establishment** (uspostava sesije) – nakon autentifikacije, uspostavlja se sesija između inicijatora i cilja, omogućujući prijenos SCSI naredbi i podataka.
- **Data Transfer** (prijenos podataka) – nakon uspostavljanja sesije, inicijator može slati SCSI naredbe cilju kako bi pročitao ili zapisao podatke.

**iSNS**
iSCSI inicijatori mogu locirati odgovarajuće resurse za pohranu koristeći *Internet Storage Name Service* (**iSNS**) protokol.
iSNS pruža usluge upravljanja slične onima koje se nalaze u *Fibre Channel* mrežama i pripadajućim *FC SAN* sustavima, omogućujući standardnoj IP mreži da radi na gotovo isti način na koji radi *Fibre Channel* mreža. Budući da ***iSNS*** može emulirati usluge *Fibre Channel fabric*-a te upravljati *iSCSI* i *Fibre Channel* uređajima, iSNS poslužitelj može se koristiti kao konsolidirana konfiguracijska točka za cijelu mrežu sustava za pohranu. Međutim, korištenje iSNS-a nije obavezno za iSCSI, dok je potrebno za **iFCP (***Internet Fibre Channel Protocol***)**. **iSCSI** standard ne zahtijeva implementaciju *iSNS*-a za rad.

**iSCSI i mrežne kartice (NIC/HBA)**
*iSCSI* može raditi na više načina. Jedan od njih je samo u softveru: pri tome sve operacije od aplikacija do nižih slojeva TCP/IP stôga odrađuje OS odnosno procesor (CPU). Ovo je naravno najgora moguća varijanta. Druga bolja varijanta rada jest sa snažnijom mrežnom karticom koja može hardverski ubrzati operacije na IP i TCP sloju (tzv. ***TCP Offload***).
Dok je najbrža varijanta rada upotrebom specijaliziranih *Ethernet* kartica koje hardverski ubrzavaju: i IP i TCP, ali i iSCSI operacije, znatno rasterečujući centralni procesor. Takve mrežne kartice (**HBA**) proizode tvrtke: ***Broadcom*** (*NetXtreme II/E/BCM88650/…*), **Mellanox** (*ConnectX-4*), **Intel** (*X720/XL710/…*), ***Chelsio*** (*T580*), …. .

**Slika 127.2.B. iSCSI mrežna sučelja (NIC/HBA)**



## Linux kao iSCSI target

Linux može biti i ***iSCSI target*** odnosno djelovati kao SAN sustav za pohranu (*SAN Storage*).
Naime ***iSCSI target*** je poslužitelj koji dijeli svoje diskovne resurse preko iSCSI protokola. Upravljanje iSCSI target-om se obično vrši pomoću alata koji su specifični za konfiguraciju i administraciju poslužitelja.
Neki od uobičajenih skupova alata za *iSCSI target* u Linuxu su:

- **LIO** (*Linux-IO*): *LIO* je standardno već ugrađeni *iSCSI target* u modernim Linux kernelima. Koristi se za konfiguraciju i upravljanje iSCSI target-ima putem alata kao što je `targetcli`. *Targetcli* alat omogućava administraciju LIO iSCSI target-a, uključujući kreiranje sustava za pohranu, dodavanje LUN-ova, i konfiguraciju ACL-ova.
- **SCST** (*SCSI Target Subsystem for Linux*): *SCST* je moćan i fleksibilan sustav za implementaciju *SCSI* targeta, uključujući *iSCSI* targete. On omogućava izlaganje različitih vrsta uređaja za pohranu preko mreže koristeći različite SCSI protokole (npr. *iSCSI* ili *Fibre Channel*). *SCST* se sastoji od jezgre koja je implementirana kao kernel modul, a koja pruža osnovne funkcionalnosti i razne popratne module koji omogućavaju povezivanje sa stvarnim uređajima za pohranu odnosno skladišnim uređajima. Pri tome iSCSI frontend modul omogućava SCST-u da funkcionira kao iSCSI target.

I **LIO** i **SCST** su implementacije unutar kernel prostora Linuxa, što znači da imaju izravan pristup hardveru sustava.
To opet znači da je pristup uređajima učinkovit te osigurava nisku latenciju. A to rezultira boljom izvedbom od implementacija u korisničkom prostoru Linuxa koje zahtijevaju prebacivanje konteksta i sistemske pozive koji unose kašnjenje u rad sustava.



Linux **LIO** (***Linux-IO***), znan i kao ***Linux Open-iSCSI*** je najčešće u primjeni kao ***iSCSI target***.

**Trenutno nećemo objašnjavati konfiguraciju Linuxa kao iSCSI Targeta!**

## Linux kao iSCSI inicijator

Linux može biti ***iSCSI initiator***, a *ISCSI inicijator* radi tako da formira sesiju za povezivanje s *iSCSI odredištem (targetom)*.
Prema zadanim postavkama, iSCSI servis se pokreće nakon pokretanja naredbe `iscsiadm`. Ako *root* disk nije na iSCSI uređaju ili nema definicije `node.startup = automatic` tada se *iSCSI* servis neće pokrenuti sve dok se ne izvrši naredba `iscsiadm`, a koja zahtijeva pokretanje *iscsid* servisa ili kernel modula `iscsi`.



Moguća je i složenije upotreba višestrukih *iSCSI* odredišta (tzv. ***Multipath I/O***). Za više detalja pogledajte:
**13.11.3.1. Device Mapper Multipath (Multipath I/O).**

Da bi slijedeći koraci radili, pretpostavimo da već imamo postavljen *iSCSI Target* (***SAN sustav***)!
***U našem slučaju je iSCSI Target (SAN Storage) na IP adresi: 192.168.1.250.***

Prvo je potrebno instalirati programski paket s iSCSI initiator komponentama:
```
yum -y install iscsi-initiator-utils
```

Sada je potrebno inicijalno pokrenuti servis:
```
systemctl start iscsid
```

I možemo ga trajno aktivirati (za automatsko pokretanje):
Sada je potrebno inicijalno pokrenuti servis:
```
systemctl enable iscsid
```

Prilikom prvog pokretanja servis kreira datoteku: `/etc/iscsi/initiatorname.iscsi` u kojoj se nalazi konfiguracijski parametar `InitiatorName`, koji se koristi za definiranje jedinstvenog naziva *iSCSI inicijatora* na Linux sustavu, primjerice:
```
InitiatorName=iqn.1994-05.com.redhat:537b712444a0
```
Što znači ova vrijednost:

- U ovom primjeru, `iqn.1994-05` označava format **IQN**-a, koji uključuje datum (svibanj 1994.) kada je **IQN** format definiran, a `com.redhat` je obrnuto napisano ime domene vlasnika inicijatora (*Red Hat*).
- Oznaka `537b712444a0` je jedinstveni identifikator za iSCSI inicijatora.

Ove oznake prema želji moženo i promijeniti.

Sada provjerimo pristup našem *SAN Storage* sustavu na kojem se nalazi *iSCSI Target*:
```
iscsiadm -m discovery -t st -p 192.168.1.250
192.168.1.250:3260,1 iqn.2005-10.org.freenas.ctl:iscsi.share.1
```
← Vidimo da smo dobili odgovor:

- Na IP adresi `192.168.1.250`, na portu `3260` nalazi se *iSCSI target* imena/identifikatora:
  - `iqn.2005-10.org.freenas.ctl:iscsi.share.1`.

Sada se pokušajmo spojiti na ovaj *iSCSI Target*:

```
iscsiadm -m node -T iqn.2005-10.org.freenas.ctl:iscsi.share.1 --login

Logging in to [iface: default, target:
iqn.2005-10.org.freenas.ctl:iscsi.share.1, portal: 192.168.1.250,3260]

Login to [iface: default, target:
iqn.2005-10.org.freenas.ctl:iscsi.share.1, portal: 192.168.1.250,3260] successful.
```

← Navedeni *iSCSI target* nema postavljena prava pristupa niti autentikaciju (što je isto moguće postaviti)!

Istu poruku možemo vidjeti i pomoću log poruka servisa:

```
systemctl status iscsid
. . .
iscsid[4553]: iscsid: Could not set session1 priority. READ/WRITE throughout and
latency could be affected.
iscsid[4553]: iscsid: Connection1:0 to [target: iqn.2005-
10.org.freenas.ctl:iscsi.share.1, portal: 192.168.1.250,3260] through [iface …]
```

I konačno novi iSCSI kao blok uređaj (disk) vidjet ćemo ovako (skratili smo ispis):

```
lsscsi
[4:0:0:0]    disk    TrueNAS  iSCSI Disk       0123  /dev/sdc
```

← Vidimo da je novi iSCSI disk vidljiv kao `/dev/sdc`.

Pogledajmo ga kao blok uređaj:

```
lsblk /dev/sdc

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
sdc    8:32   0   4G  0 disk
```

Za iSCSI detalje možemo pokrenuti naredbu `iscsiadm` (ispis smo znatno skratili):

```
iscsiadm -m session -P 3

iSCSI Transport Class version 2.0-870
version 6.2.1.9
Target: iqn.2005-10.org.freenas.ctl:iscsi.share.1 (non-flash)
        Current Portal: 192.168.1.250:3260,1
        Persistent Portal: 192.168.1.250:3260,1
                **********
                Interface:
                **********
                Iface Name: default
                Iface Transport: tcp
                Iface Initiatorname: iqn.1994-05.com.redhat:537b712444a0
                Iface IPaddress: 192.168.1.125
                *********
                Host Number: 4  State: running
                scsi4 Channel 00 Id 0 Lun: 0
                        Attached scsi disk sdc          State: running
                . . .
```

***Sada ovaj disk trebamo: 1. particionirati, 2. formatirati te 3. montirati, kao da se radi o lokalno spojenom disku.***



Za više detalja o particioniranju i formatiranju te montiranju diskova i particija pogledajte:
**13.6. Rad s particijama.**
**13.6.2. Formatiranje particije.**
**13.7. Kako se montiraju particije u direktorije i zašto.**

Za trajno montiranje potrebno je prilikom spajanja na *iSCSI target* učiniti sljedeće (naredba u jednom retku):

```
iscsiadm -m node -T iqn.2005-10.org.freenas.ctl:iscsi.share.1 --op update
-n node.startup -v automatic
```

Nakon restarta računala možemo provjeriti je li se ponovno spojio na iSCSI target:

```
iscsiadm -m node
192.168.1.250:3260,1 iqn.2005-10.org.freenas.ctl:iscsi.share.1
```

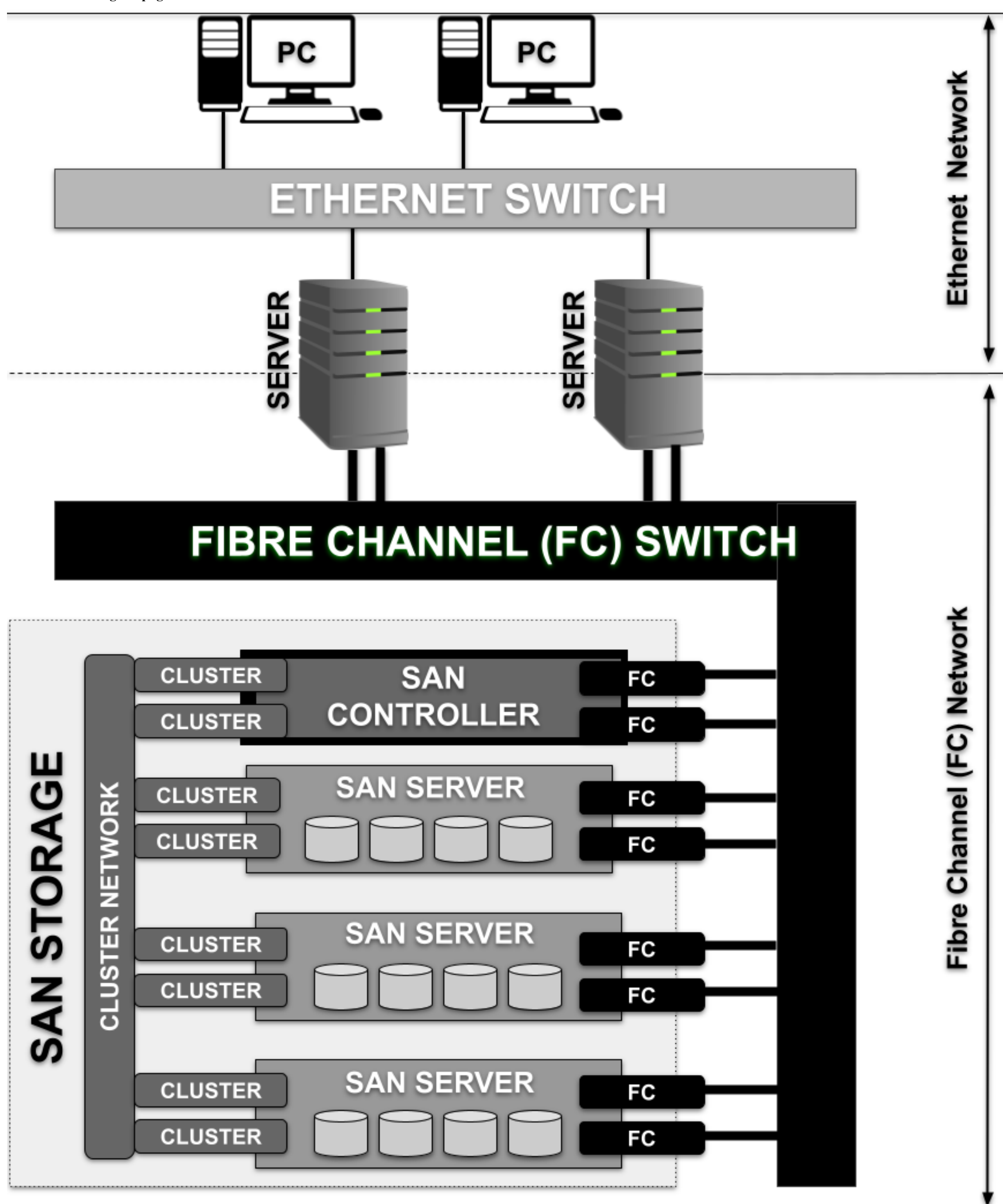
← U ovom slučaju vidimo da jest.

**Izvori informacija: (380),(994),(1169),(1715),(1716),(1717),(1718),(1719), K-17, RFC 3720, `man iscsid`,`man iscsiadm`.**

## 13.12.1.2. Fibre Channel (FC)

***Fibre Channel*** (**FC**) je specijalizirani protokol za prijenos podataka velikom brzinom koji omogućuje urednu isporuku neobrađenih blokova podataka bez gubitaka. *Fibre Channel* se prvenstveno koristi za povezivanje sustava za pohranu podataka u mrežama za pohranu podataka (***SAN***), najčešće u većim sustavima ili u podatkovnim centrima.
*Fibre Channel* mreže rade tako da tvore jednu veliku mrežnu strukturu odnosno ***SAN*** sustav u kojem preklopnici (*Switchevi*) u mreži rade kao jedan veliki preklopnik. *Fibre Channel* obično koristi optička vlakna unutar i između podatkovnih centara, ali također može raditi i upotrebom bakrenih kabela. Podržane brzine prijenosa podataka uključuju 1, 2, 4, 8, 16, 32, 64, 128 i 256 gigabita u sekundi (Gbps) kao rezultat poboljšanja u uzastopnim generacijama ove tehnologije. Industrija trenutne brzine označava kao *Gigabit Fibre Channel* (**GFC**). *Fibre Channel* standardiziran je u **T11** tehničkom odboru Međunarodnog odbora za standarde informacijskih tehnologija (**INCITS**), akreditiranom odboru za standarde Američkog nacionalnog instituta za standarde (**ANSI**). *Fibre Channel* je dizajniran kao sučelje za prevladavanje ograničenja **SCSI** sučelja fizičkog sloja.
Dakle *Fibre Channel* preko posebne *Fibre Channel* mreže prenosi **SCSI** naredbe sustavu za pohranu (*storage system*), upotrebom specijaliziranih *Fibre channel* preklopnika (*switcheva*).
*Fibre channel* preklopnici **NISU** klasični ***Ethernet*** preklopnici te ne koriste TCP/IP skup protokola niti su bazirani na OSI modelu mreže. Pogledajte pojednostavljenu shemu primjene *Fibre Channel* ***SAN*** sustava, kako je vidljivo na slici 127.2.C.

**Slika 127.2.C. Logički pogled na Fibre Channel SAN sustav.**



Na slici s prethodne stranice vidljivo je da:

- Računala (na slici **PC**) preko klasične **Ethernet** mreže, koja koristi TCP/IP skup protokola baziran na OSI mrežnom modelu, komuniciraju s poslužiteljima.
- Poslužitelji (na slici **SERVER**) imaju i klasičnu ***Ethernet*** mrežu preko koje komuniciraju s računalima (**PC**), ali za pristup diskovnim uređajima (diskovima preko ***SAN*** sustava) imaju i zasebna *Fibre Channel* mrežna sučelja odnosno specijalizirane kartice. Ove *Fibre Channel* kartice u poslužiteljima nazivaju se *Host Bus Adapter* (**HBA**), a preko njih se poslužitelji spajaju na *Fibre Channel* preklopnike (na slici **Fibre Channel Switch**) i **FC** mrežu.
- Na kraju, *Fibre Channel* preklopnik je povezan sa ***SAN Storage*** sustavom za pohranu podataka.
- ***SAN Storage*** sustav za pohranu podataka se uobičajeno sastoji od:
    - ***SAN kontrolera*** preko kojeg se upravlja i konfigurira ***SAN*** sustavom.
    - ***SAN poslužitelja*** s diskovima (uobičajeno u nekom *RAID* polju):
        - Koji su s jedne strane međusobno povezani ekstremno brzom specijaliziranom klasterskom mrežom preko.
        - S druge strane su preko *Fibre Channel* mreže spojeni na *Fibre Channel* preklopnik.

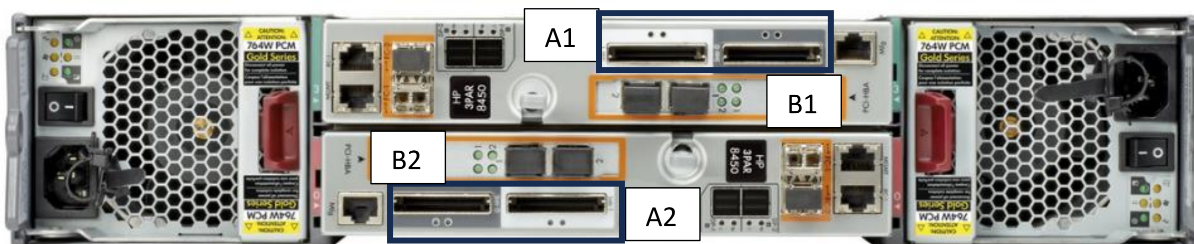
**Logical unit number (LUN)**
I na **FC SAN** sustavima **LUN** predstavlja pojedinačni adresabilni (logički) SCSI uređaj. ***LUN***-ovi mogu biti numerirani diskovni uređaji ili segmenti određenog RAID polja odnosno logičke particije *RAID* polja *SAN* sustava (**slika 127.2.A.** iz prethodnog poglavlja). Zatim **FC SAN** povezuje **LUN** s jedinstvenom **WWID** oznakom, koja se koristi za komunikaciju.

Na slici dolje vidimo *SAN* poslužitelj tvrtke **HP: 3PAR 8200** s prednje strane (s diskovima) te sa stražnje strane.

**Slika 127.2.D. SAN Poslužitelj tvrhe HP – 3PAR – 8200 (2 node) – prednnja strana**



**Slika 127.2.E. SAN Poslužitelj tvrhe HP – 3PAR – 8200 (2 node) – stražnja strana**




Sa stražnje strane se vide klasterska sučelja (**A1** i **A2**) te *Fibre Channel* sučelja: **B1 (2xFC)** i **B2 (2xFC)**.



Za jedan od primjera upotrebe ***FC SAN*** sustava i ovakvih poslužitelja pogledajte poglavlje: **13.11.3.1. Device Mapper Multipath (Multipath I/O).**

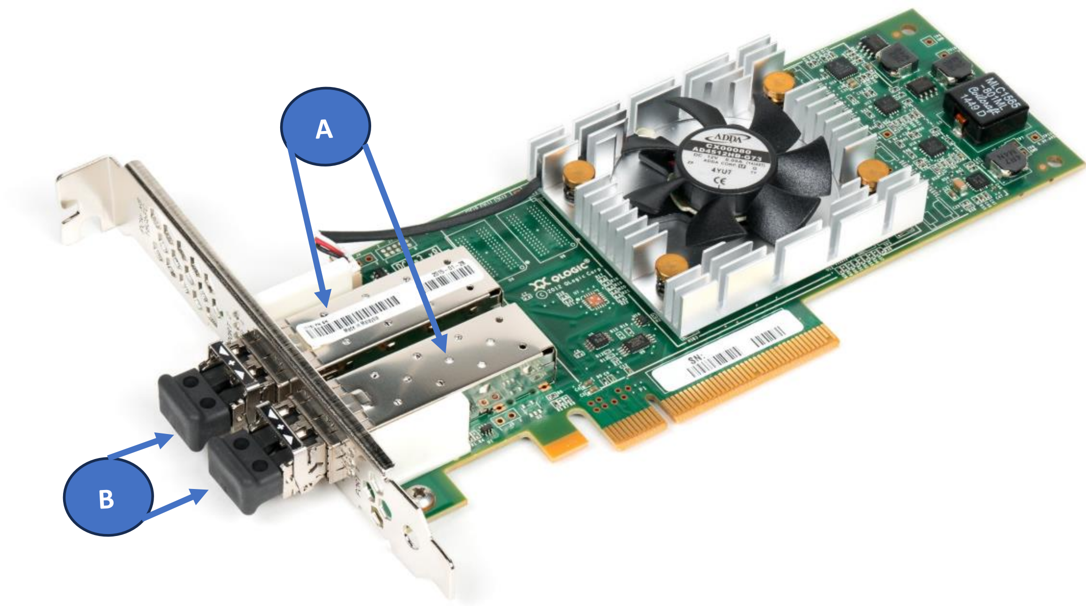
## ***Host Bus Adapter*** (HBA)

*Host Bus Adapter* (**HBA)** se odnosi na specijaliziranu karticu koja povezuje sabirnicu računala s uređajima za pohranu (***SAN***). Iako se *Host Bus Adapter* (**HBA)** može odnositi i na ***Ethernet*** mrežne kartice/adaptere ili ***Infiniband*** kartice, ovaj izraz se prvenstveno koristi za označavanje uređaja za povezivanje: ***SCSI***, ***SAS***, ***NVMe***, ***SATA*** uređaja (tzv. **DAS**), ali i ***Fibre Channel***. Odnosno možemo ga nazvati i specijaliziranim diskovnim kontrolerom ili karticom za povezivanje s uređajima za pohranu, bilo da se radi o direktnoj vezi na uređaj za pohranu (pr. *SCSI, SAS, NVMe* ili *SATA*) ili u slučaju najčešće primjene *Fibre Channel* tehnologije, na diskovni ***SAN*** sustav.

**Direct Attached Storage (DAS)** se odnosi na (diskovne) uređaje koji se direktno povezuju na računalo ili poslužitelj bez posredovanja bilo koje mreže. *DAS* se obično sastoji od jednog ili više tvrdih diskova, solid-state diskova (SSD) koji su povezani putem sučelja kao što su: SATA, SCSI, SAS, ili NVMe.

Na slici dolje (127.2.E.) vidljiv je ***Dual port 16 Gb FC host bus adapter*** s dva **SFP+** utora (oznaka **A**) u kojima se nalaze SFP+ *Fibre Channel* optički moduli (oznaka **B**).

**Slika 127.2.E. HBA Fibre Channel kartica s dva SFP+ utora.**




**Autor slike: Dmitry Nosachev, Licenca CC BY-SA 4**



Važno je razumjeti da iako primjerice na ***HBA*** *Fibre Channel* kartici na slici 127.2.D. vidimo da ona ima dva **SFP+** utora: to nisu klasični utori za SFP+ ***Ethernet*** module, iako su fizički kompatibilni.

Naime iako *Fibre Channel* koristi **SFP** *(SFP+, SFP28 ili SFP56 SFP te SFP-DD ili QSFP)* utore, u koje se mogu ugraditi pripadajući moduli, ti moduli moraju biti za *Fibre Channel* protokol.
Jedino što je ovdje zajedničko između ***Ethernet*** i ***Fibre Channnel*** opreme jesu *Single-mode* ili *Multi-mode* optička vlakna i konektori koji se mogu koristiti za povezivanje **SFP** modula.



Za više detalja o optici, **SFP** utorima i modulima pogledajte:
**19.2.1.2. Optika.**
**19.2.2.2. Small Form-factor (SFF) i Small Form-factor Pluggable (SFP).**

Pogledajmo službene nazive i brzine *Fibre channel* mrežnih sučelja:

| Naziv | Propusnost (MB/s) po smjeru | Način kodiranja – kodiranje | T11 standard dostupan od |
|---|---|---|---|
| **1 GFC** *(Gen 1)* | **100 MB/s** | NRZ - 8b/10b | 1997.g. |
| **2 GFC** *(Gen 2)* | **200 MB/s** | NRZ - 8b/10b | 2001.g. |
| **4 GFC** *(Gen 3)* | **400 MB/s** | NRZ - 8b/10b | 2004.g. |
| **8 GFC** *(Gen 4)* | **800 MB/s** | NRZ - 8b/10b | 2008.g. |
| **16 GFC** *(Gen 5)* | **1.600 MB/s** | NRZ - 64b/66b | 2011.g. |
| **32 GFC** *(Gen 6)* | **3.200 MB/s** | NRZ - 256b/257b | 2016.g. |
| **64 GFC** *(Gen 7)* | **6.400 MB/s** | PAM-4 - 256b/257b | 2020.g. |
| **128 GFC** *(Gen 8)* | **12.800 MB/s** | PAM-4 - 256b/257b | 2024.g. |

Postoje različiti protokoli više razine za *Fibre Channel*, uključujući dva za blokovnu pohranu. Među njima je najkorišteniji *Fibre Channel Protocol* (**FCP**). *Fibre Channel Protocol* je protokol koji prenosi **SCSI** naredbe preko *Fibre Channel* mreža.

## Fibre Channel protokol

*Fibre Channel* protokol ne slijedi OSI slojeve, kao primjerice TCP/IP protokol. *Fibre Channel* (**FC**) je protokol za prijenos podataka koji se koristi prvenstveno za povezivanje računala i poslužitelja s uređajima za pohranu podataka u mrežama za pohranu odnosno ***SAN*** mrežama (*Storage Area Networks*). **FC** je dizajniran za pružanje visoke brzine prijenosa podataka, pouzdanosti i niske latencije, što ga čini pogodnim za aplikacije koje zahtijevaju visoke performanse. **FC** koristi tehnike kao što su **CRC** (*Cyclic Redundancy Check*) za provjeru i korekciju pogrešaka, što osigurava visoku pouzdanost prijenosa podataka. Podaci se prenose u obliku okvira (frames). Svaki okvir sadrži zaglavlje koje uključuje kontrolne informacije i korisničke podatke.

**Topologija**
*FC* podržava nekoliko topologija, uključujući ***point-to-point*** (dvije točke povezane direktno), ***arbitrated loop*** (AL - uređaji povezani u petlju) i ***switched fabric*** (SF - uređaji povezani putem FC preklopnika, što je i najčešći način povezivanja vidljiv na slici **127.2.B.**).

**Prijenos podataka**
Podaci se prenose pomoću *Small Computer System Interface* (***SCSI***) naredbi enkapsuliranih unutar **FC** okvira. To omogućava ***FC*** da prenosi podatke između poslužitelja i uređaja za pohranu efikasno i brzo.

**Kontrola protoka**
***FC*** koristi različite mehanizme kontrole protoka, kao što su krediti za prijenos podataka (buffer-to-buffer credits), kako bi osigurao da prijenos podataka teče glatko i bez zastoja.

**Adresiranje**
Svaki uređaj u FC mreži ima jedinstvenu adresu poznatu kao *World Wide Name* (**WWN**). WWN se koristi za identifikaciju uređaja i za usmjeravanje podataka.

## *World Wide Name* (WWN)

*World Wide Name* (**WWN**) jedinstveni je identifikator koji se koristi u tehnologijama pohrane uključujući ***Fibre Channel***, ***Parallel ATA***, ***Serial ATA***, ***SCSI*** i *Serial Attached SCSI* (**SAS**). Kako smo već rekli; **HBA** adapter za *Fibre Channel* omogućuje uređajima u *Fibre Channel* mreži da razmjenjuju podatke – moguće je povezati poslužitelj s **FC** preklopnikom ili uređajem za pohranu (*SAN Storage*), povezati više sustava za pohranu ili povezati više poslužitelja.
Pri tome svaki ***Fibre Channel HBA*** ima jedinstveni *World Wide Name* (WWN), koji je sličan **Ethernet MAC** adresi po tome što koristi **OUI** dodijeljen od strane IEEE. Međutim, *WWN*-ovi su duži (8 bajtova).
Postoje dvije vrste WWN-ova na ***HBA*** odnosno *Fibre Channel* adapteru/kartici:

- **WWN** za čvor (*Node*), koji se označava kao ***WWNN***, koji dijele svi portovi (priključci) na *Fibre Channel* adapteru.
- Port **WWN** koji se naziva **WWPN (***World Wide Port Number***)**, koji je jedinstven za svaki priključak odnosno port.



**WWN** – svaki poslužitelj (ili **HBA**) ima svoj jedinstveni **WWN**, koji se koristi za identifikaciju poslužitelja unutar ***SAN***-a. S druge strane *Fibre Channel* preklopnik kao uređaj ima svoj WWN, koji se koristi za identifikaciju preklopnika unutar *FC* mreže. Nadalje i ***SAN*** sustav za pohranu (*storage array*) ima svoj **WWN.**

**WWPN**
**FC HBA** na poslužitelju može imati jedan ili više fizičkih portova (pr. **SFP+**), i svaki port ima svoj jedinstveni WWPN. WWPN se koristi za identifikaciju specifičnog porta kroz koji poslužitelj komunicira s *FC* mrežom. Svaki port na *FC* preklopniku također ima svoj WWPN. Ovi portovi povezuju različite uređaje unutar SAN-a, a WWPN-ovi portova na *FC* preklopniku se koriste za usmjeravanje podataka i za implementaciju sigurnosnih politika. Kod SAN sustava za pohranu: pojedini portovi na kontrolerima sustava za pohranu također imaju svoje WWPN-ove. Ovi portovi omogućuju komunikaciju s poslužiteljima i drugim uređajima u SAN sustavu.

**WWID**
S druge strane *World Wide Identifier* (**WWID**) je jedinstveni identifikator za **LUN**-ove. Naime i na ***FC SAN*** sustavima za pohranu, željeno RAID polje diskova (kojih može biti i više) segmentiramo odnosno particioniramo na logičke cjeline koje predstavljaju **LUN**-ove. Unutar **FC SAN** sustava se potom LUN-ovi povezuju s jedinstvenom *WWID* oznakom, koja se koristi za pristup do svakog pojedinog **LUN-a** (slika **127.2.A.** iz prethodnog poglavlja).

## Fibre Channel Zone

*Fibre Channel zone* su mehanizmi za segmentiranje mreže za pohranu (*SAN*) u manje logičke dijelove radi poboljšanja sigurnosti, upravljivosti i performansi. Zoning omogućava kontrolu pristupa između uređaja unutar SAN-a, ograničavajući komunikaciju samo na određene uređaje koji su uključeni u istu zonu. Zone definiraju skup uređaja (kao što su poslužitelji, preklopnici i uređaji za pohranu) koji mogu međusobno komunicirati unutar SAN-a. Uređaji unutar iste zone mogu komunicirati jedni s drugima, dok uređaji izvan te zone ne mogu. Postoje dvije vrste zona (i njihova kombinacija):

- **Soft zoning** – Temelje se na **WWN** adresama uređaja. Ovaj pristup je fleksibilniji jer se zone definiraju pomoću jedinstvenih identifikatora uređaja.
- **Hard zoning –** temelji se na fizičkim portovima preklopnika. Uređaji povezani na određene portove preklopnika su uključeni u zonu. Ovaj pristup je stroži i često se koristi za dodatnu sigurnost.



Za više detalja o Zonama i **Multipath I/O (MPIO)** sustavima, pogledajte poglavlje:
**13.11.3.1. Device Mapper Multipath (Multipath I/O).**

**Izvori informacija: (1167),(1168),(1709),(1710),(1711),(1712),(1713),(1714), K-17.**

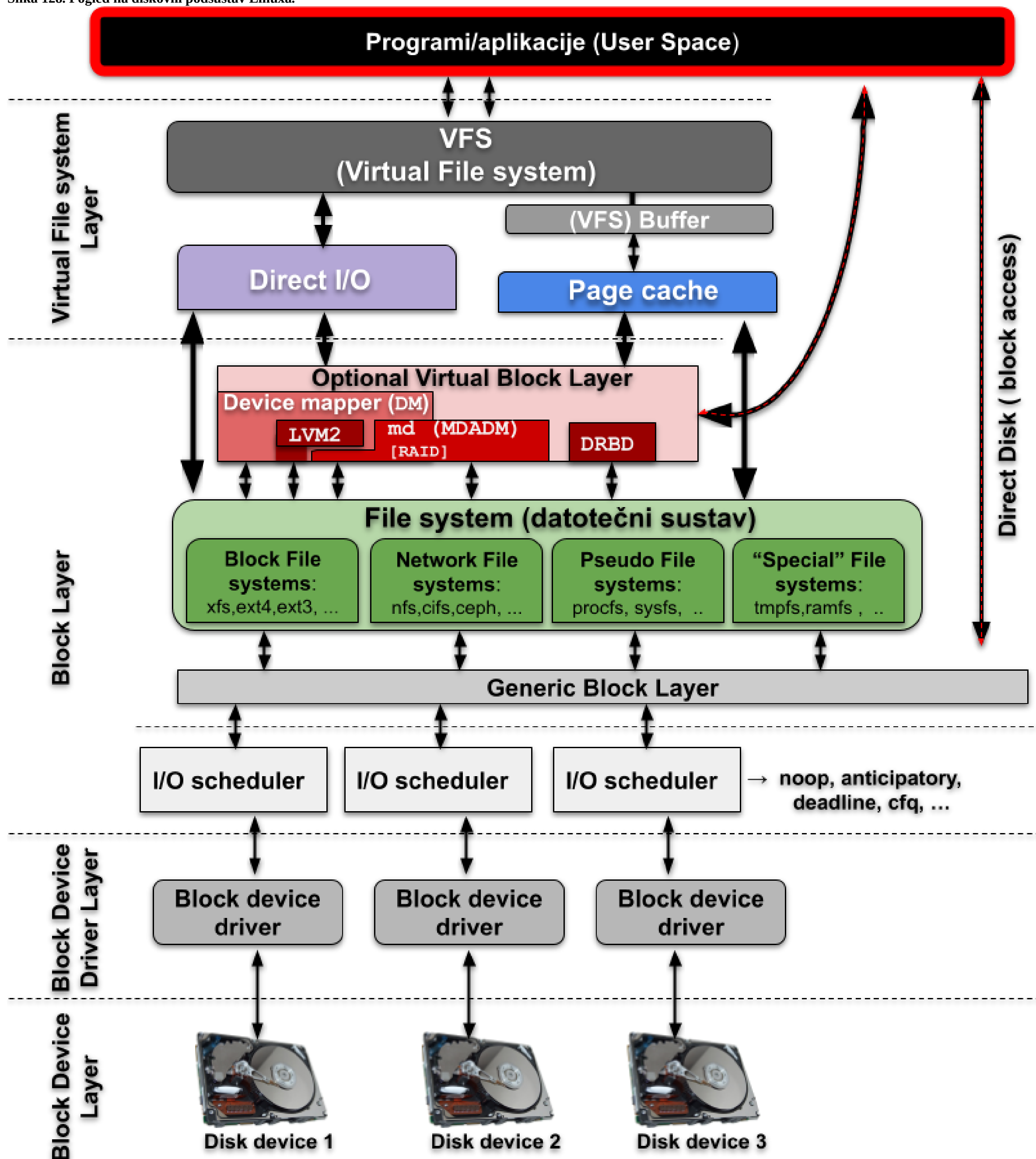
# 14. Diskovni (I/O) podsustav

**Slijedi napredno poglavlje (14.x)!**
U ovom poglavlju upoznat ćemo se s diskovnim podsustavom, odnosno njegovim komponentama i funkcijama. Ovaj sustav je također jedan od većih elemenata svakog operativnog sustava. Dodatno ćemo se upoznati i sa specifičnim tehnologijama te naučiti kako pratiti i analizirati sve pod komponente diskovnog sustava.

## 14.1. Diskovni ulazno/izlazni sustav (*I/O*)

Prije nego krenemo s detaljnim upoznavanjem diskovnog podsustava, prvo pogledajte gdje je on smješten u sâmom kernelu; → poglavlje: **11.1 Kernel**. Sada se možemo detaljnije upoznati s Linuxovim diskovnim ulazno/izlaznim (tzv. **I/O**) sustavom. Što se događa u kojem sloju diskovnog (**I/O**) sustava možemo vidjeti na slici 128. a detalje ćemo vidjeti u tekstu koji slijedi.

**Slika 128. Pogled na diskovni podsustav Linuxa.**




**Programi** odnosno aplikacije koje se, usput rečeno, nalaze u posebnom korisničkom prostoru rada (tzv. *user space*), pristupaju ***VFS*** (Engl. *Virtual Filesystem*) podsustavu kada žele nešto čitati ili zapisivati na disk. Ova (***VFS***) komponenta, prvo sve podatke prosljeđuje prema ***Page cache*** sloju (RAM *međumemorija*) odnosno pokušava iz njega čitati ili zapisivati. Osnovna namjena ***VFS*** podsustava jest prihvaćanje zahtjeva za čitanjem ili zapisivanjem podataka prema nižim slojevima diskovnog podsustava pomoću odgovarajućih sistemskih poziva. ***VFS*** je svjestan činjenice, načina i metoda pristupa ovisnih o tome o kojem se konkretnom datotečnom sustavu u konačnici radi.

***VFS*** sustav u sebi ima implementirane mnoge metode pomoću kojih podržava različite vrste datotečnih sustava odnosno on „zna" kako čitati i zapisivati podatke prema njima. Dodatno on i razlikuje nekoliko vršnih metoda pristupa, poput:

- Disk baziranih datotečnih sustava; koji su obično na lokalnim (ili *USB*) diskovima. Neki od predstavnika ovakvih datotečnih sustava su: neki od *Linux Extended* datotečnih sustava (***ext2***,***ext3***, ***ext4***) ili klasičnih *UNIX* sustava (***UFS***,***VxFS*** i sličnih) kao i *Microsoft* baziranih (***MS-DOS***, ***VFAT*** i ***NTFS***), ali i standardnih **ISO9660** (CD-ROM) ili ***UDF*** (DVD-ROM) formata, te sustava poput: ***IBM* OS/2**: **HPFS**, *Apple Macintosh*: ***HFS*** i drugih.
- Mrežnih datotečnih sustava; poput ***NFS*** sustava, ***CIFS*** sustava i sličnih.
- Posebnih lokalnih pseudo datotečnih sustava koji predstavljaju posebne unose u Linux kernelu; poput */proc* datotečnog sustava.

**Standardni scenarij čitanja je sljedeći:**

- Prvo se provjerava nalaze li se traženi podaci u međuspremniku odnosno međumemoriji (*Page cache*).
- Ako podaci nisu u međuspremniku zahtjev se prosljeđuje na blok sloj (*Block Layer*) i podaci se dohvaćaju s diska.

## Što se događa na *Page cache* sloju?

*Page cache* ubrzava pristup datotekama:

- U slučajevima kada neka aplikacija prvo pročita podatke s diska, oni se zatim spremaju u *Page cache* memoriju. Kada bilo koja druga (ili ova ista) aplikacija treba ponovno dohvatiti iste podatke, oni se već nalaze u ovoj (RAM) međumemoriji pa će se umjesto drastično sporijeg dohvaćanja s diska, oni biti pročitati iz nje.
- U slučajevima kada se podaci trebaju zapisati na disk, oni prvo završe u *page cache* memoriji i drže se u njoj neko vrijeme kako se ne bi nepotrebno radile operacije pristupanja sâmom disku koje su drastično sporije od rada s RAM memorijom odnosno pristupanjem *page cache* memoriji. Tek potom, u određenim intervalima, podaci se šalju dalje prema nižim slojevima i sâmom disku na zapisivanje. Podaci koji se nalaze u procesu zapisivanja na disk, a nalaze se u *page cache* memoriji se označavaju kao *dirty pages*, ako još nisu stvarno i zapisani na površinu diska.
  U međuvremenu, ako je neka aplikacija trebala pročitati ove podatke ona ih može pročitati iz ove međumemorije, bez čekanja na prvo zapisivanje, a potom i čitanje istih podataka s diska. Sadržaj ovih *dirty pages ili* takozvanih *prljavih stranica memorije* se periodički zapisuje prema površini diska. Prema potrebi moguće je narediti operativnom sustavu da ih snimi na površinu diska prioritetno odnosno da se stanje RAM memorije koja se koristi kao *write page cache* snimi na površinu diska. Ovo se može postići i ručno s naredbom `sync` na sljedeći način:

```
sync
```

Navedena naredba zapisuje sve podatke pohranjene u međuspremniku (*write page cache*) na disk. To može uključivati, ali nije ograničeno na: modificirane *superblokove*, modificirane *inode* unose, ali i operacije odgođenog čitanja ili zapisivanja.

Ako želimo vidjeti koliko podataka je označeno kao *dirty pages* to možemo saznati pregledavanjem posebne datoteke: `/proc/meminfo` u koju se stalno osvježavaju informacije o RAM memoriji, što ćemo postići na sljedeći način:

```
cat /proc/meminfo | grep Dirty

Dirty :    10154 kB
```

Dakle vidimo kako je trenutno označeno: **`10154 kB`** odnosno oko **`10MB`** kao „*dirty pages"*.

Ovo stanje se stalno mijenja kako se podaci iz ove *cache* memorije prosljeđuju prema nîžim slojevima; iako se sadržaj u *cache* memoriji i dalje čuva zbog ubrzavanja operacija čitanja ili pisanja.

## Flush mehanizam

Podaci koji su privremeno pohranjeni u *page cache* međumemoriju, a još nisu snimljeni na disk se označavaju kao *dirty*. Međutim, svako malo; obično svakih 30 sekundi, poseban kernel proces (kernel program) pregledava koje su sve stranice (*pages*) u *page cache* memoriji označene kao *dirty* te ih on, ovisno o zadanim pod mehanizmima i njihovim postavkama, zatim prosljeđuje na zapisivanje prema disku. Ovaj poseban kernel proces se od kernela 2.6.32 do 3.10.x zove **`flush`** dok je na novijim Linux kernelima (3.10.+) za to zadužena komponenta unutar procesa **`kworker`**. Naime na novijim kernelima od 3.10 prešlo se na model u kojem nekoliko generičkih kernel procesa može rješavati niz različitih zadataka. Vidjet ćete ih na popisu procesa kao [`kworker/`] to jest kao primjerice [`kworker/u8:0-flush-8:0`].

*Po jedan ovakav* `kworker` *kernel proces (kernel program) se periodički pokreće za svaki fizički disk.*



Za više detalja o postavkama i opcijama vezanim za ***flush*** mehanizam, pogledajte poglavlje:
**14.1.2. Optimizacija *Page Cache* sloja.**

Pogledajmo poslužitelj koji ima dva fizička diska (`/dev/sda` i `/dev/sdb`):

```
ls -al /dev/sda /dev/sdb

brw-rw---T 1 root disk 8, 0  Dec  1 21:21 /dev/sda
brw-rw---T 1 root disk 8, 16 Dec  1 21:21 /dev/sdb
```

Vidimo da prvi disk (`sda`) ima oznaku `8,0` što to su identifikatori diska, dok drugi disk (`sdb`) ima identifikatore: `8,16`.

Sada pronađimo ***flush*** procese s naredbom: `ps -eaf` na našem sustavu (***kernel procesi u nazivu imaju uglate zagrade* `[ ]`**):

```
ps -eaf | grep -i flush
```

Za kernele do 3.10 dobit ćemo nešto poput:

```
root         2703      2  0 Nov28 ?        00:00:00 [flush-8:0]
root         2704      2  0 Nov28 ?        00:00:00 [flush-8:16]
```

To jest za novije kernele od 3.10.x, nešto poput:

```
root         2704      2  0 Nov28 ?        00:00:00 [kworker/u8:0-flush-8:0]
```

Vidimo kako za svaki identifikator diska (`8:0`) i (`8:16`) postoji pokrenut po jedan Linux kernel proces (*thread*) imena `flush` (ili `kworker/XY-flush-`), s čime smo dokazali teoriju. Važno je razumjeti da ovaj proces nije stalno aktivan (pokrenut).



Za više detalja o *dirty pages* pogledajte optimizacije navedene u poglavlju:
**12.6. Dodatne optimizacije sustava virtualne memorije.**
Za više detalja o uređajima, pogledajte poglavlje:
**11.1.2.1 Uređaji (devices) detaljnije**.

Linux veličinu *Page cache* memorije konstantno optimizira, odnosno povećava ili smanjuje, ovisno traže li kernel ili aplikacije više ili manje RAM memorije. Dakle cilj je da *page cache* dio RAM memorije, bude što veći, kako bi se što više toga držalo u njemu te sâmim time ubrzalo sve operacije čitanja ili zapisivanja na fizički disk. Naredbom s kojom inače možemo vidjeti zauzeće RAM memorije, možemo ugrubo vidjeti i koliko se RAM memorije koristi kao *page cache*.
Naredba o kojoj govorimo je naredba `free` u kojoj ćemo koristiti prekidač `-m` kako bismo vidjeli zauzeće izraženo u MB:

```
free -m

                     total       used       free     shared    buffers     cached
Mem:                 96669      23394      73275          0        956       3070
-/+ buffers/cache:              19367      77301
Swap:                89343          0      89343
```

Navedeni poslužitelj na kojem gledamo zauzeće RAM memorije, ukupno ima 96GB RAM memorije, od koje se vidi kako se koristi 19.367 MB (19 GB) kao *cache* memorija. Ovo je samo okvirna vrijednost, za detalje pogledajte pôd poglavlje sustava virtualne memorije u kojem također govorimo o ovoj (*cache* međumemoriji). Pogledajte poglavlje: **12.3.6 VM - Slabs**.

## Drugi scenariji pristupa

Mogući su i drugačiji scenariji pristupa diskovnom podsustavu, koje možemo podjeliti u nekoliko kategorija:

- ***Direct I/O*** je druga metoda pristupa koju koriste primjerice neke baze podataka i drugi specifični programi koji ne žele koristiti međuspremnike (***page cache***) kojima barata operativni sustav već oni imaju svoju specifičnu implementaciju međuspremnika i mehanizama za dohvaćanje i snimanje podataka prema diskovnom sustavu. Dakle korištenjem *Direct I/O*, moguće je direktno doći do datotečnog sustava izbjegavajući *page cache* međumemoriju.
- ***Sync I/O*** odnosno sinkrono zapisivanje je sâmo dodatna metoda pristupa diskovima (kao opcija) pri kojoj se može naložiti nižim slojevima, da se tijekom operacija zapisivanja na disk čeka na potvrdu da su podaci stvarno prošli sve slojeve diskovnog podsustava te su stvarno i zapisani na površinu diska. Tek potom aplikacija koja je zatražila sinkrono zapisivanja, može poslati sljedeći blok podataka na zapisivanje. Samo kod ***SCSI*** i ***SATA*** diskova s podrškom za ***NCQ*** (*Native Command Queuing*) moguće je dodatno razgraničiti dva načina sinkronog zapisivanja. Prvi je slučaj kada disk čim primi podatke u svoju *cache* memoriju, odmah odgovori da su podaci zapisani te aplikacija može nastaviti sa slanjem drugog bloka podataka za zapisivanje. Druga (sporija) metoda je slučaj kada disk tek kada je iz svoje *cache* memorije stvarno i snimio podatke na površinu diska šalje aplikaciji poruku da pošalje sljedeći blok podataka za zapisivanje (ovo je sada standardno ponašanje). Postoji i novija implementacija koja se zove **FUA** (*Force Unit Access*) a koja ima nekoliko opcija u kojima se definira koji dijelovi iz *cache* memorije se moraju hitno zapisati na površinu diska, a koji ne. Ova opcija nije standardno uključena.
- ***Async I/O (AIO)*** odnosno asinkrona metoda pristupa, kod koje se može slati više zahtjeva istovremeno te nastaviti s izvršavanjem programa bez blokiranja (na čekanje odgovora/potvrde). O ovoj metodi ćemo detaljnije govoriti kasnije.

U određenim slučajevima, moguće je i direktno pristupiti na blok sloj diska (*Block Layer*) bez datotečnog sustava.
Ovdje su moguća dva scenarija pristupa:

- **1. Direktan pristup na *Virtual Block sloj (Layer)*.** Ovdje zahtjevi idu na opcionalni logički *block device* sloj (tzv. *Device Mapper*) koji se u pravilu koristi za softverske RAID uređaje (takozvane *volume managere*) poput ***LVM2***, ***DRBD*** ili drugih sličnih sustava. *Volume manageri* odrađuju sve operacije nad logičkim diskovima poput kreiranja (virtualnih) RAID polja i slično. Direktan pristup ovom sloju se događa primjerice kod kreiranja ovakvih polja diskova ili operacija nad njima ili u još posebnijim slučajevima baza podataka koje žele direktno koristiti disk ili particiju koja se nalazi unutar ovakvih logičkih diskova (softverskih RAID polja), kako bi ju formatirali sa svojim posebno optimiziranim datotečnim sustavom.

Ovo je opcionalan sloj koji se koristi, ako koristimo neko od softverskih *RAID* rješenja, poput:

- Linux **LVM2** *volume managera*.
- **DRBD** (*Distributed Replicated Block Device*) i drugih sličnih sustava.

Na ovom opcionalnom sloju, kreiraju se logička *RAID* polja, pa se tako sve mogućnosti logičkog (softverskog *RAID*) polja, ako primjerice koristimo Linux *LVM2*, događaju upravo ovdje. Ako ne koristimo neko logičko *RAID* polje, ovaj sloj se ne koristi..

Na dnu ovog sloja, a unutar ***Device Mapper*** komponente nalazi se još jedan ***disk Flush*** mehanizam (kernel program `kdmflush`) čija namjena je sinkronizacija podataka čije zapisivanje je odgođeno za kasnije. On je sličan `kworker/XY-flush`, ali se nalazi na razini *Device Mapper* sloja. Naime zapisivanje unutar *Device mappera* prema prema nižim slojevima radi prema principu stavljanja odloženih zadataka u red čekanja, koji `kdmflush` redovito provjerava te naknadno izvršava.

Koristimo li ***LVM2*** tada više informacije o tom polju diskova možemo dobiti s naredbom: `dmsetup`. Pogledajmo i osnovni primjer (naredba `dmsetup`) u kojem ćemo izlistati sve *Volume grupe* koje smo kreirali s *LVM2 volume managerom*

```
dmsetup  info

Name:                VolGroup-lv_swap
State:               ACTIVE
Read Ahead:          256
Tables present:      LIVE
Open count:          1
Major, minor:        253, 1
Number of targets: 1
UUID: LVM-FR9uenpE4f0RBorHqsT98N1pVYm1uuR32tcb4YGWTUk9ngqrEvpGfb8i3G1LivgS

Name:                VolGroup-lv_root
State:               ACTIVE
Read Ahead:          256
Tables present:      LIVE
Open count:          1
Major, minor:        253, 0
Number of targets: 1
UUID: LVM-FR9uenpE4f0RBorHqsT98N1pVYm1uuR3YCgjAXHpQs1IB772TvlbSkazIkFRuPO2
```

Za sada su nam vidljive dvije osnovne stvari:

- Prva *volume grupa*: *VolGroup-lv_swap* je aktivna (*State: ACTIVE*) i ima *Major* i *Minor* brojeve: `253`, `1`.
- Druga *volume grupa*: *VolGroup-lv_root* je aktivna (*State: ACTIVE*) i ima *Major* i *Minor* brojeve: `253`, `0`.



Za više detalja o *Linux* **LVM2** sustavu, **Device Mapperu** i **md (RAID)** pogledajte poglavlja:
**13.11.2. Softverski RAID (LVM2).**
**13.11.3. Device Mapper (DM) i md (RAID).**

**2. Direktan pristup na *Generic Block Layer*** događa se kada zahtjevi idu direktno na blok sloj. To se događa tijekom direktnog pristupa sâmim diskovima u trenutku kreiranja particija i slično. Ovaj sloj sve zahtjeve prosljeđuje dalje na ***IO Scheduler*** sloj. U ovom sloju se rade i sve statistike. I ovdje je u stvarno posebnim namjenama, također moguće da određeni program, a najčešće su to visoko optimizirane baze podataka ili drugi posebno optimizirani servisi, koristi ovakav pristup kako bi direktno pristupio diskovima i particijama diska. Primjerice, postoje određene baze podataka koje opcionalno mogu direktno pristupati željenim diskovima, na njima kreirati particije te sâme mogu kreirati svoj specijalizirani datotečni sustav kojemu ponovno žele pristupati direktno bez Linux slojeva koji se koriste za normalne datotečne sustave. Ovakav pristup se naziva i "***raw***" odnosno *sirovi* pristup disku. Nakon ovih slojeva dolazimo do ***IO Schedulera*** koji ima nekoliko algoritama od kojih može sâmo jedan biti aktivan odnosno odabran za dohvaćanje (čitanje) i zapisivanje podataka.

## I/O scheduleri

***I/O scheduler*** ili raspoređivač ulazno-izlaznih operacija, upravlja redoslijedom kojim se ulazno-izlazni (I/O) zahtjevi šalju uređajima za pohranu, poput tvrdih diskova ili SSD-ova. Njegova primarna funkcija je optimizacija performansi diska i smanjenje latencije u pristupu podacima. Ovisno o odabranom *IO scheduler*u podaci se grupiraju na određeni način te stavljaju u određene nîzove, a sve ovisno o tome što se želi postići: varijable su propusnost ili brzina odziva.
Stoga imamo nekoliko opcija u odabiru *disk schedulera (nazvali smo ih upravo onako kako se vide pod Linuxom)*:

- `cfq` - u većini slučajeva (kernel do **v.5.x**) se koristi *Completely Fair Queuing* odnosno *`cfq`* *scheduler* koji daje dobar odnos propusnosti i brzine odziva. `CFQ` smješta sinkrone zahtjeve poslane od svakog procesa u red (*queue*). To se odrađuje za svaki proces. Tada se određuju vremenski okviri za svaki procesni nîz (*process queue*) u kojima se odrađuje jedan po jedan procesni nîz prema disku. Duljina vremena u kojem će se odrađivati pojedini procesni nîz te broj upita unutar tog nîza ovisi o postavljenom prioritetu za taj (svaki) nîz. S druge strane svi *asinkroni* upiti za sve procese se spajaju zajedno u sâmo nekoliko nîzova i to po jedan nîz za jedan prioritet. *`cfq`* *IO Scheduler* je najuniverzalniji te se često koristi kao standardan u mnogim distribucijama linuxa. Osim toga on ima i određene optimizacije za rad s SSD diskovima.
- *`deadline`* *scheduler* ima dobro vrijeme odziva odnosno latenciju, a ima i veću propusnost u odnosu na ***CFQ***. `Deadline` *scheduler* garantira pokretanje procesa obrade zahtjeva unutar određenih vremenskih okvira. To se postiže tako što se svim diskovnim (*I/O)* operacijama koje dolaze do njega postavljaju vremenski okviri unutar kojih se moraju obraditi (engl. *time deadline* ili *expiration time*). Zapravo se podaci stalno popunjavaju u oba dva pâra nîzova: pâr nîzova za sortiranje i par nîzova: jedan za čitanje i jedan za pisanje. Relativno jednostavan mehanizam ubacuje zahtjeve (nîz podataka) u *Sorting* nîzove i u *Deadline* nîzove, a oba pâra imaju nîz za čitanje i nîz za pisanje. U prvi pâr nîzova (*sorting* nîz) se sortiraju podaci prema sektorima, a u drugi prema *deadline* vremenima. Potom *io scheduler* algoritam određuje koji blokovi podataka će se uzimati iz kojeg od pârova nîzova. Dakle prije prelaska na sljedeći zahtjev određuje se koji nîz koristiti. Nîzovi za čitanje (engl. *read queue*) imaju uvijek viši prioritet, jer procesi (na višoj razini OS-a) ulaze u fazu čekanja na operaciju čitanja (engl. *read*), dok se ona ne izvrši. Određena mjerenja pokazuju kako ***`deadline`*** *I/O scheduler* pokazuje bolje rezultate, naročito za operacije čitanja, od ***CFQ*** *I/O scheduler*a za određene višenîtne programe i također za neke baze podataka.
- *`noop`* (`none`) *scheduler* stavlja sve zahtjeve u jednostavan *FIFO* (*First In First Out*) spremnik. Ovaj *scheduler* se koristi kada se zna da računalo neće mijenjati tok podataka koji će se zapisivati na određene sektore prema disku. Dakle kada računalo (OS) nije svjesno kako će se podaci zapravo zapisati na disk, već to prepušta nekoj drugoj komponenti poput hardverskog RAID kontrolera, koji se brine o tome na koji disk i na koju poziciju će zapisivati podatke na diskove u

njegovom RAID polju te za SSD i NVMe diskove. *noop* (none) *scheduler* se preporuča i unutar virtualnih računala jer *hipervizor* (fizičko računalo) ionako koristi svoj najoptimalniji I/O scheduler.

- *anticipatory scheduler* implementira spajanje zahtjeva prema diskovnom *(I/O)* sustavu te optimizira pristup disku, smanjivanjem fizičkih pomicanja glave diska (ova funkcionalnost nema utjecaja kod SSD diskova). On rješava i probleme kada imamo puno zahtjeva za zapisivanjem između kojih imamo nekoliko zahtjeva za čitanjem. Odnosno on rješava problem kada se između većeg broja zahtjeva za zapisivanjem mora čekati na operacije čitanja. Dakle on se nakon svakog zahtjeva za čitanjem ne prebacuje odmah na sljedeći zahtjev koji bi bio zapisivanje, već čeka jedan mali vremenski okvir za drugim zahtjevom za čitanje, pa se tek onda prebacuje na zahtjeve za zapisivanje. Naziv je došao od rada u kojem on očekuje (engl. *anticipating*) *sinkrone* zahtjeve za čitanje na gore opisan način. *Anticipatory scheduler* nije dobar odabir za *Storage* uređaje (***NAS***/***SAN***), ali je ponekada vrlo dobar odabir za neke baze podataka; koje u pravilu imaju više zapisivanja nego čitanja. Ovaj *scheduler* se ne koristi u kernelima 2.6.33 i novijima jer su njegove dobre funkcionalnosti ugrađene kao opcija koju treba optimizirati unutar *cfq schedulera*. Ova opcija unutar *cfq scehdulera* je slice_idle.
- mq-deadline *scheduler* je takozvana *multi-queue* inačica *deadline schedulera* koja koristi više redova za paralelnu obradu I/O zahtjeva. U *mq-deadline*, svaki red ima vremensku granicu (ili "*deadline*") kako bi se spriječilo da neki zahtjevi čekaju predugo, što smanjuje latenciju i pruža predvidljivost u performansama. On ima izbalansiranu obradu I/O zahtjeva, s fokusom na nisku latenciju te bolje upravljanje mješovitim čitanjima i zapisivanjima. Prema tome pogodan je za mješovite I/O operacije, kao što su baze podataka i aplikacije koje zahtijevaju predvidljive performanse. Što se diskova tiče dobar je odabir za mehaničke diskove (HDD) a može se koristiti i za SSD diskove iako je za njih bolji odabir noop. Osim toga on je dobar odabir i za **iSCSI** ili **FC** sustave za pohranu, kao i za **RAID** kontrolere, iako je kod RAID kontrolera općenita preporuka odabira noop algoritma jer se tada optimizacije prepuštaju konkretnom RAID kontroleru.
- kyber *scheduler* je moderan I/O *scheduler* optimiziran za **NVMe** uređaje, fokusiran na jednostavnost i nisku latenciju. *Kyber* ograničava broj zahtjeva koji ulaze u red na temelju brzine uređaja, umjesto tradicionalnog sortiranja zahtjeva prema prioritetima. Dizajniran za visoku paralelizaciju s NVMe i SSD diskovima, omogućujući bolje performanse s niskom latencijom. Nema značajno opterećenje CPU-a i dobro se ponaša s modernim uređajima koji imaju visoke IOPS mogućnosti. Mada je za NVMe diskove ponekad preporuka odabira noop schedulera
- bfq (*Budget Fair Queueing*) *scheduler* je baziran na cfq i fokusira se na pravičnost i pružanje zajamčenih resursa za svaku aplikaciju. Omogućuje postavljanje "***budžeta***" za svaku I/O operaciju na temelju I/O zahtjeva pojedine aplikacije, čime se jamči da nijedna aplikacija neće monopolizirati I/O operacije. Ovakvo „pravednije“ dijeljenje I/O resursa, izvrsno je za sustave s mnogo različitih procesa. BFQ također nudi glatku korisničku interakciju na desktop sustavima jer sprječava da intenzivne I/O operacije uzrokuju zastoj korisničkog sučelja. Preporuča se za desktop sustave i sustave s više procesa koji se natječu za I/O resurse (npr. streaming medija, virtualizacija s mnogim VM-ovima i slično).

## IO Scheduleri: Hipervizori i virtualizirana okruženja

U virtualiziranim okruženjima, hipervizor koristi vlastiti *I/O scheduler* na razini fizičkog računala (poslužitelja), a virtualno računalo svoj. Međutim, ako se na obije razine koriste dva kompleksnija algoritma za I/O zahtjeve: na hipervizoru i u virtualnim računalima, tada može doći do nepotrebnog dupliciranja rada, što dovodi do gubitka performansi.
Stoga za ovakva okruženja pogledajmo tablicu s preporukama o optimalnom odabiru algoritma (*I/O schedulera*).

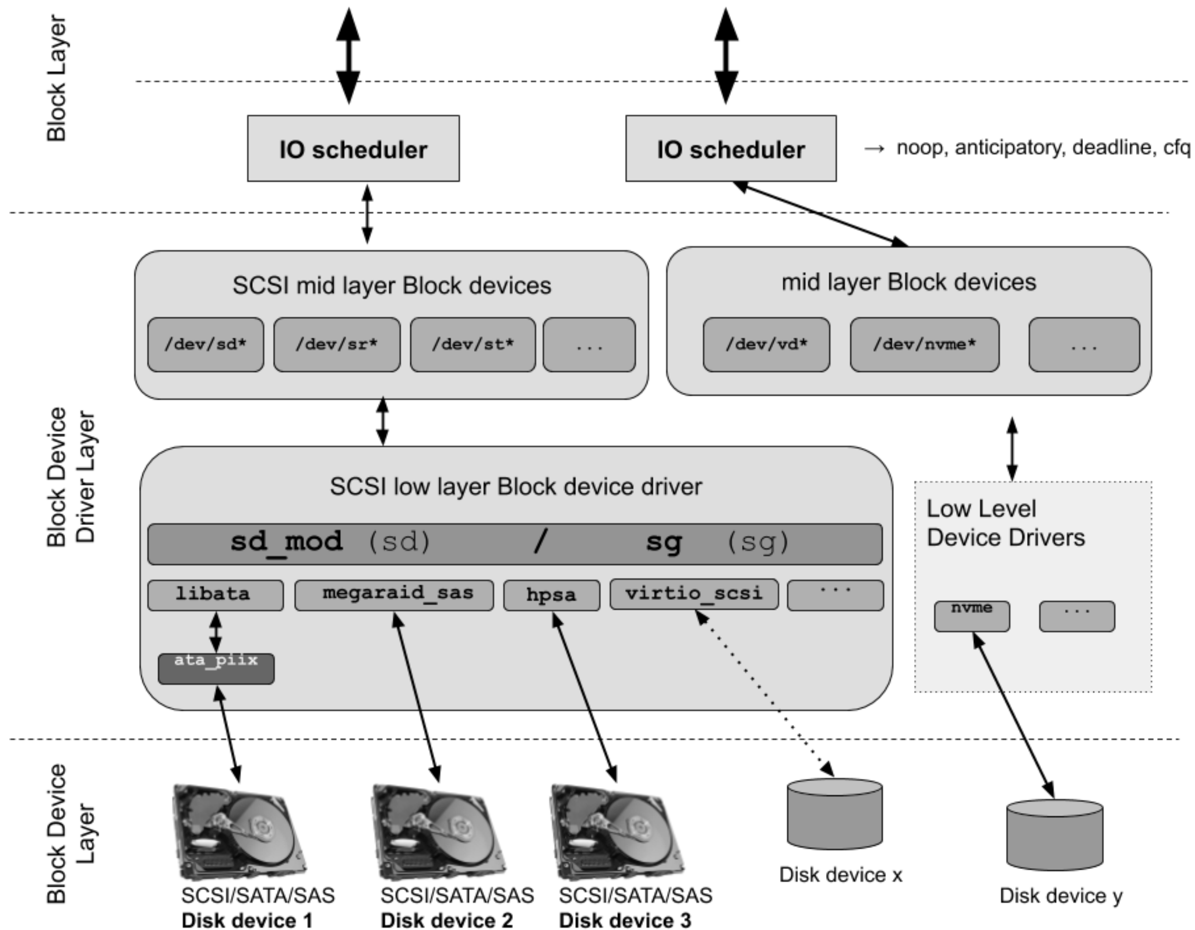
| Vrsta diska | I/O scheduler Hipervizora | I/O scheduler Virtualnog računala | Objašnjenje |
|---|---|---|---|
| **HDD** (*mehanički*) | mq-deadline<br>bfq | mq-deadline | HDD-ovi imaju korist od mq-deadline *schedulera* koji daje prioritet čitanju nad pisanjem, ili BFQ *schedulera* koji osigurava pravednu podjelu resursa među procesima. Unutar VM-a mq-deadline je dobar izbor kako bi se osigurale brze operacije čitanja. |
| **SSD** | noop<br>mq-deadline | noop<br>mq-deadline | SSD-ovi nemaju pokretne dijelove, pa im kompleksni *scheduleri* nisu potrebni. noop smanjuje opterećenje jer kernel izravno prosljeđuje I/O zahtjeve na SSD, osobito na hipervizoru. Ako postoji miješani rad, mq-deadline ovdje može pomoći. |
| **NVMe** | kyber<br>noop | kyber<br>noop | NVMe diskovi su brzi i imaju nisku latenciju, pa je kyber idealan za upravljanje latencijom. noop je prikladan kada latencija nije ključna. Oba su preporučljiva za hipervizore i VM-ove. |
| **RAID kontroler** | noop<br>mq-deadline | mq-deadline<br>noop | RAID kontroleri s *cache* međumemorijom obično već optimiziraju I/O, pa je noop idealan jer smanjuje opterećenje sustava. Dok za balansiranje I/O operacija, mq-deadline može biti dobar odabir. |
| **iSCSI** | mq-deadline<br>noop | mq-deadline | mq-deadline može biti dobar za optimizaciju latencije mrežnog diska i daje prednost čitanju. noop na hipervizoru može smanjiti CPU opterećenje, dok mq-deadline nudi predvidljivije performanse. |
| **FC** (***Fibre Channel***) | mq-deadline<br>noop | mq-deadline | Slično kao i kod iSCSI-ja, za FC je preporuka mq-deadline kako bi optimizirao rad s latencijom i osigurao predvidljivost performansi. Dok noop može smanjiti CPU opterećenje ukoliko hardver optimizira I/O. |

## Blok uređaj i njegov upravljački program

***Block Device Driver*** odnosno upravljački program – na kraju se svi zahtjevi za čitanje ili zapisivanje prosljeđuju upravljačkom programu za disk (*Block device driver)* odnosno kernel modulu, koji u konačnici šalje sve na disk kontroler i na kraju na sâmi disk. Na razini ***Block Device Drivera*** (upravljačkog programa) nalazi se nekoliko slojeva.

Pogledajmo i sliku 129, samo donjeg segmenta diskovnog podsustava. Na ovoj slici je vidljivo što se događa na *blok* sloju komunikacije.

**Slika 129. Pogled na donji segment Linuxovog diskovnog podsustava.**




Ovdje je vidljiva veza između hardverskih uređaja preko posebnih datoteka u `/dev` direktoriju, koje indirektno predstavljaju (disk) uređaje pomoću njihovih upravljačkih programa.

Za ***SATA/SAS/SCSI*** diskove ova poveznica je malo složenija jer se sastoji od dva sloja (gledano od gore):

- Na prvom sloju se nalazi ***SCSI*** srednji sloj, koji je vidljiv kao neki od disk uređaja: primjerice `/dev/sda`.
- Na drugom sloju se nalazi generički sloj `sd` (kernel modul `sd_mod`) i/ili `sg` (kernel modul `sg`) posebnog upravljačkog programa koji se spaja(ju) na konkretan upravljački program za hardver, ispod njega/njih (pr. `ata_piix`), a koji konačno komunicira s disk kontrolerom i sâmim diskom.

Za sve navedene slojeve diskovnog *(I/O)* sustava postoji cijeli nîz opcija i parametara koje je moguće konfigurirati (optimizirati). Važno je dobro proučiti i testirati optimizacije koje ste napravili jer loš odabir parametara ili opcija može usporiti diskovni sustav ili ga učiniti nepouzdanim.

**Izvori informacija: (1010),(1032),(1642),(1760),(1761),(1762),(K-5), `man 5 procfs`,`man sync`, `info coreutils sync`.**

### 14.1.1. Sinkroni i asinkroni *I/O*

U radu s diskovnim sustavom, moguća su dva načina pristupanja podacima kod operacija zapisivanja: ***sinkroni*** i ***asinkroni***.
U **asinkronom** načinu zapisivanja zahtjev za zapisivanjem se normalno šalje diskovnom *(I/O)* sustavu, ali naša aplikacija (proces), čim pošalje zahtjev za zapisivanjem može nastaviti s radom i s novim zahtjevima. Potom kasnije može provjeriti status procesa zapisivanja. Kod ovog načina rada moguće je poslati više paralelnih ulazno/izlaznih *(I/O)* zahtjeva te kasnije provjeriti njihov status. Na ovaj način se diskovnom I/O sustavu daje mogućnost da on paralelizira više ulazno/izlaznih *(I/O)* operacija, kada je to moguće. **Ovo je preporučen način rada**.

U **sinkronom** procesu, koji je druga metoda odnosno opcija zapisivanja (ili čitanja) s diska, zahtjev za zapisivanje se prosljeđuje diskovnom ulazno/izlaznom *(I/O)* sustavu, potom procedura zapisivanja zablokira dalji rad aplikacija sve dok diskovni *(I/O)* sustav ne vrati aplikaciji potvrdu da je zapisivanje prošlo sve slojeve diskovnog podsustava te da je zapisivanje u potpunosti završilo, zapisivanjem na površini diska. Tada aplikacija može nastaviti dalje s novim operacijama zapisivanja.

***Zamislimo dva scenarija u kojima trebamo zapisati četiri bloka podataka na disk:***
**1. Sinkroni** I/O rad: šaljemo zahtjev prema diskovnom I/O sustavu za zapisivanje prvog bloka, potom se čeka potvrda da je prvi blok zapisan. Šalje se zahtjev za zapisivanje na drugi blok, čeka se potvrda da je i on zapisan, …. i tako sve do zadnjeg bloka podataka. Dakle procedura zapisivanja se svodi na to da se šalje zahtjev za zapisivanje ta čeka na potvrdu da je proces zapisivanja završio, a tek potom se može poslati sljedeći zahtjev.
Vrijeme odziva za sva četiri zahtjeva za zapisivanjem je jednaka zbroju svih vremena, od svih odrađenih zahtjeva.

**2. Asinkroni** I/O rad: šaljemo sva četiri bloka na zapisivanje istovremeno (paralelno), potom čekamo potvrdu da su sva četiri bloka zapisana. Pošto je diskovni *(I/O)* sustav zaprimio četiri zahtjeva paralelno, on će ih i pokušati izvršiti paralelno.
Vrijeme odziva je jednako vremenu za pojedini zahtjev koji je najduže trajao. Tada dobivamo sve četiri potvrde istovremeno i nastavljamo s radom.

Međutim, kod obje metode rada u ekstremnim slučajevima, kod vrlo velikih opterećenja diskovnog podsustava, najprije kod sinkronog zapisivanja jer ono pristupa disku što direktnije, a potom (znatno rjeđe) i kod asinkronog, može doći do zagušenja diskovnog podsustava, što je vidljivo kao **`I/O Wait`** stanje (*iowait* statistika). Primjerice s naredbama `iostat` ili `vmstat`:

```
vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id  wa st
 1  0      0 1504932   3128 298876    0    0    34     7   62   76  0  0 100  0  0
```

- *I/O wait* statistika se nalazi u stupcu pod CPU statistikom `--cpu--` i to `wa` stupac.

Za detalje o praćenju performansi diskovnog sustava, vidjeti ćete primjere u sljedećim poglavljima.

**Asinkroni [*Async I/O (AIO)]* i sinkroni pristup detaljnije**
Sinkroni način rada odnosno pristupa diskovnom (I/O) sustavu je prilično jasan te ćemo malo detaljnije pojasniti i asinkroni koji se naziva i ***AIO***. Naime Linux asinkroni I/O (*AIO*) dodan je u Linux kernel 2.6. tijekom 2000-tih. Osnovna ideja koja stoji iza AIO-a je dopustiti programu (procesu) da pokrene određeni broj I/O operacija (čitanja ili zapisivanja) bez potrebe za blokiranjem ili čekanjem da bilo koja od njih završi.
Kasnije, ili nakon obavijesti o završetku I/O operacija, proces (program) može dohvatiti podatke.

Pogledajmo grubu razliku između asinkronog i sinkronog načina pristupa diskovnom podsustavu:

| Način I/O komunikacije | Blokirajući (I/O) pristup | Ne blokirajući (I/O) pristup |
|---|---|---|
| **Sinkroni** | **Read/Write** (čitanje/zapisivanje) | **Read/Write** (čitanje/zapisivanje) [`O_NONBLOCK`] |
| **Asinkroni** | **I/O multipleksiranje** [`select/poll`] | **AIO** |

**Sinkroni blokirajući pristup.**
Jedan od najčešćih modela je model sinkronog rada to jest blokirajući I/O pristup. U ovom modelu program (proces) izvodi sistemski poziv prema diskovnom podsustavu (za čitanje ili zapisivanje) koji rezultira blokiranjem aplikacije. To znači da se aplikacija blokira sve dok se I/O poziv sustava ne završi (prijenos podataka ili pogreška). Program koji je zatražio I/O zahtjev je u stanju u kojem ne troši CPU resurse, ali čeka odgovor od diskovnog sustava, tako da je učinkovit iz perspektive obrade, ali neučinkovit po pitanju dohvaćana podataka s diskovnog sustava. Ova metoda je najčešća metoda pristupa diskovnom sustavu.

**Sinkroni ne blokirajući pristup.**
U ovom načinu rada pristup diskovnom sustavu nije blokirajući. To znači da umjesto dovršetka I/O operacija odmah, primjerice metoda čitanja podataka može vratiti kôd pogreške koji pokazuje da se naredba ne može odmah izvršiti (poruke `EAGAIN` ili `EWOULDBLOCK`), zahtijevajući da aplikacija obavi brojne sistemske pozive kako bi dočekala završetak ove operacije. To može biti krajnje neučinkovito jer u mnogim slučajevima aplikacija mora čekati dok podaci nisu dostupni ili pokušati obaviti neki drugi posao dok se naredba prema diskovnom sustavu izvodi u kernelu. Dakle ovakav rad nije blokirajući, ali je izrazito neefikasan te unosi očito kašnjenje (latenciju).

**Asinkroni blokirajući pristup**
Još jedna paradigma blokiranja je ne blokirajući I/O s blokirajućim obavijestima. U ovom modelu konfigurira se neblokirajući I/O pristup, a zatim se blokirajući sistemski poziv (`select`) koristi za utvrđivanje kada postoji bilo kakva aktivnost za I/O sustav. Primarni problem ovakve metode rada je taj što nije vrlo učinkovita. Iako je to prikladan model za asinkrone obavijesti, ne savjetuje se njegova upotreba za visoke performanse diskovnog pristupa.

**Asinkroni ne blokirajući pristup (AIO)**
I konačno smo došli do potpuno ne blokirajuće metode pristupa diskovnom sustavu. Ova metoda pristupa radi tako da se zahtjev za primjerice čitanje vraća odmah, pokazujući da je čitanje uspješno pokrenuto. Program (aplikacija) zatim može izvršiti neke druge radnje (obradu) dok se operacija čitanja u pozadini ne dovrši. Kada nam stigne odgovor za operaciju čitanja, može se generirati signal ili povratni sistemski poziv temeljen na programskim nîtima, da se dovrši I/O diskovna transakcija. To znači da se asinkroni I/O (*AIO*) u biti odnosi na sposobnost programa (procesa/aplikacije) da izvodi ulazno/izlazne diskove operacije (čitanje/zapisivanje)) uz nastavak normalnog izvršavanja. Umjesto uobičajenog načina rada u kojem je moguće čitati ili zapisivati na diskovni sustav te čekati (faza blokiranja) dok se ove operacije ne završe, program stavlja operacije čitanja ili zapisivanja u red čekanja da bi ih sustav izvršio u nekoj kasnijoj točki. Pri tome sustav obavještava program kada su diskovne I/O operacije (čitanja ili zapisivanja) dovršene.

**Linux AIO API**
U Linuxu postoji nekoliko metoda kako programi (aplikacije) mogu koristiti asinkroni pristup diskovnom sustavu:

- **POSIX AIO** - ovo asinkrono I/O (**AIO**) sučelje omogućuje programima (aplikacijama) da pokrenuti jednu ili više I/O operacija koje se izvode asinkrono (tj. u pozadini). Program može izabrati da bude obaviješten o završetku I/O operacija na različite načine: isporukom signala prekida, instanciranjem programskih nîti ili uopće bez obavijesti. *POSIX AIO* sučelje nudi sljedeće funkcije: `aio_read` (za čitanje), `aio_write` (za zapisivanje), `aio_return` (status izvršenog *AIO* zahtjeva),`aio_error` (status grešaka izvršenog *AIO* zahtjeva), `aio_suspend` (obustava *AIO* zahtjeva dok se jedan ili više *AIO* ne dovrši), `aio_cancel` (otkazivanje *AIO* zahtjeva) i `lio_listio` (stavljanje više *I/O* zahtjeva u red koristeći jedan poziv funkcije). ***POSIX AIO*** se naziva i ***Linux native AIO***. Primjerice u postavkama KVM/QEMU sustava za virtualizaciju, on se definira kao `aio=native`.

- ***IOUring (IO_URING)*** je novi *AIO* model koji je uveden od kernela 5.1.x s novim značajkama, dizajniran za veće performanse. *IO_URING* nudi tri sistemska poziva: `io_uring_setup` (postavljanje konteksta za izvođenje *AIO*), `io_uring_register` (registriranje datoteka ili međuspremnika za *AIO*) i `io_uring_enter` (pokretanje ili dovršavanje *AIO*). Naime *POSIX AIO* je imao određena ograničenja koja je *IO_URING* ispravio:
    - *POSIX AIO* ne podržava međuspremnik za I/O već je podržan samo izravni I/O pristup.
    - Za *POSIX AIO* su bila potreba dva sistemska poziva po jednoj *AIO* operaciji.
    - U konačnici je *POSIX AIO* bio neefikasan (sporiji komunikacijski kanal, način izvršavanja), s višom latencijom i problemima na sustavima s velikim brojem I/O operacija (tzv. IOPS-a).

  Za razliku od Linux AIO (*POSIX AIO*), *io_uring* koristi nezavisne dijeljene memorijske spremnike za podnošenje i dovršenje naredbi umjesto sistemskih poziva. Jedan arhitektonski cilj *io_uringa* je minimiziranje latencije smanjenjem troškova sistemskih poziva. Nadalje ***IO_URING*** može raditi u nekoliko načina rada:
    - *Interrupt driven* – ovo je zadani način rada vođen signalima prekida.
    - *Polled* - izvršava se metoda za čekanje za završetak I/O operacija, za razliku od primanja obavijesti putem asinkronih zahtjeva [IRQ-a] (zahtjeva/signala za prekid). Datotečni sustav i blok uređaj moraju podržavati ovakav način rada kako bi to radilo. On pruža manju latenciju, ali može potrošiti više CPU resursa nego I/O potaknut signalima prekida (*Interrupt driven*) [IRQ].
    - *Kernel pooled* - u ovom načinu rada stvara se programska nît unutar kernela za izvođenje operacija. Ovako konfigurirana instanca *io_uring* omogućuje aplikaciji izdavanje *AIO* bez ikakvog prebacivanja konteksta u kernelu. Na ovaj način program (aplikacija) može zatražiti ili primiti *AIO* bez i jednog sistemskog poziva.

  Primjerice u postavkama KVM/QEMU sustava za virtualizaciju *io_uring* se definira kao `aio=io_uring`.

Što se optimizacije ***AIO*** tiče na razini cijelog sustava dostupne su dvije ***sysctl*** varijable:

| Sysctl varijabla | Zadana vrijednost | Opis |
|---|---|---|
| **`fs.aio-max-nr`** | 65536 | Maksimalan ukupni broj ***AIO*** zahtjeva na cijelom sustavu.<br>Zadana vrijednost 65536 je obično dovoljna za većinu primjena. |
| **`fs.aio-nr`** | ~ | Ukupan broj (brojač) ***AIO*** operacija na sustavu |



Za detalje upotrebe ***AIO*** u virtualizaciji pogledajte poglavlje:
**27.1.2.3. Disk kontroleri**.

**Izvori informacija: (1415),(1419),(1420),(1421),(1422),(1423),(1424),(1425),(K-3),(K-4),(K-8)**,**`man 7 aio`**, **`man vmstat`**,**`man iostat`**.

### 14.1.2. Optimizacija *Page Cache* sloja

Kada se prvi put čita ili zapisuje na diskove poput tvrdih, SSD, NVMe, USB ili drugih vrsta diskova, Linux pohranjuje te podatke u neiskorištena područja RAM memorije, koja se potom koriste kao iznimno brza predmemorija. Naime, ako se ovi podaci ponovno trebaju čitati kasnije, mogu se vrlo brzo pročitati iz (ove) predmemorije. Navedena predmemorija naziva se *Page Cache*. U slučaju zapisivanja podaci se također ovdje zapisuju i čuvaju sve dok ih proces zadužen za zapisivanje na površinu diska (tzv. ***Flush***) ne preuzme i zapiše na disk.



Za detalje pogledajte poglavlja:
**14.1. Diskovni ulazno/izlazni sustav (*I/O*).**
**12.3.3.5. Naredba *vmstat*.**

Linux *page cache* koristi strategiju takozvanog povratnog zapisivanja (engl. *write-back*) za zapisivanje podataka.
Ova strategija znači slijedeće:

- Kada se podaci zapisuju u datoteku, promjene se prvo zapisuju u *Page cache* međumemoriju u RAM-u.
- Ažurirane stranice memorije se tada označavaju kao "prljave" (engl. *dirty pages*).
- Stvarno zapisivanje na disk događa se kasnije.To znači da program (ili proces) koji je pokrenuo zapisivanje može nastaviti s radom bez čekanja da se podaci stvarno i zapišu na disk (tzv. *asinkroni rad*).

I druge strategije zapisivanja podataka na disk su također moguće (ali ne i za **Page cache**).
Ove metode se primjerice mogu koristiti i u virtualizaciji. One se u načelu svode na dvije metode odnosno načina zapisivanja:

- ***Write-through*** - metoda koja primjerice koristi ***Direct-IO*** metodu, bez upotebe *Page cache* međumemorije. Njena osnovna značajka jest pouzdanost zapisivanja jer se ne koristi međumemorija, ali i značajno sporiji rad.
- ***Write-back*** - metoda koja standardno koristi ***Page cache*** međumemoriju. Njena osnovna značajka jest velika brzina rada (*asinkrona* metoda zapisivanja), ali i manja pouzdanost. Primjerice u slučaju nestanka električne energije tijekom procesa zapisivanja će se svi podaci o ovoj (RAM) memoriji izgubiti jer se podaci prvo pohranjuju u RAM memoriji.

U ovoj cjelini objasnit ćemo koje su sve optimizacije moguće na razini *Page Ccahe* sloja. Kako smo vidjeli u uvodnom dijelu priče ovo je prvi sloj između korisnika i diskovnog podsustava, a koji služi kao predmemorija, koja se zove ***Page Cache***. Dodatno, moguće je optimizirati određene parametre *Page Cache* sustava, jer se on sastoji od nekoliko cjelina odnosno mehanizama. Veličina ***page cache*** memorije se automatski podešava unutar granica koje možemo definirati.
Međutim u određenim slučajevima možemo imati potrebu isprazniti *Page Cahe* memoriju, odnosno narediti sustavu da sve što se trenutno nalazi u ovoj memoriji, uredno snimi na disk.

Dodatno važno je znati kako se u *Page Cache* memoriji ne pohranjuju sâmo podaci (datoteke) već i određeni metapodaci, poput:

- *Inode* unosa - ovo su strukture koje predstavljaju datoteke.
- *Dentries*-a odnosno strukture koja predstavlja direktorije, ali i neke strukture **VFS**-a vezane za datotečni sustav i *Inode* unose ili međumemorije; primjerice: `xfs_inode`, `xfs_buff`, `proc_inode_cache`, … .
- I konačno onoga što se zove *pagecache*, koji sadrži konkretne podatke odnosno sadržaj nedavno otvorenih datoteka.

## Rad s *Page Cache* međumemorijom

Moguće je narediti sustavu da kompletno stanje ***Page Cache*** memorije snimi da disk, pomoću sljedeće naredbe:

```
echo 3 > /proc/sys/vm/drop_caches
```

Standardno stanje ove varijable je nula (`0`) a kada mu promijenimo stanje, kao u primjeru gore, ona se potom sâma vraća u stanje nula (normalan rad). Objasnit ćemo i druge vrijednosti koje možemo poslati ovoj varijabli (`/proc/sys/vm/drop_caches`):

- `0` – ovo je standardno stanje.
- `1` – služi za pražnjenje samo *page cache* unosa.
- `2` – služi za pražnjenje samo *inode* i *dentry* unosa.
- `3` – služi za pražnjenje svega: *page cache*, *inode* i *dentry* unosa.



Za detalje oko ispisa stanja i upotrebe međumemorija u Linuxu pogledajte poglavlja:
**12.3.3.5. Naredba *vmstat*.**
**12.3.6. VM - Slabs.**

Sljedeća opcija koju možemo mijenjati je takozvana *cache pressure* opcija.
Navedenu varijablu odnosno funkcionalnost možemo mijenjati na sljedećim mjestima:

| /proc | Sysctl varijabla | Standardna vrijednost | Min-Max vrijednosti |
|---|---|---|---|
| /proc/sys/vm/vfs_cache_pressure | vm.vfs_cache_pressure | 100 | 0-200 |

Kako smo spomenuli, *Page Cache* predmemorija se koristi za spremanje tri vrste podataka: *page cache* unosa (podaci) te *inode* i *dentry* unosa (meta podaci). Ako je vrijednost ove varijable u sredini odnosno **100** (što je standardno) sustav će se brinuti o tome da se predmemorija (*Page Cache*) koristi tako da se podjednaki postotak ove memorije koristi i za *page cache* odnosno podatke, ali i za meta podatke (*inode* i *dentry*).

Ovdje imamo dvije mogućnosti:

- Ako smanjujemo ovu vrijednost, tada veći prioritet u postotku cijele *Page Cache* memorije, imaju meta podaci odnosno *inode* i *dentry* unosi.
- Ako povećavamo ovu vrijednost, tada veći prioritet u postotku cijele *Page Cache* memorije, imaju podaci odnosno *page cache* a ne meta podaci (*inode* i *dentry*).

U ekstremnim slučajevima u kojima se konstantno pristupa velikom broju malih datoteka i direktorija, povećanje ove vrijednosti ima smisla, dok se u slučajevima kada se konstantno pristupa manjem broju velikih datoteka, preporučuje smanjenje ove vrijednosti. Naravno ako niste sigurni, ništa ne dirajte. Sljedeći niz opcija vezan je za podatke koji se nalaze u *Page cache* memoriji.



**Slijede opcije koje se odnose na odgođeno zapisivanje na disk, odnosno na čuvanje podataka u RAM memoriji koja se koristi kao Page Cache.**

Pojam dirty_ratio se odnosi na postotak ukupne RAM memorije koja se može koristiti kao *Page cache*. Ako je ova vrijednost recimo 20%, a imate 10*GB RAM* memorije, to znači kako će se do 2GB *RAM-a* moći koristiti kao *Page Cache*.
Standarda vrijednost je upravo 20(%).



Pogledajte i poglavlje:
**12.6. Dodatne optimizacije sustava virtualne memorije.**

U navedenom poglavlju su vidljivi i drugi aspekti ove opcije. Ova opcija se nalazi vidljiva u:

| /proc | Sysctl varijabla | Standardna vrijednost | Min-Max vrijednosti |
|---|---|---|---|
| /proc/sys/vm/dirty_ratio | vm.dirty_ratio | 20 | 0-100 |

Pojam dirty_background_ratio je usko vezan uz gornju opciju, i ona definira gornju granicu kao granicu prijašnje opcije pri kojoj će, ako se dosegne, pokrenuti proces zapisivanja podataka na disk. Standardna vrijednost ove opcije je 10(%) što znači da ako se dosegne iskorištenje *Page Cache* memorije u kojemu je iskorišteno do 10% ukupne RAM memorije, sustav će pokrenuti program (proces) flush koji će sve unutar ovih granica početi zapisivati iz *page cache* međumemorije, na disk.
Odnosno isprazniti *page cache* međumemoriju kako bi se oslobodila RAM memorija za druge programe i/ili servise.



Opcija **dirty_background_ratio** mora biti manja od (prijašnje) opcije **dirty_ratio**.

Opcija dirty_background_ratio je vidljiva u:

| /proc | Sysctl varijabla | Standardna vrijednost | Min-Max vrijednosti |
|---|---|---|---|
| /proc/sys/vm/dirty_background_ratio | vm.dirty_background_ratio | 10 | 0-100 |



**Sljedeće opcije definiraju vremenske okvire izvan kojih će se pokrenuti procedura pražnjenja *Page Cache* memorije odnosno zapisivanja njenog sadržana na disk.**

Pojam `dirty_expire_centisecs` je opcija u kojoj se definira maksimalno vrijeme u stotinkama sekunde (*centi sekunde*) unutar kojih podaci smiju ostati u *Page Cache* memoriji, prije nego se automatizmom zapišu na disk.
Naime podaci neće beskonačno stajati u *Page Cache* memoriji već će nakon isteka ovog vremena biti zapisani na disk.

Standardna vrijednost je 3000 stotinki sekundi (odnosno 30 sekundi).
Ova opcija se nalazi vidljiva u:

| **/proc** | **Sysctl varijabla** | **Standardna vrijednost (stotinke sekunde)** | **sekundi** |
|---|---|---|---|
| /proc/sys/vm/dirty_expire_centisecs | vm.dirty_expire_centisecs | 3000 | 30 |

Drugo vrijeme, odnosno `dirty_writeback_centisecs` definira vremenski interval, svakih koliko će se kernel proces zadužen za zapisivanje podataka iz *Page cache* memorije na disk (proces `flush`) probuditi i provjeriti status podataka u *Page cache* memoriji.
Ovo vrijeme je također izraženo u stotinkama sekunde, a njegova standardna vrijednost je 500 stotinki sekunde (5 sekundi).

Dakle standardno se svakih 5 sekundi aktivira proces `flush` koji provjerava je li vremenski okvir definiran u varijabli `dirty_expire_centisecs` istekao ili, je li postotak zauzeća RAM memorije za *Page Cache* definiran u varijabli: `dirty_background_ratio` dosegnut.

Ako je zadovoljen jedan od ova dva uvjeta, radi se *flush* odnosno zapisivanje podataka iz *Page cache* memorije na disk. Ovo se radi iz sigurnosnih razloga, odnosno zbog očuvanja konzistencije podataka:
*aplikacija - memorija - disk.*

Vrijednost nula (`0`), isključuje *flush* funkcionalnost globalno, te se **ne preporučuje** jer može uzrokovati probleme u radu sustava.
Ova opcija se nalazi vidljiva u:

| **/proc** | **Sysctl varijabla** | **Standardna vrijednost (centi sekunde)** | **sekundi** |
|---|---|---|---|
| /proc/sys/vm/dirty_writeback_centisecs | vm.dirty_writeback_centisecs | 500 | 5 |

**Izvori informacija: (157),(158),(159),(160),(161),(162), (K-5)**,`man vmstat`.`man 5 procfs`.

### 14.1.3. Optimizacija *Filesystem* sloja

Svaka datoteka koju ćemo spomenuti predstavlja jedan kernel objekt koji je zadužen za određenu opciju ili parametar.
Neki se mogu sâmo pročitati (koriste se za statistike), a drugi se mogu i mijenjati i služe za optimizaciju sustava.
Sve opcije i informacije dostupne za *Filesystem sloj* odnosno sloj datotečnog sustava, nalaze se unutar strukture direktorija `/proc/sys/fs/`.

Pogledajmo sadržaj ovog direktorija:

```
ls -F /proc/sys/fs/

aio-max-nr      dir-notify-enable        fsync-enable          inotify/
nfs/            overflowuid              relatime_interval     ve-xattr-policy
aio-nr          epoll/                   fuse-ve-odirect       lease-break-time
nr_open         pramcache/               suid_dumpable         xfs/
binfmt_misc/    file-max                 inode-nr              leases-enable
odirect_enable  pramcache_ploop_nosync   ve-area-access-check  dentry-state
file-nr         inode-state              mqueue/               overflowgid
quota/          ve-mount-nr
```

Objasnit ćemo nekoliko opcija i informacija dostupnih na ovom sloju (u primjeru su lokacije u ***/proc*** ali i ***sysctl*** varijable):

| ***Opcija*** *(sysctl varijabla)* | **Opis** |
|---|---|
| `aio-max-nr` (**`fs.aio-max-nr=`**`XY`) | Ovo je maksimalan broj dozvoljenih paralelnih diskovnih (I/O) operacija.<br>Uglavnom je postavljen 65.536. Odnosi se na asinkrone I/O zahtjeve.<br>Za neke namjene (pr. neke baze podataka) ovaj broj je potrebno povećati. |
| `aio-nr` (**`fs.aio-nr=`**`XY`) | Informacija o tome koliko je trenutno aktivnih asinkronih I/O operacija. |
| `dentry-state` (**`fs.dentry-state=`**`XY`) | Daje nam informacije o *cache* memoriji za direktorije. Sastoji se od 6 podataka u nîzu. Prvi broj označava ukupan broj unosa direktorija koji su u *cache* memoriji. Drugi broj pokazuje broj neiskorištenih unosa u *cache* memoriji. Treći broj govori koliko sekundi je prošlo između zadnjeg unosa koji je oslobođen i kada je zauzet novi unos. Četvrti govori o trenutnom broju unosa koji je sustav zatražio. Zadnja tri broja se trenutno ne koriste. |
| `/quota/allocated_dquots` | Prikazuje broj alociranih disk kvota (*quota*) odnosno ograničenja.<br>Može ih biti više te se zbog toga prebrojava njihov broj. |
| `/quota/free_dquots` | Broj slobodnih disk kvota. |

| ***Opcija*** *(sysctl varijabla)* | ***Opis*** |
|---|---|
| `file-max` (**`fs.file-max=`**`XY`) | Označava maksimalan broj ***file deskriptora*** dostupnih za cijeli operativni sustav.<br>U slučajevima kada naš sustav otvara veliki broj datoteka, potrebno je povećati ovaj broj. Poruke tipa "*Running out of file handles*" ukazuju na ovaj problem. Za detalje pročitajte poglavlje o file deskriptorima **(`4.5.6.1`)** jer postoji dodatno ograničenje na sloju iznad ovoga.<br>Naredba `ulimit -n` će pokazati koliko je *file deskriptora* dostupno za programe. |
| `file-nr` (**`fs.file-nr=`**`XY`) | Prikazuje broj *file deskriptora* u tri broja. Prvi broj je broj trenutno upotrjebljenih, drugi je broj alociranih, ali neiskorištenih i treći broj je ukupan broj dostupnih. |
| `lease-break-time` (**`fs.lease-break-time=`**`XY`) i `leases-enable` (**`fs.leases-enable=`**`XX`) `(0/1)` | Kada bilo koji program otvori datoteku za zapisivanje, na nju se postavlja posebna oznaka, kako drugi programi u isto vrijeme ne bi mogao pisati u nju. Ako neki drugi program pokuša pisati u nju, sustav na to upozorava proces (program) koji drži otvorenu datoteku, da ju oslobodi; na način da snimi sve što je trebao i oslobodi ju unutar vremena definiranog ovdje (45 sekundi). Ako program koji drži datoteku to ne učini niti nakon ovog vremena, kernel će osloboditi datoteku za novo zapisivanje (**`fs.leases-enable=1`**). |
| `nr_open` (**`fs.nr_open=`**`XY`) | Ovdje se definira maksimalan broj *file deskriptora* koje bilo koji proces može alocirati. |

Za više informacija možete pogledati: https://www.kernel.org/doc/Documentation/sysctl/fs.txt

**Izvori informacija: (157), `man 5 procfs`.**

### 14.1.4. Optimizacija *Generic Block Layera* i I/O *Schedulera*

Definicija parametara *scheduler*-a se nalazi definirana u datotekama unutar direktorija `/sys/block/*/`. Svaka datoteka predstavlja jedan kernel objekt koji je zadužen za određenu opciju ili parametar. Opcije ili parametri su specifični za pojedini disk. Dakle datoteke unutar direktorija `/sys/block/sda/` su zadužene za disk `/dev/sda` odnosno `/sys/block/sdb/` za disk `/dev/sdb` i tako dalje. Pogledajmo što je sve moguće optimizirati na *Generic Block Device* sloju za recimo disk `/dev/sdb`. Stoga pogledajmo sadržaj direktorija: `/sys/block/sdb/`

```
ls -F /sys/block/sdb/

alignment_offset    capability  device@    ext_range
inflight            queue/      removable  sdb1/              slaves/    subsystem@
uevent              bdi@        dev        discard_alignment  holders/   power/
range               ro          size       stat               trace/     ve_device_add
```



Vidimo kako ovdje postoje i dodatni direktoriji koji predstavljaju logičku pôd cjelinu. Unutar tih direktorija se nalaze datoteke u kojima su opcije specifične za te logičke cjeline. Tako je recimo unutar pod direktorija `queue`; cijele putanje: `/sys/block/sdb/queue/` vidljiv nîz opcija (datoteka). Ovdje se nalaze opcije koje spadaju pod ***General Block I/O*** sloj:

```
ls -F /sys/block/sdb/queue

add_random           discard_zeroes_data  iostats             max_sectors_kb
minimum_io_size      optimal_io_size      rotational          unpriv_sgio
discard_granularity  hw_sector_size       logical_block_size  max_segments
nomerges             physical_block_size  rq_affinity         discard_max_bytes
iosched/             max_hw_sectors_kb    max_segment_size    nr_requests
read_ahead_kb        scheduler
```

Objasnit ćemo nekoliko osnovnih opcija odnosno parametara na ovom sloju:

| /sys *datoteka/opcija* | **Značenje/opis** |
|---|---|
| `rq_affinity` | Ova je opcija takozvani *I/O CPU affinity*. Njena vrijednost može biti: (**1**), što znači da kada je *I/O* zahtjev prema ovom disku procesirala jedna CPU jezgra i proslijedila ostatak operacija prema sâmom disku, kada disk završi ono što je trebao (čitanje ili zapisivanje) na nastavak *I/O* operacija, može mu odgovoriti CPU jezgra iz iste grupe, a najčešće ista koja je inicirala i početak *I/O* operacija. Ako je njena vrijednost postavljena na (**2**), to znači kako na nastavak operacija prema istom disku uvijek MORA odraditi apsolutno ista CPU jezgra koja je započela *I/O* operacije. |
| `max_hw_sectors_kb` | Maksimalna veličina pojedinog (svakog) I/O zahtjeva (*block size*) u kilobajtima. |
| `nr_requests` | Koliko I/O zahtjeva ja moguće alocirati na blok sloju za operacije čitanja ili zapisivanja. Ukupna stvarna veličina je dvostruko veća od odabrane zbog „*read+write*" zahtjeva. |
| `optimal_io_size` | Ako nije postavljeno na (`0`) sam "*storage*" uređaj može postaviti svoju preporučenu vrijednost. |
| `read_ahead_kb` | Definirano kao veličina memorije u kilobajtima koju će kernel alocirati kao predmemoriju kod sekvencijalnog čitanja s diska. 128KB je standardna vrijednost. Ako se koristi softverski RAID poput ***LVM2*** ili ako imate potrebe za čitanjem većih količina sekvencijalnih (podataka u nîzu) podataka tada se povećanje ove predmemorije može isplatiti. |
| `rotational` | Ako je vrijednost (**1**) to znači da je uređaj standardni tvrdi disk ("rotacijski" uređaj), ako je vrijednost (**0**) tada se vjerojatno radi o ***SSD*** disku. Ova opcija se koristi za provjeru vrste disk uređaja. |
| `scheduler` | Ovdje se zapisuje koji je točno diskovni (I/O) *scheduler* u upotrebi na konkretnom disku. |

Više podataka o ostalim opcijama možete vidjeti na sljedećim poveznicama:

- Za *sysctl* varijable: https://www.kernel.org/doc/Documentation/sysctl/fs.txt.
- Za ostale opcije: https://www.kernel.org/doc/Documentation/block/.

Pogledajmo za disk `/dev/sdb` koji disk (I/O) *scheduler* mu je u upotrebi odnosno koji se od njih koristi:

```
cat /sys/block/sda/queue/scheduler

noop anticipatory deadline [cfq]
```

Vidimo kako se trenutno koristi ***`cfq`*** *scheduler*.

Unutar direktorija `/sys/block/sdb/queue/iosched/` nalaze se datoteke (opcije) za optimizaciju samog disk *(I/O) schedulera*. Datoteke koje će se pojaviti unutar ovog poddirektorija ovise o tome koji smo ***IO scheduler*** odabrali.
Pogledajmo što je dostupno, ako je u našem slučaju u upotrebi **`deadline`** *scheduler*:

```
ls -F /sys/block/sdb/queue/iosched/

fifo_batch  front_merges  read_expire  write_expire  writes_starved
```

### Pogledajmo osnovne opcije specifične za neke od *I/O schedulera*:

**CFQ** *scheduler*

| */sys datoteka/opcija* | **Značenje/opis** |
|---|---|
| `fifo_expire_async` | Broj milisekundi koje asinkroni diskovni (I/O) zahtjev može/smije ostati neposlužen (neobrađen). |
| `fifo_expire_sync` | Broj milisekundi koje sinkroni diskovni (I/O) zahtjev može/smije ostati neposlužen (neobrađen). |
| `low_latency` | **0** - isključeno: Latencija se ignorira, svaki proces za I/O ima puni vremenski okvir za obradu.<br>**1** - uključeno: Prioritet je pravila raspodjela zahtjeva, ispred propusnosti.<br>Forsira se maksimalno "*wait time*" vrijeme od 300 milisekundi za svaki I/O proces. |
| `quantum` | Broj I/O zahtjeva poslanih na uređaj (disk) u jedinici vremena (odjednom). Ograničava se "*queue depth*". Može uzrokovati probleme s "običnim" diskovima ali poboljšati performanse u slučaju upotrebe RAID kontrolera s većom količinom *cache* RAM memorije. |
| `slice_idle` | Vremenski okvir u milisekundama unutar kojih ***CFQ*** smije čekati na sljedeći I/O zahtjev, unutar jednog niza (*queue*) prije nego smije preći na obradu na drugi nîz (*queue*). |
| ... | … |

Više detalja je dostupno na poveznici: https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt

**DEADLINE** *scheduler*

| */sys datoteka/opcija* | **Značenje/opis** |
|---|---|
| `fifo_batch` | Broj diskovnih (I/O) operacija koje se mogu izvršiti u jednom koraku (engl. *batch*). Niže vrijednosti mogu smanjiti latenciju (kašnjenje). Više vrijednosti mogu povećati propusnost ali uz povećanje latencije. |
| `read_expire` | Vremenski okvir u milisekundama unutar kojeg se svaka operacija čitanja mora odraditi<br>(standardno je postavljeno 500 ms.). |
| `write_expire` | Vremenski okvir u milisekundama unutar kojeg se svaka operacije zapisivanja mora odraditi<br>(standardno je postavljeno 5000 ms.). |
| `writes_starved` | Broj koraka čitanja (*read batch*) koji se može procesirati prije nego dođu na red koraci za zapisivanje<br>(standardno su dvije operacije čitanja).<br>Povećavanjem se daje još veći prioritet čitanju u odnosu na zapisivanje. |
| ... | … |

Više detalja je dostupno na poveznici: https://www.kernel.org/doc/Documentation/block/deadline-iosched.txt

**MQ-DEADLINE** *scheduler*

| */sys datoteka/opcija* | **Značenje/opis** |
|---|---|
| `fifo_expire_async` | Vrijeme (u milisekundama) koje ne-sinkronizirani zahtjevi (npr. upisi u pozadini) mogu čekati prije izvršenja. Zadana vrijednost je obično 250 ms. |
| `fifo_expire_sync` | Vrijeme (u milisekundama) koje sinkronizirani zahtjevi (npr. čitanja) mogu čekati prije izvršenja. Zadana vrijednost je 75 ms. |
| `front_merges` | Određuje hoće li *scheduler* pokušati spojiti novi zahtjev s postojećim na početku reda.<br>Vrijednosti su 0 (onemogućeno) i 1 (omogućeno). |
| `read_expire` | Vrijeme (u milisekundama) koliko dugo zahtjev za čitanje može ostati u redu prije nego što dobije prioritet. Ovo je slična opcija kao **fifo_expire_sync**. |
| `write_expire` | Vrijeme (u milisekundama) koliko dugo zahtjev za pisanje može ostati u redu prije nego što dobije prioritet, slično **fifo_expire_async.** |



Za promjenu disk *schedulera* pogledajte poglavlje:
**14.1.5. Diskovni I/O sustav: primjeri.**



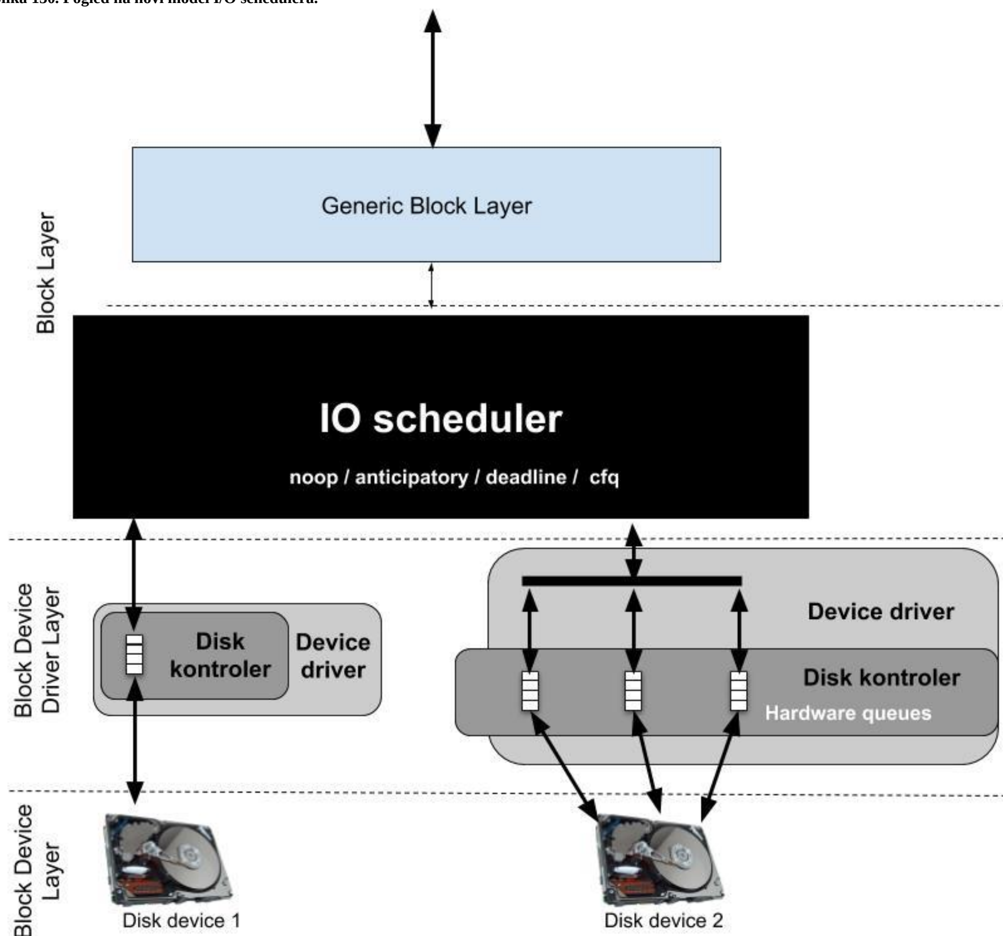
***U nekim slučajevima, određene kombinacije disk (I/O) schedulera i NCQ mehanizma, mogu usporiti diskovni podsustav!***

**Izvori informacija:** `man 5 procfs`,`man 5 sysfs`.

## 14.1.4.1. Nove tehnologije na donjim slojevima I/O podsustava

Tradicionalni linuxov model diskovnog podsustava, iako je konfigurabilan, za vrlo velike, te sustave koji traže ekstremne brzine, ne skalira se baš najbolje. Međutim noviji disk i **RAID** kontroleri uveli su veći broj hardverskih međumemorija za istovremeno primanje i slanje podataka prema diskovima koji ublažavaju ili otklanjaju ovaj problem.
Pogledajmo ovakvu shemu na slici **130**, gledajući samo donji dio diskovnog podsustava.

**Slika 130. Pogled na novi model I/O schedulera.**




Na slici **130** su vidljiva dva scenarija:

- Prvi (lijevi) je klasičan model u kojem disk ili **RAID** kontroler ima samo jednu međumemoriju (*I/O queue*) u koju i iz koje se podaci šalju ili primaju s diskova.
- Drugi model je noviji (desno) u kojemu pojedini disk ili **RAID** kontroler može imati više ovakvih međumemorija, a dodatno, svaka od njih se prijavljuje sustavu, na razini hardverskih signala prekida (*hardware IRQ/Interrupts*) kao zaseban disk kontroler, koji se veže za pojedinu jezgru procesora (CPU). Tako se obrada ulazno/izlaznih (I/O) zahtjeva prema diskovima, preraspodjeljuje između više jezgri procesora i sâmim time ubrzava.

**Optimizacija hardvera**

Istu optimizaciju koju ćemo kasnije odraditi i u poglavlju: **25.2. Dodatne mogućnosti linuxa,** a iste promjene je moguće napraviti i s disk i/ili RAID kontrolerima koji mogu imati više *MSI-X* komunikacijskih kanala, poput RAID kontrolera tvrtke ***HP***: *HP Smart Array P220i* koji koristi `hpsa` kernel modul (upravljački program). Stoga prvo pogledajmo kako se svaki I/O nîz (*queue*) predstavlja sustavu, kao zaseban uređaj, koji je vidljiv u sljedećem ispisu.
Ispis je skraćen jer imamo vidljivo sve od `hpsa0-msix0` do `hpsa0-msix31` odnosno konkretno 32 `hpsa` uređaja:

```
cat /proc/interrupts

IRQ     CPU0        CPU1        CPU2      CPU3      CPU4      CPU5      CPU6      CPU7
--------------------------------------------------------------------------------------------------------
27:   18219       15343       21261     29166     22955     37240     28930     33548   IR-PCI-MSI-edge  hpsa0-msix0
28:  455438        1833     2593994   2204443    253526    286025         0     60654   IR-PCI-MSI-edge  hpsa0-msix1
29: 1255690    33608702         490    422996         6    332616   1489774    178173   IR-PCI-MSI-edge  hpsa0-msix2
30:33950485           0           0         0         0       372         0         0   IR-PCI-MSI-edge  hpsa0-msix3
31:       3           0           0         0         0         0     21599         0   IR-PCI-MSI-edge  hpsa0-msix4
```



Za optimizaciju signala prekida (*IRQ*), proučite sve što smo napravili za **IRQ** i mrežne kartice u poglavlju:
**25.2. Dodatne mogućnosti linuxa.**

Sada pronađimo o kojem hardveru se zapravo radi, pomoću naredbe `lspci` na sljedeći način:

```
lspci | grep -i raid

03:00.0 RAID bus controller: Hewlett-Packard Company Smart Array Gen8 Controllers
```

Dakle **PCI** identifikator ovog RAID kontrolera je: `03:00.0`, a sada pogledajmo njegove detalje pomoću naredbe `lspci`:

```
lspci -s 03:00.0 -v

03:00.0 RAID bus controller: Hewlett-Packard Company Smart Array Gen8 Controllers
(rev 01)
        Subsystem: Hewlett-Packard Company P220i
        Flags: bus master, fast devsel, latency 0, IRQ 26, NUMA node 0
        Memory at f7f00000 (64-bit, non-prefetchable) [size=1M]
        Memory at f7ef0000 (64-bit, non-prefetchable) [size=1K]
        I/O ports at 4000 [size=256]
        [virtual] Expansion ROM at f7e00000 [disabled] [size=512K]
        Capabilities: [80] Power Management version 3
        Capabilities: [90] MSI: Enable- Count=1/32 Maskable- 64bit+
        Capabilities: [b0] MSI-X: Enable+ Count=64 Masked-
        Capabilities: [c0] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [300] #19
        Kernel driver in use: hpsa
        Kernel modules: hpsa
```

Vidimo kako je upravljački program (kernel modul) `hpsa` te da se koriste (`Enable+`) signali prekida MSI-X te da fizički **RAID** kontroler koristi hardverski signal prekida (***IRQ***) broj `26`, koji za razliku od mrežnih kartica koje koriste hardverske međumemorije (*TX/RX queue*) **NEĆE** biti vidljiv u datoteci `/proc/interrupts` jer ona radi na malo drugačiji način.



*Vezano za **hpsa** kernel modul: kod njega konkretno se opcije vezane za I/O queue ne podešavaju, već se to odrađuje na drugoj razini.*

Za neke od promjena, a ovisno o **RAID** kontroleru ponekada (i u ovom slučaju) je potrebno instalirati poseban program od strane proizvođača s kojim se mijenjaju navedene opcije. U konkretnom slučaju je to alat imena: `ssacli`, pa ga instalirajmo:

```
rpm -ivh ssacli-3.10-3.0.x86_64.rpm
```

Ovaj program možete skinuti s web stranica proizvođača. Sada provjerimo kako je na RAID kontroleru postavljena vrijednost *I/O queue-a* korištenjem programa `ssacli`, na sljedeći način:

```
ssacli ctrl all show detail |grep -i que

   Queue Depth: Automatic
```

Vidimo kako je postavljeno na automatski, što znači da se može kontrolirati broj I/O *kanala* koji će se koristiti, a što u konkretnom slučaju znači kako će ih biti kreiran maksimalno veliki broj koji je jednak broju CPU jezgri koje imamo na našem računalu (poslužitelju). U našem slučaju je to 32. Maksimalan broj ovisi o hardveru (disk kontroleru).
Ako to želimo promijeniti, to možemo napraviti i permanentno (trajno) pomoću programa: `ssacli`.

Kod drugih disk/**RAID** kontrolera poput onih tvrtke ***QLogic*** koji koristi kernel modul: `qla2xxx` sve je dostupno direktno iz Linuxa, dakle sve ovisi o uređaju (*RAID*). Konkretno kod RAID kontrolera tvrtke *QLogic* se trenutni broj *I/O queue-a* nalazi u datoteci: `/sys/module/qla2xxx/parameters/ql2xmaxqdepth`. Pogledajmo koliko ih on ima postavljenih:

```
cat /sys/module/qla2xxx/parameters/ql2xmaxqdepth
32
```

Dakle vidimo vrijednost 32, što naravno direktno i možemo mijenjati, promjenom ove vrijednosti. Kod ovog disk kontrolera ova opcija(dolje) je uredno dokumentirana i dostupna direktno iz ovog konkretnog kernel modula. Pogledajmo kako ju vidimo:

```
modinfo qla2xxx | grep ql2xmaxqdepth

parm:    ql2xmaxqdepth:Maximum queue depth to set for each LUN. Default is 32. (int)
```

Pošto je ova funkcionalnost dostupa kao normalna opcija kernel modula, stoga ju je i moguće mijenjati na **normalan** način dodavanjem opcije kernel modulu. To bi u slučaju kada želimo smanjiti broj *I/O queue-a* na `16` bilo kreiranje datoteke imena: `/etc/modprobe.d/qla2xxx.conf` koja bi trebala sadržavati sâmo sljedeći redak odnosno liniju:

```
options qla2xxx ql2xmaxqdepth=16
```

**Sada pogledajmo postavke dužine softverskog *I/O queue-a*/*niza*,** ali ne i broja hardverskih I/O nîzova (*I/O queue*)
Vezano za dužinu odnosno veličinu softverskih *I/O nîzova*, oni se podešavaju na razini SCSI/SATA/SAS uređaja.
Pri čemu je identifikator našeg konkretnog uređaja, broj: `0:0:0:0`, koji ćemo i koristiti kasnije.

**Kako uopće znamo koji je to uređaj?**
Provjerimo koje sve disk kontrolere imamo na sustavu (u našem slučaju je to RAID kontroler), stoga pogledajmo njegov ***PCI identifikator*** (`03:00.0`) na sabirnici:

```
lspci | grep -i RAID

03:00.0 RAID bus controller: Hewlett-Packard Company Smart Array Gen8 Controllers
```

Vidimo kako je njegov identifikator: `03:00.0`, a sada izlistajmo datoteke u direktoriju:`/sys/bus/scsi/devices/` koje počinju s brojevima:

```
ls -l /sys/bus/scsi/devices/

lrwxrwxrwx 1 root root 0 Sep 28 08:50 0:0:0:0 ->
../../../devices/pci0000:00/0000:00:02.2/0000:03:00.0/host0/port-0:1/end_device-
0:1/target0:0:0/0:0:0:0
lrwxrwxrwx 1 root root 0 Sep 28 08:50 0:1:0:0 ->
../../../devices/pci0000:00/0000:00:02.2/0000:03:00.0/host0/target0:1:0/0:1:0:0
```

Na ispisu je vidljiva poveznica; jer se broj `03:00.0` nalazi u linku imena datoteke `0:0:0:0` koja predstavlja sâm disk/RAID kontroler, dok drugi broj predstavlja sâmi disk odnosno u ovom slučaju RAID polje koje predstavlja disk.
Prema tome trebamo ući u direktorij:

```
cd /sys/bus/scsi/devices/0\:0\:0\:0/
```

Sada pogledajmo sadržaj datoteke: `queue_depth`

```
cat queue_depth
1024
```

Dakle vidimo kako je upisana vrijednost *1024* što znači kako se unutar softverskog *I/O* nîza (*queue)* na razini disk kontrolera, a isto toliko ih se obično koristi i za disk (`/dev/sda`). To je stoga jer se ova vrijednost obično nasljeđuje od disk kontrolera i može pohraniti do 1024 *SCSI* naredbe unutar softverskog *I/O* nîza *(queue)*.

Naime ovaj broj ne označava broj softverskih *I/O* nîzova već veličinu koliko *SCSI* naredbi može stati u jedan softverski *I/O* nîz. Naime ovo je broj takozvanih SCSI naredbi, a koriste se i za ***SAS*** i ***SATA*** diskove, koliko ih se odjednom može spremiti u ovu međumemoriju.
Te naredbe će se potom na najnižoj razini poslati na disk kontroler, a konačnici će biti odrađene na samom disku.

**Odakle znamo što je oznaka za cijeli disk/RAID kontroler a što za pojedini disk?**
Pokrenimo naredbu `lsscsi` ovako:

```
lsscsi
[0:0:0:0]    storage HP       P220i            6.34  -
[0:1:0:0]    disk    HP       LOGICAL VOLUME   6.34  /dev/sda
```

Poveznica je sada jasnija:

- `0:0:0:0` → ona pokazuje na ***HP P220i*** RAID/disk kontroler.
- `0:1:0:0` → ona pokazuje na *LOGICAL VOLUME* → odnosno pokazuje na `/dev/sda` (hard disk).

Ovi brojevi (pr.: `0:0:0:0`) u SCSI terminologiji. a vrijede i za ***SAS*** i ***SATA*** diskove, označavaju slijedeće; redom: od lijevo na desno:

- Prvi broj (`0`) označava „*host*".
- Drugi broj (`0`) označava kanal (*channel*).
- Treći broj (`0`) označava uređaj (*device*).
- I zadnji četvrti broj (`0`) označava logički uređaj (*logical unit*).

Odnosno sve se zapisuje u formatu: ***host:channel:device:logical unit***

Ako pokrenemo sljedeću naredbu vidjeti ćemo i vrijednosti ovih softverskih *I/O nizova* direktno:

```
lsscsi -l


[0:0:0:0]    storage HP  P220i            6.34  -
  state=running queue_depth=1024 scsi_level=6 type=12 device_blocked=0 timeout=0
[0:1:0:0]    disk    HP  LOGICAL VOLUME   6.34  /dev/sda
  state=running queue_depth=1024 scsi_level=6 type=0 device_blocked=0 timeout=30
```

Odnosno, ako želimo vidjeti i na koji se najniži sloj za ***SCSI*** (***SAS/SATA***) spaja, odnosno koji od njih koristi naš disk kontroler, to možemo vidjeti s naredbom `lsscsi` na sljedeći način:

```
lsscsi -g

[0:0:0:0]    storage HP       P220i            6.34  -           /dev/sg0
[0:1:0:0]    disk    HP       LOGICAL VOLUME   6.34  /dev/sda    /dev/sg1
```

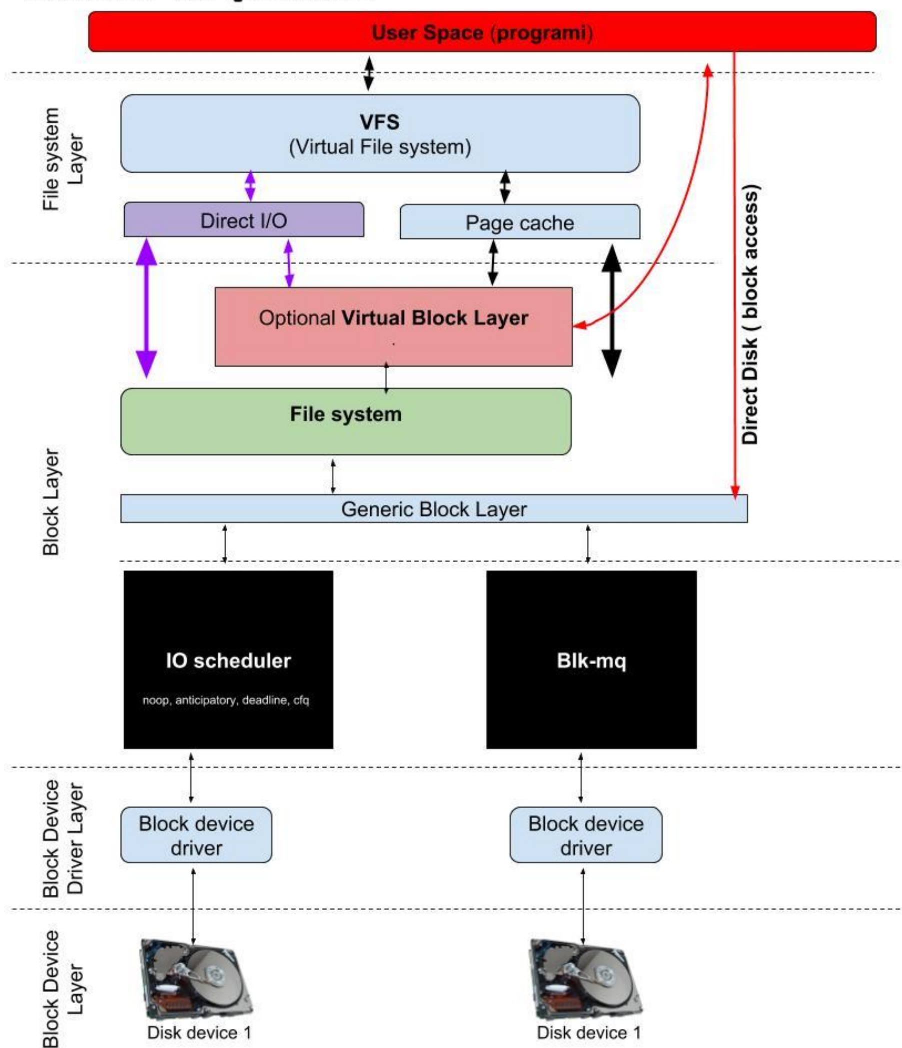
Vidimo kako koristimo upravljački program `sg` na ovom sloju dakle koristimo ***SG*** upravljački program.

**Izvori informacija: (375),(376),(377),(378),(379),(380),(381),(382),(383),(384), `man lsscsi`, `man lspci`, `man ssacli`,`man 5 proc`,`man 5 sysfs`.**

## 14.1.4.2. Nove tehnologije (nastavak)

Ipak, prema mnogima analizama(pr. 374) i modifikacije koje smo spomenuli u prethodnoj cjelini, mogu nam dati do maksimalno 500.000 IOPS operacija prema logičkim uređajima (LUN) odnosno oko 1.000.000 IOPS prema disk kontroleru (**HBA**). Pogotovo kod upotrebe [***NVMe*** uređaja](). To u normalnom radu zvuči nevjerojatno mnogo, ali u svakom slučaju od Linux **kernela 3.13** (2014.g.) uveden je još jedan model rada na ovoj razini. Pogledajmo logičku shemu oba sustava; stari s linux *I/O schedulerom* i novi koji koristi takozvani ***Blk-mq***, vidljivog na slici 131.

**Slika 131. Pogled na još noviji model diskovnog podsustava Linuxa.**




***Blk-mq*** jednostavno se integrira u Linuxov diskovni model te osigurava osnovne funkcije upravljačkim programima kako bi preslikali upite, odnosno sve I/O operacije prema nižim dijelovima diskovnog pôdsustava, dolje prema sâmim diskovima i to u više nizova (Engl *queues*).

To se odrađuje tako da se svi zadaci prema nižim dijelovima diskovnog podsustava, odnosno u konačnici prema disk kontroleru i sâmim diskovima, distribuiraju preko više nîti kernel programa i stoga se mogu i vezati za više CPU jezgri (Engl. *per-core softver queues*).

Dakle ovim modelom se dodalo ono što je nedostajalo prijašnjem modelu.

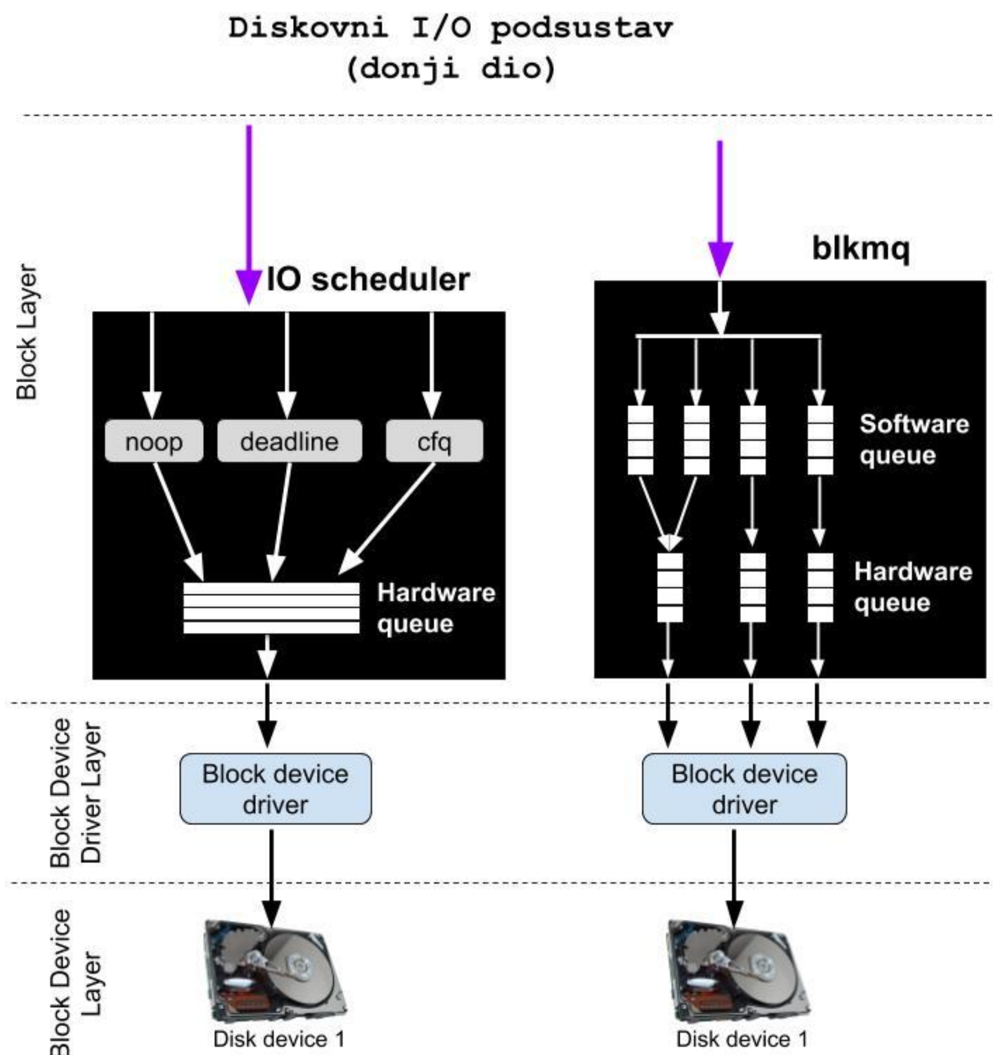
Kako bi to sve uopće radilo, potrebni su *Blk-mq* kompatibilni upravljački programi (kernel moduli) koji informiraju *blk-mq* komponentu koliko paralelnih hardverskih nîzova (*I/O queues*) podržava uređaj.

Naime mnogi napredniji disk kontroleri mogu imati više I/O nîzova to jest redova međumemorije za primanje i/ili slanje podataka, kako smo vidjeli u prethodnoj cjelini. Svaki od tih hardverskih nîzova (I/O *queue*) vidljiv je kao zasebna hardverska komponenta sustava, koja koristi svoj signal prekida (**IRQ**). Upotrebom *Blk-mq* novi upravljački programi zaobilaze standardni *Linuxov I/O scheduler*. Dakle kod njih se *Linuxov I/O scheduler* s nekim od pripadajućih mehanizama; ***noop***, ***anticipatory***, ***deadline*** ili ***cfq***, uopće ne koristi. S druge strane, svi standardni/klasični upravljački programi (kernel moduli) koji koriste klasičnu metodu rada i oslanjaju se na *Linuxov I/O scheduler* mogu neovisno o ***Blk-mq*** upravljačkim programima, nastaviti normalno raditi. Pogledajmo listu nekih od novih upravljačkih programa koji su napisani od početka, a koriste ***Blk-mq*** značajku. U tablici vidimo i inačicu kernela od koje su podržani.

| Upravljački program (kernel modul) | Ime linux uređaja | Za koje uređaje je namijenjen | Inačica (verzija) linux kernela |
|---|---|---|---|
| **virtio_blk** | /dev/vd* | KVM/QEMU ***VirtIO*** virtualni disk | 4.0 |
| **virtio_scsi** | /dev/sd* | KVM/QEMU ***VirtIO*** virtualni disk | 3.8 |
| **scsi_mq** (**scsi_mod,sd_mod**) | /dev/sd* | SAS i SCSI diskovi | 3.13 (4.10) |
| **loop** | /dev/loop* | *Loopback* uređaji | 4.16 |
| **rbd** | /dev/rdb* | ***CEPH*** *Rados* blok uređaj | 4.0 |
| **nvme** | /dev/nvme* | NVMe diskovi | 3.19 |
| **dm_mod** | /dev/dm-* | *Device Mapper* (pr. za LVM2) | 4.12 |
| **md_mod, raid*** | /dev/md* | MD RAID | 4.15 |
| **ahci** | /dev/sd* | SATA diskovi | 4.8 |

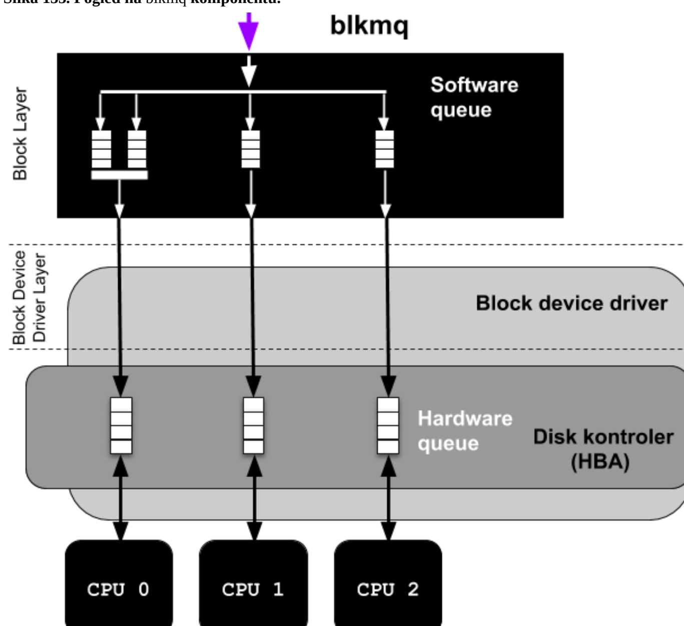
Na slici 132 pogledajmo i detaljniju shemu, sâmo donjeg dijela diskovnog sustava; s usporedbom standardnog ***Linux I/O schedulera*** *(lijevo)* i novog ***blk-mq*** *(desno).*

**Slika 132. Pogled na novi model diskovnog I/O podsustava, s detaljima.**




Ako sada pogledamo sâmo ***Blk_mq*** model (slika 133 dolje), vidjet ćemo sljedeće.

**Slika 133. Pogled na** blkmq **komponentu.**




U slučajevima kada imamo upravljački program (kernel modul), ali i disk ili **RAID** kontroler koji ga koriste i imaju podršku u hardveru za više hardverskih *disk I/O* nîzova, događa se sljedeće. Moguće je imati više softverskih nîzova ili međumemorija (*software disk I/O queue*) od kojih se čak više njih može povezati sa sâmo jednim hardverskim nîzom (lijevi dio slike). Odnosno u drugom slučaju se svaki softverski *disk I/O nîz* može povezati s jednim hardverskim nîzom (desni dio slike), prema principu 1:1.
U svakom slučaju po jedan hardverski *disk I/O nîz* koristi po jedan signal prekida (**IRQ**), a koji u konačnici odrađuje jedna jezgra procesora (CPU). Dakle isto kao kod naprednih disk/**RAID** kontrolera o kojima smo govorili.

Međutim ovdje imamo i softverske *disk I/O nîzove* za koje je također zadužena po jedna jezgra procesora, a koji se odrađuju na razini kernel programa (*kernel space*).

Na ovaj način su se mnoge operacije prema diskovnom podsustavu u potpunosti paralelizirale i rasporedile na više jezgri procesora (CPU) što jasno povećava sveukupne performanse jer se tek sada apsolutno sve operacije prema nižim dijelovima diskovnog podsustava mogu odrađivati paralelno odnosno istovremeno.



**Dalje optimizacije ovise o tome imamo li minimalno linux kernel 3.13 ili noviji.**
**Mada su potpunije optimizacije uvedene u kernelima 4.x+ ili se tek očekuju u novijim 5.x inačicama.**



Optimizacije koje ćemo ovdje spominjati često nisu najbolje rješenje kod upotrebe na mehaničkim (*rotacijskim*) diskovima ili kod upotrebe na pojedinim mehaničkim diskovima koji nisu u **RAID** polju (pr. na **RAID** kontroleru).
To je stoga jer se klasični linuxov disk (*I/O) scheduler* jako dobro ponaša zbog način rada u kojemu praktično pretvara nasumični pristup (*random*) u pristup u slijedu (*sequential*) odnosno u nîzu, a koji je bolje optimiziran za mehaničke diskove.
Ipak već nakon kernela 3.13 uočeno je kako će i ovdje biti potrebno uvesti nekoliko novih mehanizama, slično kako ih i klasični linuxov disk (*I/O) scheduler* ima: ***deadline***, ***noop***, ***cfq***, … Naime niti jedan model nije univerzalno najbolji za različite namjene pa je odlučeno kako će se razviti više njih. Stoga ovdje nećemo objašnjavati detalje, već je na vama da se u budućnosti pozabavite s ovim područjem i proučite koja metoda je najbolja za vaš konkretni sustav.

Pogledajte trenutnu usporedbu klasičnog Linuxovog disk (*I/O) schedulera* - oznaka ***SQ*** (*Single Queue*) u odnosu na osnovni mehanizam novog *blkmq* – oznaka ***MQ*** (*Multi Queue*) te načina rada/mjerenja u kojem je zaobiđen cijeli linuxov *blok* sloj vidljiv kao ***RAW***, koji možemo promatrati više kao teoretski model, kako je vidljivo na slici 134.

**Slika 134. Rezultati mjerenja.**

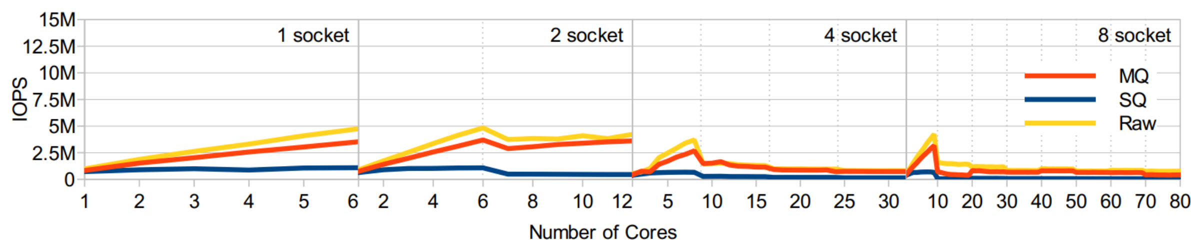



**Figure 6: IOPS for single/multi-queue and raw on the 1, 2, 4 and 8-nodes systems.**

**Izvor mjerenja, slike i tablice latencije (dolje): http://kernel.dk/systor13-final18.pdf**

Iz ovih mjerenja je vidljivo sljedeće:

- Kako se linuxov standardni model disk (*I/O) schedulera* (***SQ***) uopće ne može skalirati povećanjem broja CPU jezgri, te u svakom slučaju ostaje na maksimalnoj granici od 1.000.000 (1 milijuna) IOPS (ulazno/izlaznih) operacija prema pojedinom blok uređaju (disku).
- Kako se novi *blockmq* (***MQ***) i to dvo razinski, skalira povećanjem broja CPU jezgri, do 3.500.000 (3,5 milijuna) pa sve do 5 milijuna IOPS operacija. Međutim povećanjem broja CPU jezgri do granice kada se počinju koristiti CPU jezgre drugih *NUMA* procesora (*Sockets/Nodes*) se više od toga ne može skalirati. To je ipak poboljšanje od tri i pol do čak pet puta više od standardnog linuxovog modela disk (*I/O) schedulera*. Važno je i to da se kašnjenje (latencija) nije uopće (za 2 *NUMA CPU*) promijenilo, odnosno povećao se tek kod više *NUMA* procesora, odnosno tek kod osam *NUMA/Socketa*. Dakle tek kod sustava koji imaju konfiguraciju s osam fizičkih procesora na matičnoj ploči.

| | 1 socket | 2 sockets | 4 sockets | 8 sockets |
|---|---|---|---|---|
| SQ | 50 ms | 50 ms | 250 ms | 750 ms |
| MQ | 50 ms | 50 ms | 50 ms | 250 ms |
| Raw | 50 ms | 50 ms | 50 ms | 250 ms |

**Novosti u novijim kernelima**
U kernelu 4.12 uvedeni su ***BFQ*** i ***Kyber*** *scheduleri*, da bi se u novijim kernelima pojavio i ***mq-deadline*** scheduler unutar ***Blk-mq* modela**.

Što nam donose ovi novi disk *scheduleri* za ***mq*** (*multi queue*):

- ***Mq-deadline*** je donio klasičnu *deadline* funkcionalnost u ***Blk-mq*** model pa je sâmim time dobar odabir kod uređaja koji imaju podršku za višestruke nîzove (*multi queue*), a ***deadline*** mehanizam im najbolje odgovara po logici rada.
- ***BFQ*** odnosno *Budget Fair Queueing* je pak druga priča, jer je on razvijen s namjerom da se smanji latencija prema višim slojevima (u konačnici aplikacijama) te poveća propusnost, slično kao što omogućavaju: ***cfq***, ***noop*** i ***deadline*** kod klasičnog (starog modela). Pri tome on uspijeva distribuirati propusnost kroz više nîzova (*queues*): svaki nîz za jednu krajnju aplikaciju (koliko je moguće) i time se kod uređaja koji to podržavaju, postižu veće performanse.
- ***Kyber*** *scheduler* je razvio ***Facebook***, a on je dizajniran kao znatno jednostavniji od ***BFQ*** (ima samo oko 1.000 linija programskog kôda) i manje je kompleksnosti. On se u načelu svodni na to da interno ima dva zasebna nîza za izvršavanje (*schedulera*) i to jedan za *sinkrone*, a jedan za *asinkrone* vrste upita, što obično znači: jedan za operacije pisanja, a jedan za operacije čitanja.

**Izvori informacija: (374),(382),(383),(391),(392),(393),(394),(395),(559),(560),(561).**

## 14.1.5. Diskovni I/O sustav: primjeri

U ovoj cjelini proći ćemo primjere vezane za diskovni I/O sustav i njegovu optimizaciju.

**1.** Kako za disk `/dev/sda` provjeriti koji standardni *I/O scheduler* koristi?

```
cat /sys/block/sda/queue/scheduler

noop anticipatory deadline [cfq]
```

Vidimo kako se ovdje koristi `[cfq]`

**2.** Promijenimo standardni *I/O scheduler* iz `cfq` u `deadline` za disk `/dev/sdb` na kojem imamo *SQL* bazu podataka za koju smo na osnovu pokrenutih testova dobili bolje rezultate s `deadline` disk (*I/O) schedulerom*. Ova promjena iz primjera neće biti trajna odnosno traje samo do prvog restarta računala. Za trajnu konfiguraciju pogledajte primjer broj **4.**
Dakle za privremenu promjenu, napravimo sljedeće:

```
echo deadline > /sys/block/sdb/queue/scheduler
```

**3.** Kako provjeriti statistike pristupa disku `/dev/sda` ?
***Prva metoda***: sadržaj datoteke `/sys/block/sda/stat` - ovo je datoteka u kojoj se zapisuju statistike pojedinog diska (`/dev/sda`).

***Druga metoda***: sadržaj datoteke `/proc/diskstats` - ovo je metoda za statistiku svih diskova. Pogledajmo ovu datoteku:

```
cat /sys/block/sda/stat

200441 1457 7037314 1023095 22728 71 352178 24031 0 113482 1047037
```

Statistike `/proc/diskstats` se zapisuju u realnom vremenu, a podijeljene su u 11 dijelova odnosno segmenata.

Pogledajmo opis segmenata ovih statistika (od lijevo na desno):

- **1** - *`read I/Os` requests* - ukupan broj uspješnih disk (I/O) operacija čitanja.
- **2** - *`read merges` requests* - ukupan broj disk (I/O) operacija čitanja koje su zbog optimizacije uspješno odrađene kroz jedan zahtjev. Ovo se događa u slučajevima kad je dvije operacije čitanja moguće odraditi u jednom koraku (pr. kada se u dva koraka moraju čitati susjedni sektori, a moguće ih je spojiti u jedan zahtjev za čitanje).
- **3** - *`read sectors`* - ukupan broj sektora koji su uspješno pročitani.
- **4** - `read ticks` *milliseconds* - vrijeme potrebno kako bi se pročitao pojedini sektor. Ovo vrijeme je kumulativno (povećava se za svaki pročitani sektor).
- **5** - *`write I/Os` requests* - ukupan broj uspješnih disk (I/O) operacija pisanja.
- **6** - *`write merges` requests* - ukupan broj disk (I/O) operacija pisanja koje su zbog optimizacije uspješno odrađene kroz jedan zahtjev. Ovo se događa u slučajevima kad je dvije operacije zapisivanja moguće odraditi u jednom koraku (sličnom kao i točka **2** samo za zapisivanje).
- **7** - *`write sectors`* - ukupan broj sektora koji su uspješno zapisani.
- **8** - *`write ticks`* - vrijeme (u ms) potrebno kako bi se zapisao pojedini sektor. Ovo vrijeme je kumulativno (povećava se za svaki zapisani sektor).
- **9** - *`in_flight`* - trenutni broj ulazno/izlaznih (I/O) operacija u sekundi (tzv. ***IOPS***).
- **10** - *`io_ticks`* - vrijeme (u ms) potrebno za I/O operacije (vrijeme je isto kumulativno).
- **11** - *`time_in_queue`* -vrijeme (u ms) potrebno za sve I/O operacije.

**4.** Primjer permanentne (trajne) konfiguracije *disk schedulera*.



Za permanentnu konfiguraciju *disk schedulera* pogledajte poglavlje:
**11.1.2.1.1 Udev pravila (*udev rules*).**

**Izvori informacija:** `man 5 sysfs`, `man 5 procfs`.

### 14.1.5.1. Ograničenje diskovnih (I/O) performansi programa/procesa

U slučajevima kada imamo potrebu, moguće je ograničiti diskovne (I/O) performanse pokrenutog programa (procesa) ili onoga koji tek pokrećemo. To se postiže naredbom `ionice`. Naime ova naredba je slična naredbi `nice` koja za određeni program/proces može smanjiti ili povećati prioritet obrade (CPU), ali ona za razliku od nje smanjuje ili povećava propusnost prema diskovnom (I/O) sustavu. Drugim riječima moguće je ograničiti brzinu pristupa prema diskovnom sustavu.

Važno je razumjeti kako ovdje imamo tri kategorije od kojih svaka može imati svoje vrijednosti odnosno razine prioriteta, kao što je vidljivo u tablici:

| Klasa ili kategorija | Brojčana oznaka klase | Prioritet |
|---|---|---|
| *U realnom vremenu (Real time)* | 1 | Osam (8) razina prioriteta. |
| *Najbolje (koliko je) moguće (Best-effort)* | 2 | Osam (8) razina prioriteta: 0-7, pri čemu je nula (0) najveći prioritet. |
| *U posebnom stanju čekanja (Idle)* | 3 | Nema definicije prioriteta. |

Ovdje je moguć odabir jedne od tri klase (kategorije) i unutar odabrane klase mijenjati prioritet. Imamo sljedeće klase:

- ***Idle*** klasu: u kojoj proces može čitati ili pisati prema disku samo u trenucima kada niti jedan drugi proces ne pristupa disku. Ovdje nemamo prioritet koji treba definirati.
- ***Best effort*** klasu: koja je standardna, a u kojoj imamo osam razina prioriteta koje možemo definirati, kod kojih je nulti (0) prioritet najveći, a sedmi (7) najmanji i najsporiji.
- ***Real time*** klasu: koja ima najveći mogući prioritet prema disku (i koji treba izbjegavati jer može usporiti sve ostale programe/procese).

Kako vidjeti trenutni prioritet procesa, ako je ***PID*** našeg procesa `55140` ?. Za ovu potrebu koristit ćemo naredbu `ionice`:

```
ionice -p 55140
```

S druge strane možemo promijeniti klasu i prioritet za isti proces (***PID*** `55140`).
Dakle postavimo klasu na ***Best-effort*** (`2`), a prioritet postaviti na najviši (`0`):

```
ionice -c2 -n0 -p 55140
```

Također je moguće pokrenuti novi program ili skriptu s promjenom; primjerice pokrenimo našu skriptu `/root/skripta.sh` s istim postavkama diskovnog prioriteta kao u prijašnjem primjeru:

```
ionice -c2 -n0 /root/skripta.sh
```

Sažetak rada naredbe `ionice`:

- Naredba `ionice` u Linuxu omogućava korisnicima i administratorima da podešavaju ili prikazuju klasu i prioritet **I/O** (input/output) operacija za određene procese. To je korisno za kontrolu pristupa I/O resursima na sustavu, posebno u situacijama kada više procesa istovremeno želi koristiti diskovne resurse.
- Optimizacijom I/O operacija može se značajno poboljšati performanse sustava u situacijama visokog opterećenja diskovnim operacijama.
- Moguće je pokretati nove procese sa specifičnim I/O prioritetima ili modificirati prioritete postojećih procesa (pokrenutih programa), kao i prikazivati trenutne prioritete i klase za određene procese.

**Izvori informacija:** **(567)**,**(568)**,`man ionice`.

# 14.2. Praćenje performansi I/O sustava

**Slijedi napredno poglavlje (14.2.x)!**
Za praćenje performansi diskovnog (I/O) sustava potrebno nam je nekoliko osnovnih alata. Instalirat ćemo paket koji sadrži većinu alata koji će nam biti potrebni i za praćenje performansi drugih komponenti našeg sustava: CPU, *Context switch*, RAM, mreža, … Prvo instalirajmo paket `sysstat` na sljedeći način (on je na nekim sustavim već instaliran, ali bolje je provjeriti):

```
yum -y install sysstat
```

Nakon instalacije ovog paketa bit će nam dostupno nekoliko programa odnosno alata, koje ćemo ukratko objasniti:

- `sar` - prikuplja i prikazuje SVE aktivnosti sustava.
- `sadc` - odnosno "*system activity data collector*". Ovo je alat koji u pozadini prikuplja sve statistike za `sar` alat.
    - `sa1` prikuplja statistike u binarnom formatu. `sa1` ovisi o `sadc` od kojeg zapravo prikuplja podatke. Obično se pokreće preko `cron`-a, u određenim vremenskim intervalima.
    - `sa2` kreira dnevnu statistiku (*summary*) prikupljanih podataka. Također se obično pokreće preko `cron`-a, u određenim vremenskim intervalima (jedan puta na dan).
- `sadf` - on je alat koji služi za kreiranje `sar` izvještaja u ***CSV***, ***XML*** i drugim formatima.
- `iostat` - kreira CPU i disk (I/O) statistike.
- `mpstat` - kreira CPU statistike.
- `pidstat` - kreira statistike za pokrenute procese ili pojedini proces (prema proces ID-u (***PID***)).
- `nfsiostat` - kreira I/O statistike za NFS mrežni datotečni sustav.
- `cifsiostat` - kreira I/O statistike za CIFS mrežni datotečni sustav.

Pošto govorimo o diskovnom (I/O) sustavu, pogledajmo nekoliko najčešće traženih statistika našeg I/O sustava.

## Primjeri

**1. Pogledajmo sveukupne I/O aktivnosti** pomoću naredbe `sar`. Prikupljat ćemo statistiku u realnom vremenu, svaku sekundu (`1`) te to ponavljati pet (`5`) puta. Dakle imat ćemo pet statistika nakon pokretanja sljedeće naredbe

```
sar -b 1 5

Linux 2.6.32-42-pve (server1)  11/11/2015      _x86_64_        (4 CPU)
20:40:26 AM       tps      rtps      wtps   bread/s   bwrtn/s
20:40:27 AM     11.00      0.00     11.00      0.00     28.00
20:40:28 AM      3.00      0.00      3.00      0.00     32.00
20:40:29 AM      7.00      0.00      7.00      0.00     64.00
20:40:30 AM      4.00      0.00      4.00      0.00    120.00
20:40:31 AM     16.00      0.00     16.00      0.00    360.00
Average:         8.20      0.00      8.20      0.00    120.80
```

Vidljivo je da prosječno imamo: 8,2 **IOPS** tj. I/O operacija u sekundi (`tps`). U trenutku analize nismo imali operacije čitanja sa diskova (`rtps`) te smo imali oko 8,2 operacija zapisivanja na disk (`wtps`).

Zatim u trenutku analize nismo imali operacije čitanja sa diskova (`rtps`) pa nije bilo niti pročitanih *bajta* (`bread/s`) te smo imali oko 120 *bajta* zapisivanih na disk (`bwrtn/s`)
Opis vrijednosti koje smo dobili su sljedeće:

- `tps` – broj transakcija u sekundi. Ovo se naziva i **IOPS** (uključuje i *read* i *write* operacije).
- `rtps` – broj transakcija čitanja (*Read*) u sekundi.
- `wtps` – broj transakcija zapisivanja (*Write*) u sekundi.
- `bread/s` – *bajta* pročitanih (*Read*) u sekundi.
- `bwrtn/s` – *bajta* zapisanih (*Write*) u sekundi.

**1.1 Pogledajmo sveukupne diskovne aktivnosti povezane sa sustavom virtualne memorije**, pomoću naredbe `sar -B`. Prikupljat ćemo statistiku u realnom vremenu, svaku sekundu (`1`) te to ponavljati pet (`5`) puta. Dakle imat ćemo pet statistika

```
sar -B 1 5

Linux 2.6.32-42-pve (server1)  11/17/2015      _x86_64_        (40 CPU)

21:22:34 PM  pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s %vmeff
21:22:35 PM      0.00     40.00     37.00      0.00   1081.00      0.00      0.00      0.00   0.00
21:22:36 PM      0.00      0.00     29.00      0.00    835.00      0.00      0.00      0.00   0.00
21:22:37 PM      0.00     36.00     20.00      0.00    847.00      0.00      0.00      0.00   0.00
21:22:38 PM      0.00     32.00     75.00      0.00    930.00      0.00      0.00      0.00   0.00
21:22:39 PM      0.00   1456.00     20.00      0.00    884.00      0.00      0.00      0.00   0.00
Average:         0.00    312.80     36.20      0.00    915.40      0.00      0.00      0.00   0.00
```

Opis, s time da je fokus dan na parametre važne za I/O sustav i sustav virtualne memorije povezan s diskovnim I/O sustavom:

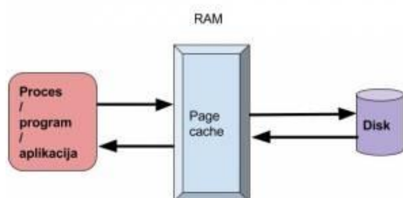
- `pgpgin/s` - ukupan broj kB koje je sustav povukao iz međumemorije* (*page cache*) s diska u svakoj sekundi. Pod *page* se smatra povlačenje podataka s diskovnog I/O sustava iz međumemorije i u konačnici s diska.
- `pgpgout/s` - ukupan broj kB koje je sustav poslao iz međumemorije* na disk (na zapisivanje) u svakoj sekundi. Pod *page* se smatra slanje podataka na diskovni I/O sustav, prema priručnoj memoriji i u konačnici na disk.

- `fault/s` - (*Minor Fault*) - broj standardnih *page fault* grešaka u dohvaćanju podataka prema sustavu virtualne memorije u sekundi. Također je mogući uzrok ovih grešaka dostupnost *page*-a* u virtualnoj memoriji, ali u tom trenutku nije alociran u adresni prostor procesa/programa/aplikacije. Ovo ne označava upit prema diskovnom I/O sustavu jer se većina ovih grešaka odnosi na pristup sustavu virtualne memorije za koje ionako nije potreban disk I/O.
- `majflt/s` - (*Major Fault*) - broj grešaka tijekom čitanja (*read*) koje se događaju kada se pristupa virtualnom memorijskom adresnom prostoru koji nije učitan u međumemoriju* (*page cache*) te se mora ići prema diskovnom I/O sustavu. Ako se ovdje krenu pojavljivati greške to ukazuje da procesi koji pristupaju diskovnom I/O sustavu prečesto moraju ići prema diskovnom I/O sustavu. Ovo može ukazivati i na problem s nedostatkom slobodne RAM memorije. **Teži se da ove vrijednosti budu što manje.**

Dodatne opcije koje su važne za dublje razumijevanje i analizu I/O sustava

- `pgscank/s` - broj stranica (*pages*) skeniranih od strane `kswapd` servisa, u svakoj sekundi. `kswapd` servis je zadužen za skeniranje raspoložive RAM memorije. Kada raspoloživa RAM memorija padne ispod neke točke, ovaj servis započinje alociranje *Swap* memorije odnosno *swap* particije na tvrdom disku, kao proširenja RAM memorije.
- `pgscand/s` - broj stranica (*pages*) skeniranih od strane sâmog sustava (ovdje se traže oslobođene memorijske lokacije) te se indirektno i oslobađa RAM memorija koja je oslobođena primjerice od strane sâmog procesa/programa odnosno aplikacije jer više nije u upotrebi.
- `pgsteal/s` - broj stranica (*pages*) koje je sustav uzeo iz priručne memorije (*pagecache* i/ili *swapcache*) u sekundi. Dakle sustav koristi što veću količinu memorije za *page cache** kako bi ubrzao operacije prema diskovnom I/O sustavu, kada cijeli sustav padne ispod neke granice s dostupnom (slobodnom) RAM memorijom on pokreće ovaj proces u kojemu uzima ili bolje reći posuđuje RAM memoriju za svoj rad ili za nove procese (programe).
- `%vmeff` - računa se kao *pgsteal/pgscand*, ovaj parametar je i metrika koja govori o efikasnosti `pgsteal` mehanizma. Možemo ga promatrati i ovako:
    - Ako je 100% to znači da je svaka stranica (*page*) koja je slobodna (ili u stanju *inactive*) uspješno i oslobođena.
    - Ako je ispod 30% tada postoje neki problemi ili procesi koji usporavaju ovu proceduru.
    - Ako je vrijednost 0% to znači da u trenutku vremena kada se prikupljala statistike nije bilo uopće procesa skeniranja ili oduzimanja (*steal*).

**Slika 135. Veza između programa, međumemorije i diska**




Pojam ***page**** koji se spominje kod diskovnog sustava se odnosi na *page cache* odnosno priručnu memoriju prema diskovnom sustavu, a ne na *page cache* tablice ili druge mehanizme ili pojmove vezane za sustav virtualne memorije. Ovo je malo zbunjujuće zbog gotovo istih naziva. U svakom slučaju standardna veličina stranice je 4kB jer je to standardna veličina stranice na razini memorije: točnije na razini sustava virtualne memorije. Pojednostavljeno veza između procesa/programa/aplikacije, *page cache* i diska je vidljiva na slici 135.

### 1.2 Pogledajmo detaljnije diskovne (I/O) operacije, ali prema svakom disku zasebno, pomoću naredbe `sar`.

Provjere ćemo raditi svake sekunde (`1`) i to dva puta (`2`) pošto će ispis biti malo duži. Naš sustav ima dva tvrda diska:

- Disk na koji je instaliran operacijski sustav: `/dev/sda`
- Disk za podatke: `/dev/sdb`

Pokrenimo naredbu `sar` na sljedeći način:

```
sar -p -d 1 2

Linux 2.6.32-42-pve (server1)  11/11/2015      _x86_64_        (4 CPU)
20:56:01 AM  DEV   tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz   await   svctm   %util
20:56:02 AM  sda   0.00     0.00      0.00      0.00      0.00    0.00    0.00    0.00
20:56:02 AM  sdb   4.00     0.00     36.00      9.00      0.00    0.00    0.00    0.00

20:56:02 AM  DEV   tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz   await   svctm   %util
20:56:03 AM  sda   0.00     0.00      0.00      0.00      0.00    0.00    0.00    0.00
20:56:03 AM  sdb   0.00     0.00      0.00      0.00      0.00    0.00    0.00    0.00

Average:     DEV   tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz   await   svctm   %util
Average:     sda   0.00     0.00      0.00      0.00      0.00    0.00    0.00    0.00
Average:     sdb   2.00     0.00     18.00      9.00      0.00    0.00    0.00    0.00
```

Na ispisu vidimo još više detalja od kojih je `tps` zapravo ***IOPS*** (kao i prije). Nadalje: `rd_sec/s` i `wr_sec/s` su statistike operacija čitanja i pisanja na disk. Dodatne nove statistike su:

- `avgrq-sz` - označava prosječnu veličinu u sektorima podijeljen s brojem diskovnih (I/O) operacija u sekundi koje se traže (obrađuju) prema disku.
- `avgqu-sz` - označava prosječan broj I/O operacija koje su još uvijek na obradi (*average queue length*).

Ono što je sada zanimljivo su statistike pred kraj ispisa:

- `await` - označava ukupno vrijeme za sve I/O operacije (sumarizirano) te podijeljeno s ukupnim brojem I/O operacija koje su uspješno odrađene. Dakle ovo je vrijeme koje govori koliko je I/O operacija obrađivano unutar *I/O Scheduler-a*, od ulaska u I/O *scheduler* do njihove obrade.
    - Dobra vremena trebaju biti što manja; ispod ili oko jedne milisekunde za disk i za do nekoliko milisekundi za vanjske ili sporije diskove.

- `svctm` - je prosječno vrijeme obrade (u milisekundama) za I/O zahtjeve prema diskovima. Dakle ovo vrijeme govori koliko je vremena bilo potrebno za svaku I/O operaciju prema disku. Odnosno koliko je disku trebalo vremena da ih odradi.
    - Očekivana vremena se kreću ispod ili oko jedne milisekunde.
    ***NAPOMENA: Ova opcija se više neće koristiti u budućnosti!.***
- `util` - je postotak CPU vremena za obradu tijekom kojeg su I/O zahtjevi poslani prema uređajima (diskovima). Veliki postotak znači da CPU čeka na diskovni sustav koji je zbog nekog razloga usporen, pa CPU troši više vremena na čekanje prema disku nego na stvarnu obradu podataka.

**1.3 Pogledajmo detaljnije diskovne (I/O) operacije, ali prema svakom disku zasebno pomoću naredbe `iostat`.**
Sada ćemo dobiti slične statistike kao iz prethodnog primjera ali pomoću naredbe `iostat`.
Koristit ćemo prekidač `-m` kako bismo dobili ispis u MB/s :

```
iostat -m

Linux 2.6.32-42-pve (server1)  11/11/2015      _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           4.22    0.00    1.09    0.01    0.00   94.69


Device:            tps    MB_read/s    MB_wrtn/s    MB_read    MB_wrtn
sda               1.43         0.00         0.01        631      13286
sdb              25.24         0.42         0.48     460796     529849
```

Dakle ovdje je sve prilično jasno :

- `MB_read/s` - označava MB/s pročitanih s diska.
- `MB_wrtn/s` - označava MB/s snimanja na disk.
- `tps` - označava broj transakcija u sekundi; ovo se označava i kao **IOPS** (*Input Output Per Second*).
- `MB_read` - označava ukupno MB/s pročitano s diska.
- `MB_wrtn` - označava ukupno MB/s snimljeno na disk.

**1.3.1 Pogledajmo još detaljnije diskovne (I/O) operacije prema svakom disku zasebno, pomoću naredbe `iostat` s dodatnim prekidačima.**
Sada ćemo dobiti slične statistike kao iz prethodnog primjera, ali pomoću naredbe `iostat` uz koju ćemo koristiti prekidač `-m` te dodatno i prekidač `-x` kako bismo dobili dodatne napredne statistike:

```
iostat -mx

Linux 2.6.32-42-pve (server1)  11/11/2015      _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.17    0.00    0.52    0.01    0.00   98.30

Device:  rrqm/s  wrqm/s    r/s    w/s  rMB/s   wMB/s avgrq-sz avgqu-sz   await r_await w_await svctm  %util
sda        0.01    1.08   4.45   1.57   0.01    0.03    14.95     0.00    0.26    0.30    0.15  0.19   0.12
sdb        3.05    9.59   7.68  16.71   0.32    2.11   204.15     0.51   20.74    0.62   29.99  0.34   0.82
```

Sada imamo malo drugačije odnosno znatno detaljnije statistike:

- `rrqm/s` - je broj zahtjeva za čitanje (*READ*) u sekundi koji su spojeni (*merge*) i zajedno poslani u nîz za izvršavanje (*queue*) operacija čitanja.
- `wrqm/s` - je broj zahtjeva za zapisivanje (*WRITE*) u sekundi koji su spojeni (*merge*) i zajedno poslani u nîz za izvršavanje (*queue*) operacija zapisivanja.
- `r/s` - je broj zahtjeva za čitanje u sekundi, koji su nakon što su zaprimljeni u nîz za izvršavanje (*queue*) i odrađeni (pročitani) od strane uređaja (diska).
- `w/s` - je broj zahtjeva za zapisivanje u sekundi, koji su nakon što su zaprimljeni u nîz za izvršavanje (*queue*) i odrađeni (zapisani) od strane uređaja (diska).
- `rMB/s` - je broj sektora koji su pročitani s diska u sekundi.
- `wMB/s` - je broj sektora koji su snimljeni na disk u sekundi.
- `avgrq-sz` - označava prosječnu veličinu (u sektorima) upita koji je poslan prema disku.
- `avgqu-sz` - označava veličinu *queue* (nîza) upita koji je poslan na disk.
- `await` - je ukupno vrijeme u milisekundama (*ms*) za I/O upit prema disku. Ovo vrijeme uključuje i vrijeme koje je upit proveo u nîzu za izvršavanje (*queue*) te vrijeme koje je sâmom disku trebalo da ga obradi, koje se mjeri u *`svctm`* (*service time*) vremenu.
- `r_await` - je srednje vrijeme u milisekundama za upit za zapisivanje (*READ request*) koje također uključuje i vrijeme koje je upit proveo u nîzu za izvršavanje (*queue*) i vrijeme koje je disku trebalo da ga obradi.
- `w_await` - je srednje vrijeme u milisekundama za upit za čitanje (*WRITE request*) koje također uključuje i vrijeme koje je upit proveo u nîzu za izvršavanje (*queue*) i vrijeme koje je disku trebalo da ga obradi.
- `svctm` - je vrijeme u milisekundama koje je potrebno da se diskovni (I/O) zahtjev obradi od strane sâmog uređaja (diska). Dakle ovo vrijeme pokazuje koliko je vremena trebalo uređaju (disku) da završi zahtjev čitanja ili zapisivanja, što nema veze s operativnim sustavom već sâmim diskom (i/ili disk ili RAID kontrolerom). Samim time mora biti manje od vremena `await` koje prikazuje ukupno vrijeme koje uključuje i ovo (`svctm`) vrijeme.

- `%util` postotak CPU vremena za obradu tijekom kojeg su diskovni (I/O) zahtjevi poslani prema uređaju (disku). Veliki postotak znači kako *CPU* čeka na diskovni podsustav koji je zbog nekog razloga usporen, pa CPU troši više vremena na čekanje prema disku nego na stvarnu obradu podataka odnosno čitanje ili pisanje na disk.

Vezano za program `iostat` on statistike diskova u realnom vremenu čita iz sljedećih datoteka:

- `/proc/stat` - ovdje se u realnom vremenu snimaju razni sistemski i podaci iz kernela: *idle*, *iowait*, *irq*, *softirq*, … Ova datoteka sadrži prilično puno podataka, pa ju sada nećemo objašnjavati. Za više informacija [(218)]*.
- `/proc/diskstats`[(217)] - ovdje se u realnom vremenu snimaju statistike vezane za sâm diskovni podsustav, pa se iz ove datoteke i čita veći dio podataka, koji je podijeljen u nekoliko stupaca:
  - `1` - *Major (značajniji)* broj diska. Pogledajte poglavlje: **11.1.2.1. Uređaji (*devices*) detaljnije.**
  - `2` - *Minor (manje značajan)* broj diska. Pogledajte poglavlje: **11.1.2.1. Uređaji (*devices*) detaljnije.**
  - `3` - Ime diska.
  - `4` - Koliko operacija čitanja je odrađeno uspješno.
  - `5` - Koliko operacije čitanja je spojenu u jedan upit za čitanje (*read merge*).
  - `6` - Koliko je sektora pročitano.
  - `7` - Vrijeme potrošeno za operaciju čitanja (*ms*).
  - `8` - Koliko operacija zapisivanja je odrađeno uspješno.
  - `9` - Koliko operacije zapisivanja je spojenu u jedan upit za čitanje (*write merge*).
  - `10` - Koliko je sektora snimljeno.
  - `11` - Vrijeme potrošeno za operaciju snimanja (*ms*).
  - `12` - Koliko je paralelnih I/O operacija trenutno.
  - `13` - Vrijeme potrošeno na I/O operacije (*ms*).
  - `14` - Težinsko vrijeme za I/O operacije: *povećava se za svaku I/O operaciju* (*I/O start, I/O završetak ili I/O merge*).

Naime i sâmi možemo pratiti ove statistike, pomoću sljedeće kombinacije naredbi 🎬:

```
cat /proc/diskstats | column  -t
```

Naredba `column` lijepo filtrira ispis datoteke: `/proc/diskstats` koju smo otvorili s naredbom `cat`.

**1.4 Pogledajmo detaljnije I/O operacije, ali za sve pokrenute procese (programe/aplikacije) pomoću naredbe `pidstat`.**
Ako naredbu `pidstat` pokrenemo s prekidačem `-d` dobit ćemo statistiku za diskovni sustav i to za sve pokrenute procese (programe/aplikacije).

```
pidstat -d

Linux 2.6.32-42-pve (server1)  11/17/2015      _x86_64_        (4 CPU)

20:06:17 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
20:06:17 PM         1     37.67      0.07      0.00  init
20:06:17 PM      1215      0.00      1.43      0.00  kjournald
20:06:17 PM      1351      0.01      0.00      0.00  udevd
20:06:17 PM      3058      0.00      0.00      0.00  xfsalloc/0
20:06:17 PM      6422      0.00      0.00      0.00  sendmail
```

Opis slijedi:

- `kB_rd/s` – označava kB/s za operacije čitanja.
- `kB_wr/s` – označava kB/s za operacije zapisivanja
- `kB_ccwr/s` – označava kB/s za operacije zapisivanja za koje je proces/program/aplikacija odustala od zapisivanja. Ovo je isti parametar kao *cancelled_write_bytes* opisan dalje u tekstu. Uglavnom želimo da je ova vrijednost nula (0).

**2. Ručni rad - pogledajmo I/O operacije, ali za pojedini pokrenuti proces**. Kada sâmo želimo vidjeti diskovne (I/O) operacije za pojedini pokrenuti program/aplikaciju, prvo saznajmo njen *process ID* odnosno ***PID*** broj.
Recimo da je to *Apache Web server*, sa svojim procesima. Saznajmo njegove ***PID****ove*:

```
pidof httpd

2738847 6441
```

Sada kada smo dobili ***PID*** brojeve, možemo gledati diskovne (I/O) operacije za svaki od tih procesa u realnom vremenu.
Sve I/O statistike se stalno zapisuju u datoteku imena `io` unutar `/proc/` direktorija i poddirektorija koji ima ime jednako ***PID*** broj procesa/programa koji pratimo. Dakle za naš ***PID*** broj `6441` je to datoteka: `/proc/6441/io`. Pogledajmo njen sadržaj:

```
cat /proc/6441/io

rchar: 334965
wchar: 901
syscr: 207
syscw: 16
read_bytes: 20480
write_bytes: 36864
cancelled_write_bytes: 8192
```

Pošto se statistika stalno osvježava svaki novi pogled u ovu datoteku daje nove statistike, jer se popunjava u realnom vremenu.

Pogledajmo što je sve dostupno od statistika:

- `rchar` je broj bajta čiji zahtjev je proslijeđen diskovnom (I/O) sustavu za operacije čitanja. Ova vrijednost se povećava (zbraja). Ovo se odnosi na čitanje iz **VFS** sloja (*Virtual File System*) te se tok dohvaćanja podataka ne prati dublje od tog sloja.
- `wchar` je broj bajta čiji zahtjev je proslijeđen diskovnom I/O sustavu za operacije zapisivanja. Ova vrijednost se isto povećava (zbraja). Ovo se odnosi na čitanje iz **VFS** sloja te se tok dohvaćanja podataka ne prati dublje od tog sloja.
- `syscr` je brojač operacija čitanja prema diskovnom (I/O) sustavu (***IOPS*** za operacije čitanja).
- `syscw` je brojač operacija zapisivanja prema diskovnom (I/O) sustavu (***IOPS*** za operacije zapisivanja).
- `read_bytes` je broj pročitanih bajta s diskovnog podsustava iz sloja "*Block Layer*".
- `write_bytes` je broj zapisanih bajta s diskovnog podsustava iz sloja "*Block Layer*".
- `cancelled_write_bytes` se odnosni na količinu podataka (u bajtima) koja je prvo zapisana pa onda obrisana te se stvarna transakcija nije zapravo dogodila na tvrdi disk već je bila odrađena u višim slojevima diskovnog podsustava. Ako se primjerice u nekom trenutku zapisuje 1MB podataka, on prvo završi u *page cache* priručnoj međumemoriji u višim slojevima diskovnog sustava, a ako se nakon nekog vremena tih istih 1MB podataka briše, a oni su još u priručnoj memoriji i nisu se niti počeli prosljeđivati nižim slojevima niti u konačnici na disk, onda se ti podaci brišu sâmo iz priručne međumemorije i zapisuju se u ovu statistiku.

*Više informacija o procfs datotečnom sustavu (`/proc/` direktorij) možete vidjeti na izvoru informacija: (**218**)*

**2.1 Pogledajmo I/O operacije, ali za pojedini pokrenuti proces, pomoću naredbe** `pidstat`. U slučaju da smo željeli vidjeti I/O operacije za pojedini pokrenuti proces/program/aplikaciju, prvo bi morali saznati njen ***process id*** odnosno ***PID*** broj.
U našem slučaju gledati ćemo statistiku našeg Apache (*httpd*) servisa čiji trenutni ***PID*** je **6441**.
Pratit ćemo statistiku u realnom vremenu, svake sekunde (`1`) četiri puta (`4`) za redom:

```
pidstat -d -p 6441 1 4

20:17:13 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
20:17:14 PM      6441      0.00      0.00      0.00  httpd
20:17:15 PM      6441      0.00      0.00      0.00  httpd
20:17:16 PM      6441      0.00      0.00      0.00  httpd
20:17:17 PM      6441      0.00      0.00      0.00  httpd
Average:         6441      0.00      0.00      0.00  httpd
```

Podaci iz statistike su isti kao i iz prethodnog primjera: kB/ za čitanje (`kB_rd/s`), pisanje (`kB_wr/s`) i "*cancelled_write_bytes*" (`kB_ccwr/s`). Ovo je vrlo zgodna naredbe jer možemo pratiti stanje pojedinog proces u realnom vremenu u vremenskim intervalima u kojima želimo pratiti stanje procesa.

**3. Pogledajmo statuse na razini VFS sloja (*Virtual Filesystem sloj*), pomoću naredbe** `sar -v`.
Sada ćemo pogledati osnovne statistike na ovom sloju. Statistike ćemo gledati svake sekunde (`1`) i to četiri (`4`) puta za redom:

```
sar -v 1 4

20:22:53 PM dentunusd   file-nr  inode-nr    pty-nr
20:22:54 PM     35271      5680     44127         2
20:22:55 PM     35271      5680     44127         2
20:22:56 PM     35276      5680     44132         2
20:22:57 PM     35276      5680     44132         2
Average:        35274      5680     44130         2
```

Opis statistika koje smo ovdje odbili je sljedeći:

- `dentunusd` - je broj neiskorištenih unosa u priručnoj memoriji za direktorije (engl. *Directory cache*).
  Kako bi se ubrzale operacije pristupa često korištenim direktorijima ***VFS sloj*** ima priručnu memoriju za direktorije u kojoj drži unose s direktorijima. To radi tako da kada datotečni sustav prvi puta pristupa nekom direktoriju ili datotekama unutar njega, datotečni sustav šalje ***VFS***-u sve podatke o tom direktoriju koje on stavlja u ovu priručnu međumemoriju. Sljedeći puta kada netko pristupa tom istom direktoriju ili datotekama unutar njega, svi potrebni podaci se čitaju iz ove brze priručne (RAM) memorije.
- `file-nr` - je ukupan broj otvorenih *File handle-ova (file deskriptora)* za cijeli sustav, trenutno.
- `inode-nr` – je ukupan broj otvorenih *Inode handle-ova (file deskriptora)* za cijeli sustav, trenutno.
- `pty-nr` – je ukupan broj pseudo terminala otvorenih na sustavu. Ne zaboravimo: ovo su isto posebne datoteke.

**4. Disk I/O wait:** Postoje i slučajevi u kojima *CPU* čeka na diskovni podsustav, jer je diskovni (I/O) podsustav preopterećen.
Možemo pokrenuti i naredbu `vmstat` s prekidačem `-d` ili `-D`, poput (skratili smo ispis samo na I/O statistike):

```
vmstat -D
            0 inprogress IO
          653 milli spent IO
```

Brojač `inprogress IO` označava trenutni broj ulazno izlaznih operacija, dok `milli spent IO` označava broj mili sekundi potrošenih na ulazno/izlazne (I/O) operacije prema diskovnom sustavu.



Pogledajte i primjere iz poglavlje: **10.7.2.3.5 Naredba vmstat za CPU** (pôd **CPU** statistikom, stupac: `wa`)

**Izvori informacija:** **`man sar`**, **`man iostat`**, **`man pidstat`**, **`man vmstat`**, **`man 5 procfs`**.

### 14.2.1. Još detaljnije praćenje I/O sustava

Ako nam alati koji dolaze unutar `sysstat` paketa nisu dovoljni te želimo još dublju analizu diskovnog (I/O) prometa, dostupan je paket `blktrace` unutar kojega dolazi nekoliko programa od kojih ćemo spomenuti njih sâmo nekoliko:

- `blktrace` - za praćenje diskovnog (I/O) sustava i pripremu izvještaja.
- `blkparse` - za analizu snimljenih izvještaja od gore navedenog programa.
  - `btt` – za obradu (*parsiranje)* i prikaz analize iz gornjeg programa.

Gornja dva alata nam omogućavaju vrlo detaljnu analizu pristupa I/O sustavu. U ovom primjeru ćemo prikupljat informacije o diskovnom podsustavu (I/O), za disk `/dev/sdb`. Prvo provjerimo imamo li *montiran* datotečni sustav ***debugfs***

```
mount | egrep debugfs
```

Ako nemamo montiran *debugfs*, moramo ga *montirati (u dva koraka)*:

```
mkdir /sys/kernel/debug
mount -t debugfs debugfs /sys/kernel/debug
```

Sada instalirajmo paket `blktrace`:

```
yum install blktrace
```

Potom možemo krenuti sa snimanjem svih I/O aktivnosti za željeni disk (`/dev/sdb`), upotrebom programa `blktrace`:

```
blktrace -w 30 -d /dev/sdb -o izlaz.sdb.out
```

S navedenim primjerom će se snimati I/O aktivnosti diska, sa svim detaljima u trajanju 30 sekundi te snimiti u datoteku: `izlaz.sdb.out`. Zatim možemo pomoću programa `blkparse` *parsirati* dobivenu *log* datoteku i prikazati na razumljiviji način. Izvještaj je u binarnom formatu pa je potreban ovaj korak. Ovisno o količini prometa prema disku, možemo imati nekoliko izlaznih datoteka. Mi smo dobili dvije a analizirati ćemo prvu (`izlaz.xvdg.out.blktrace.0`)

```
blkparse -i izlaz.xvdg.out.blktrace.0
```

Ako želimo trenutni ispis statistike, bez zapisivanja u datoteku, možemo pokrenut i naredbe:

```
blktrace -d /dev/sda -o - | blkparse -i -
```

Nakon nekoliko trenutaka možemo prekinuti ovaj program sa `CTRL c`. U svakom slučaju pozvali prve dvije naredbe ili ovu zadnju, dobit ćemo sljedeći ispis. Naime izrezali smo samo vrlo mali sitni djelić ovog ispisa razdvojen u dvije cjeline (1) i (2) kako slijedi:

```
(1)
... ... ...
202,96   0        0     0.132479634     0  m   N cfq2517S / sl_used=8 disp=6 charge=6 iops=1
sect=192
202,96   0        0     0.132480223     0  m   N cfq2517S / del_from_rr
202,96   0        0     0.132482419     0  m   N cfq2515S / set_active wl_prio:0 wl_type:1
202,96   0        0     0.132483548     0  m   N cfq2515S / fifo=(null)
202,96   0        0     0.132484188     0  m   N cfq2515S / dispatch_insert
202,96   0        0     0.132485535     0  m   N cfq2515S / dispatched a request
202,96   0        0     0.132486273     0  m   N cfq2515S / activate rq, drv=1
202,96   1      466     0.133106889     0  C   R 120927 + 32 [0]
202,96   1        0     0.133110911     0  m   N cfq2515S / complete rqnoidle 0
202,96   1        0     0.133111855     0  m   N cfq2515S / set_slice=100
... ... ...
(2)
... ... ...
CPU0 (izlaz.sdb.out):
 Reads Queued:         127,     2,618KiB  Writes Queued:          18,      157KiB
 Read Dispatches:      127,     2,618KiB  Write Dispatches:       18,      157KiB
 Reads Requeued:         0                Writes Requeued:         0
 Reads Completed:        0,         0KiB  Writes Completed:        0,        0KiB
 Read Merges:            0,         0KiB  Write Merges:            0,        0KiB
 Read depth:            17                Write depth:             2
 IO unplugs:           132                Timer unplugs:           0

Total (izlaz.sdb.out):
 Reads Queued:       1,439,    39,188KiB  Writes Queued:         157,    2,341KiB
 Read Dispatches:    1,439,    39,188KiB  Write Dispatches:      157,    2,341KiB
 Reads Requeued:         0                Writes Requeued:         0
 Reads Completed:    1,438,    39,172KiB  Writes Completed:      159,    2,349KiB
 Read Merges:            0,         0KiB  Write Merges:            0,        0KiB
 IO unplugs:         1,259                Timer unplugs:
```

Osnovni opis prve cjeline `(1)`:

- Prvi stupac je *Device major,minor* (u našem slučaju `202,96`). Ovo označava disk na najnižoj razini (u `/dev/` direktoriju), na kojem se radi određena I/O operacija.

Pogledajmo o kojem se to disku odnosno uređaju radi:

```
ls -alh /dev/ | grep "202,  96"

brw-rw----   1 root disk     202,  96 Nov 19 05:01 sdb
```

- Drugi stupac pokazuje broj CPU jezgre koja odrađuje diskovni (I/O) zadatak.
- Treći stupac pokazuje redni broj operacije (*sequence number*).
- Četvrti stupac je vrijeme odnosno vremenska oznaka.
- Peti stupac je ***PID*** broj procesa koji je zatražio diskovni (I/O) pristup.

- Šesti stupac označava vrstu *događaja*:
    - `A` - I/O je prespojen na drugi uređaj (Engl. *IO was remapped to a different device*).
    - `B` - I/O je preskočen (Engl. *IO bounced*).
    - `C` - I/O završetak (Engl. *IO completion*).
    - `D` - I/O je prebačen na upravljački program (Engl. *IO issued to driver*).
    - `G` - GET upit (Engl. *Get request*).
    - `P` - Upit za spajanjem uređaja (Engl. *Plug request*).
    - `Q` - I/O je odrađen prema određenom kôdu nîza (Engl. *IO handled by request queue code*).
    - `S` - Upit za spavanjem (Engl. *Sleep request*).
    - `U` - Upit za odspajanjem uređaja (Engl. *Unplug request*), … … …
- Sedmi stupac je:
    - `R` - Zahtjev za čitanje (Engl. *Read).*
    - `W` - Zahtjev za zapisivanje (Engl. *Write).*
    - `D` - Blok (Engl. *block*).
    - `B` - Operacija barijere (Engl. *Barrier operation).*
- Osmi stupac je disk (*I/O) scheduler* (pr. ***cfq***) + "*block number*" uz koji slijedi blok koji se traži.
- Deveti stupac odnosno vrijednost između zagrada "[ ]" je ime procesa koji radi/traži disk (I/O) aktivnost.

Više detalja nećemo objašnjavati, važno je da budete svjesni koliko detaljno je moguće analizirati diskovni (I/O) sustav.

Ako iz ovog izvještaja ipak želimo samo malo osnovnije stvari, možemo napraviti sljedeće (instalirati novi softver):

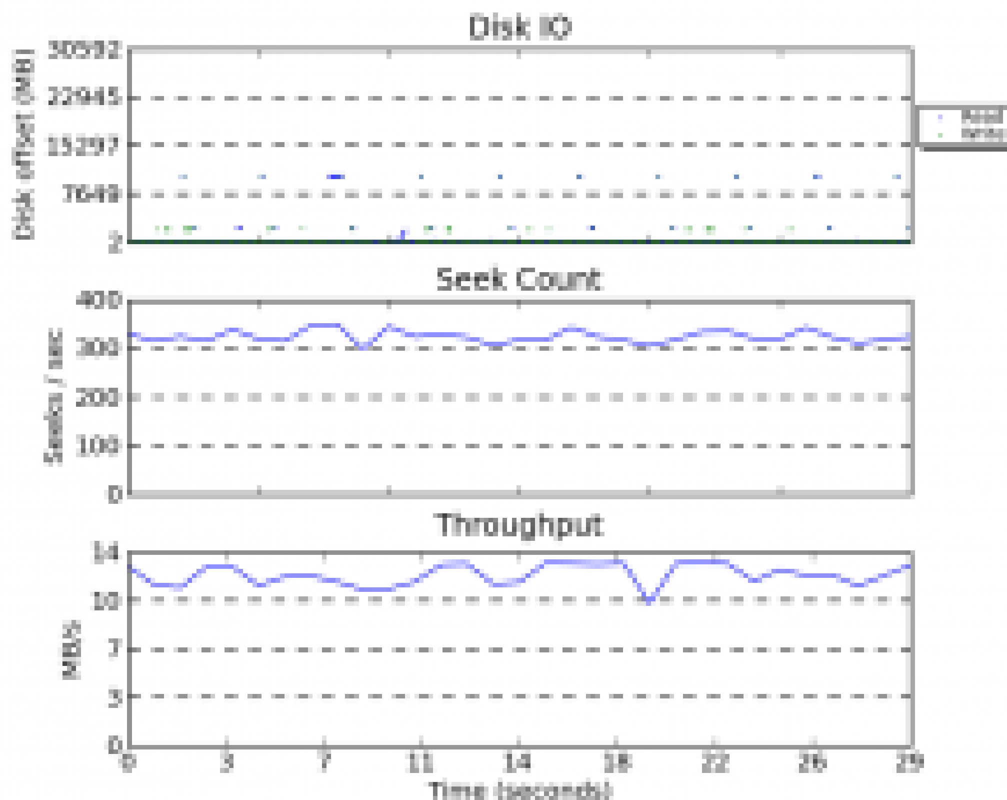
```
yum install seekwatcher
```

Sada ćemo pretvoriti izvještaj u sliku (**.png** format) pomoću novo instaliranog programa `seekwatcher`:

```
seekwatcher -t izlaz.sdb.out -o izlaz.png
```

Slika u našem slučaju izgleda ovako (slika 136):

**Slika 136. Slika predstavlja rezultate mjerenja**




Dodatno, u paketu `blktrace` dolazi i program `btt` s kojim je također moguće izraditi izvještaj. Pogledajmo i kako:

Prvo moramo izraditi odredišnu datoteku, iz svih *log* datoteka:

```
blkparse -i izlaz.xvdg.out.blktrace.* -d disk.out
```

I sada kada smo dobili statistiku u jednoj datoteci (*disk.out*), možemo ju analizirati s programom `btt` na sljedeći način:

```
btt -i disk.out | more
```

Sada dobivamo detaljnu statistiku, poput ove (koju smo skratili):

```
==================== All Devices ====================================
            ALL           MIN           AVG           MAX           N
--------------- ------------- ------------- ------------- -----------

Q2Q               0.000003699   0.010400227   2.125710371         400
Q2G               0.000000158   0.000000753   0.000013885       19200
G2I               0.000000112   0.000000543   0.000013659       19200
Q2M               0.000001335   0.000001335   0.000001335          48
I2D               0.000000115   0.000000485   0.000002823       19200
M2D               0.000002910   0.000002910   0.000002910          48
D2C               0.000027688   0.000047481   0.000197677         401
Q2C               0.000029297   0.000049269   0.000209201         401
==================== Device Overhead =========================
       DEV |       Q2G       G2I       Q2M       I2D       D2C
---------- | --------- --------- --------- --------- ---------
 (  8,  0) |  73.2023%  52.8059%   0.3243%  47.1385%  96.3713%
---------- | --------- --------- --------- --------- ---------
   Overall |  73.2023%  52.8059%   0.3243%  47.1385%  96.3713%

==================== Device Merge Information =============================
       DEV |       #Q       #D   Ratio |   BLKmin   BLKavg   BLKmax    Total
---------- | -------- -------- ------- | -------- -------- -------- --------
 (  8,  0) |    19200    19200     1.0 |        8       12      512   235776

==================== Device Q2Q Seek Information ============================
       DEV |          NSEEKS            MEAN          MEDIAN | MODE
---------- | --------------- --------------- --------------- | ---------------
 (  8,  0) |             401      26530652.9               0 | 0(300)
---------- | --------------- --------------- --------------- | ---------------
   Overall |          NSEEKS            MEAN          MEDIAN | MODE
   Average |             401      26530652.9               0 | 0(300)

==================== Device D2D Seek Information ============================
       DEV |          NSEEKS            MEAN          MEDIAN | MODE
---------- | --------------- --------------- --------------- | ---------------
 (  8,  0) |           19200        554103.7               0 | 0(19099)
---------- | --------------- --------------- --------------- | ---------------
   Overall |          NSEEKS            MEAN          MEDIAN | MODE
   Average |           19200        554103.7               0 | 0(19099)

==================== Plug Information ====================
       DEV |    # Plugs # Timer Us  | % Time Q Plugged
---------- | ---------- ----------  | ----------------
 (  8,  0) |          4(         0) |   0.000316734%

       DEV |    IOs/Unp   IOs/Unp(to)
---------- | ----------   ----------
 (  8,  0) |        1.0          0.0
---------- | ----------   ----------
   Overall |    IOs/Unp   IOs/Unp(to)
   Average |        1.0          0.0

==================== Active Requests At Q Information =================
       DEV |  Avg Reqs @ Q
---------- | -------------
 (  8,  0) |           0.0

==================== I/O Active Period Information ====================
       DEV |     # Live      Avg. Act     Avg. !Act % Live
---------- | ---------- ------------- ------------- ------
 (  8,  0) |          5   0.000724715   1.039162389   0.09
---------- | ---------- ------------- ------------- ------
 Total Sys |          5   0.000724715   1.039162389   0.09
```

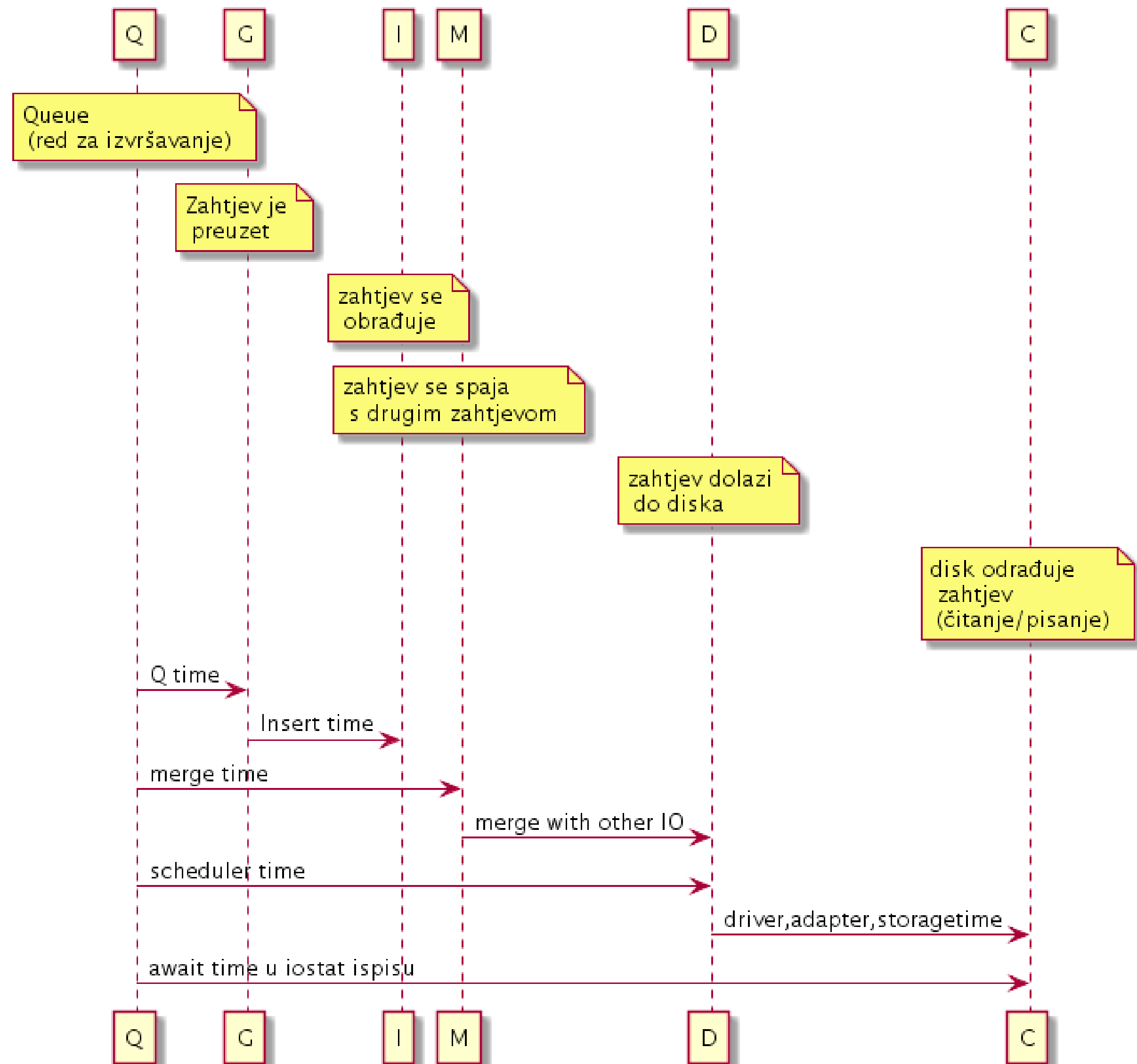
Opis (sva vremena su u milisekundama):

- `Q2Q` - vrijeme između zahtjeva prema *block layer* sloju.
- `Q2G` - koliko vremena je potrebno od kada je sa *block layer* došao do nîza za izvršavanje (*queue*) do vremena kada je zahtjev alociran (*allocated*).
- `G2I` - vrijeme od kada je zahtjev alociran do vremena kada je upit ubačen u nîz za izvršavanje unutar fizičkog diska i njegovog nîza za izvršavanje (*disk device queue*).
- `Q2M` - koliko vremena je potrebno od kada je na blok sloju zahtjev došao do nîza za izvršavanje (*queue*) do vremena kada je spojen s postojećim zahtjevom (*request merging* slučaj).
- `I2D` - vrijeme koje je potrebno od kada je zahtjev ubačen u nîz za izvršavanje unutar fizičkog diska do vremena kada ga fizički disk stvarno izvrši.

- `M2D` - vrijeme od trenutka kada je zahtjev došao do blok sloja (*block layer*) preko trenutka kada je zahtjev spojen s drugim zahtjevima (ako ih ima), pa sve do trenutka kada su svi zajedno došli do fizičkog diska, na izvršavanje operacija čitanja ili pisanja.
- `D2C` - vrijeme obrade zahtjeva od strane fizičkog diska.
- `Q2C` - ukupno vrijeme provedeno o blok sloju.

Pogledajmo logičku shemu komunikacije između slojeva diskovnog podsustava čije statistike smo dobili u prethodnom primjeru (slika 137).

**Slika 137. Opis rezultata mjerenja diskovnog podsustava.**




**Izvori informacija:** `man blkparse`,`man blktrace`,`man sekwatcher`,`man btt`.

# 15. Analiza rada linux sustava i programa

**Slijedi napredno poglavlje (15.x)!**
U ovom poglavlju obradit ćemo nekoliko alata s kojima može analizirati Linux programe, kako one pokrenute tako i izvršne programe odnosno one koje tek treba pokrenuti.

## 15.1. Linux *perf* naredba

Naredba `perf`[228],[229],[230] se koristi za analizu performansi odnosno profiliranje izvođenja određenog programa, koji testiramo. Uglavnom, na većini distribucija linuxa, potrebno je instalirati ovaj program, što ćemo napraviti sa:

```
yum install perf
```

Dakle s ovim programom možemo, čak do vrlo niske razine, pratiti izvođenje programa koji testiramo, od razine:

- *CPU* instrukcija koje CPU izvršava tijekom izvršavanja našeg (testiranog) programa: koliko je promašaja u *cache* memoriji procesora, krivog grananja na razini CPUa i slično.
- Sustav *Virtualne* memorije: razne operacije prema i od sustava virtualne memorije, a samim time i RAM memorije.
- Datotečni i diskovni sustav: operacija od i prema datotečnom i diskovnom sustavu.
- Hardverski uređaji: *blok, scsi, net, IRQ.*
- Mrežni sustav: operacije od i prema mrežnom sustavu.
- Linux *task scheduler*: operacije koje se odnose na Linux *task* scheduler.
- Linuxovih sistemskih poziva.

Dodatno je moguće pratiti i gore navedene statistike cijelog sustava ili pojedinog već pokrenutog procesa/programa, prema njegovom ***PID*** broju.

Naredba `perf` ima nekoliko osnovnih funkcionalnosti, koje pozivamo na sljedeći način:

- `perf stat` *OPCIJE* i *prekidači*: s ovom naredbom možemo pratiti željene dijelove sustava.
- `perf probe`[232] *OPCIJE* i *prekidači*: za dinamičko praćenje željene točke u programu (*tracepointa).*
- `perf  record` *OPCIJE* i *prekidači*: s ovom naredbom možemo pratiti željene dijelove sustava te sve snimiti u datoteku za kasniju detaljniju analizu.
- `perf  report` - s ovom naredbom izrađujemo izvještaj iz snimljene datoteke (od prijašnjeg primjera (`perf record`).
- `perf annotate`[231] - koristi se *disassemble* za prikaz izvršavanja izvršnog procesa/programa, kako se izvršava u *assemblerskom kôdu.*
- `perf top` - prikazuje status cijelog sustava u realnom vremenu. Poput linux naredbe `top` za procese.
- `perf bench` - koristi se za testove performansi (*benchmark*).

Prije svega, potrebno je imati *montiran* ***debugfs*** datotečni sustav.
Ako on slučajno na vašem linuxu nije montiran, to možete napraviti pomoću sljedeće naredbe:

```
mount -t debugfs nodev /sys/kernel/debug
```

Za početak, provjerimo što sve možemo pratiti na našem trenutnom Linuxu, s trenutnim kernelom:

```
perf list
```

Dobit ćemo ispis koji se proteže na cijeli nîz stranica, stoga ćemo ga filtrirati sa slijedećim nîzom naredbi, pozvanih u jednom retku:

```
perf list | awk -F: '/Tracepoint event|Software event|Hardware event|Hardware cache
event|Kernel PMU event/ { lib[$1]++ } END { for (l in lib) { printf "  %-16.16s
%d\n", l, lib[l] } }' | sort | column | more
```

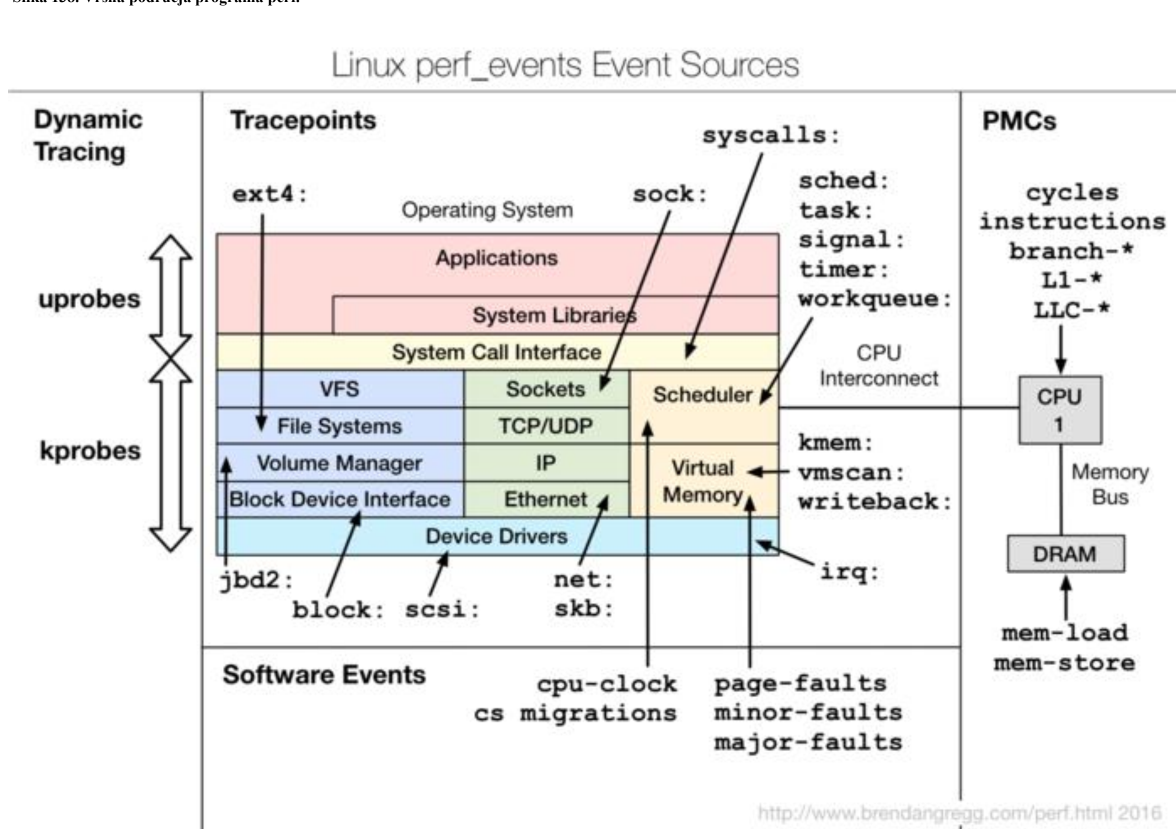
I dobit ćemo pobrojano koliko ima vršnih područja te koliko unutar svakog od njih ima podkomponenti (ovo napravite sami).



Pogledajte kratku animaciju pozivanja gore navedenog nîza naredbi.

Pogledajte i shemu iz koje ćete vidjeti vršna područja te njihovu vezu s dijelovima sustava koje želite pratiti ili analizirati s programom `perf`, kako je vidljivo na slici 138.

**Slika 138. Vršna područja programa perf.**




*Autor slike: Brendan Gregg,*
*Licenca : Attribution-ShareAlike 4.0.*

Dakle za svako od vršnih područja koje želimo pratiti, postoji cijeli nîz podkomponenti, unutar vršnog područja koje možemo pratiti. Vršna područja su označena kao ***IME*:** nakon kojega slijedi podkomponenta unutar tog područja koju želimo pratiti.
Nabrojit ćemo sâmo nekoliko vršnih područja:

- `block:` - za blok uređaje (diskove).
- `branch-misses:` - za promašaje u grananju unutar CPUa.
- `cache-misses:` - za promašaje čitanja iz CPU *cache* (L1,L2 i L3) memorije.
- `context-switches:` - broj *context switcheva.*
- `cpu-clock:` - koliko taktova CPUa je potrebno za određene operacije.
- `dTLB-load-misses:` - promašaji iz CPU ***dTLB*** memorije.
- `ext4:` - podaci vezani za *ext4* datotečni sustav.
- `iommu:` - podaci vezani za ***MMU (komponentu unutar memorijskog kontrolera)*** to jest dio *CPUa*.
- `irq:` - podaci vezani za ***IRQ*** (*interrupte*).
- `iTLB-load-misses:` - promašaji iz CPU ***dTLB*** memorije.
- `kmem:` - podaci vezani za kernel memoriju.
- `kvm:` - podaci vezani Linux ***KVM*** virtualizaciju.
- `L1-dcache-load-misses:` - promašaji iz *CPU Layer 1 dcache* memorije.
- `mem:` - podaci vezani za memoriju.
- `module:` - podaci vezani za kernel module.
- `net:` - podaci vezani za mrežu.
- `page-faults:` - greške vezane za stranice memorije (*memorijski paging).*
- `sched:` - podaci vezani za *Linux scheduler.*
- `syscalls:` - podaci vezani za sistemske pozive.
- `sock:` - podaci vezani za mrežne *sockete.*
- ...

Svako od ovih vršnih cjelina, pratimo/analiziramo s prekidačem `-e`.
U sljedećoj cjelini ćemo vidjeti i primjere kako to napraviti.

**Izvor informacija: (228),(229),(230),(231),(232), `man perf`.**

### 15.1.1. *perf stat*

Naredba `perf stat` koristi se za statičku analizu programa koji želimo analizirati. Pogledajmo primjere.

**Osnovna statička analiza izvršavanja željenog programa**
U ovom primjeru napravit ćemo statičku analizu programa `gzip` koji će u trenutku analize komprimirati našu datoteku `messages.txt` (30MB). Koristimo i prekidač `-d` koji daje malo više detalja odnosno vidimo statistike *L1* i *LLC CPU cache* memorije. Ispis ovisi o inačici i konfiguraciji kernela te sâmom računalu!.

```
perf stat -d gzip messages.txt
```

```
 227.127204   task-clock (msec)         #    0.998 CPUs utilized
          1   context-switches          #    0.004 K/sec
          0   cpu-migrations            #    0.000 K/sec
        239   page-faults               #    0.001 M/sec
529,465,702   cycles                    #    2.331 GHz                    [39.25%]
202,280,729   stalled-cycles-frontend   #   38.20% frontend cycles idle    [39.64%]
107,323,212   stalled-cycles-backend    #   20.27% backend  cycles idle    [40.80%]
936,221,666   instructions              #    1.77  insns per cycle
                                        #    0.22  stalled cycles per insn [50.89%]
117,275,237   branches                  #  516.342 M/sec                  [51.10%]
  1,308,896   branch-misses             #    1.12% of all branches         [50.90%]
176,549,757   L1-dcache-loads           #  777.317 M/sec                  [49.35%]
 15,674,171   L1-dcache-load-misses     #    8.88% of all L1-dcache hits   [19.89%]
    893,914   LLC-loads                 #    3.936 M/sec                  [19.46%]
    241,720   LLC-load-misses           #   27.04% of all LL-cache hits    [29.15%]

0.227620363 seconds time elapsed
```

Što je ovdje vidljivo[233]:

- `task-clock` - govori nam koliko je trajalo zauzeće CPUa, u *ms* (lijevi dio) te koliko je CPU jezgra ili jezgri bilo opterećeno (desni dio) izvršavanjem našeg programa.
- `context-switches` - govori koliko je *context switcheva* bilo aktivno tijekom izvršavanja našeg programa (što manje to bolje) s lijeve strane, odnosno njihov broj u sekundi za desni dio statistike. Dakle ako se *Linux task scheduler* tijekom izvršavanja našeg programa morao na kratko prebaciti na izvršavanje drugog programa pa vratiti na naš ili ako se dogodio neki signal prekida (IRQ) te se morala odraditi neka rutina vezana za signal prekida te ponovno vratiti na izvršavanje našeg programa. U graničnim slučajevima, ako naš program ima veliki broj programskih nîti ili ako se izvršava na velikom broju CPU jezgri te imamo veliki broj takvih programa na sustavu, događat će se preveliki broj *context switcheva* što ponekad nije dobro za performanse.
- `cpu-migrations` - govori je li bilo prebacivanja našeg programa između više CPU jezgri, što može biti dobro, ali i loše, ovisno o sâmom programu. Ovdje konkretno vidimo broj puta koliko je program prebacivan na izvršavanje na druge CPU jezgre.
- `page-faults` - svaki puta kada je naš program pokušao dohvatiti neke podatke iz sustava virtualne memorije, a oni nisu bili u RAM memoriji, dolazi do ovog stanja u kojem se podaci prvo moraju zapisati u RAM memoriju kako bi mogli biti dohvaćeni od strane (našeg) programa. Što manje vrijednosti ovdje, time bolje.
- `cycles` - govori koliko je CPU ciklusa (taktova) bilo potrebno da se naš program izvrši (lijevo), te na kojem je taktu CPU radio (*GHz*) i koliko je bilo opterećenje *CPUa* u postotcima (*%*).
- `stalled-cycles-frontend` i `stalled-cycles-backend` - ako se može prikazati (ovisno o konfiguraciji kernela) - govori koliko efikasno se program izvršava i stoje li na čekanju neke radnje unutar cjevovoda procesora. Drugim riječima čeka li CPU na nešto: primjerice na dohvaćanje podataka u ili iz memorije, čeka li se nešto unutar programskog kôda ili se čeka na nešto drugo poput primjerice dohvaćanja podataka s diska, mreže ili slično.
- `instructions` - govori koliko dobro se programski kôd izvršava na najnižoj razini, odnosno koliki broj CPU mikro instrukcija je CPU u stanju odraditi unutar jednog ciklusa (takta). Sve ispod **1** je loše. Međutim sve ovisi kako o samom *CPUu* i njegovom dizajnu, tako i o sâmom programu koji testiramo. Mi vidimo kako se na našem sustavu izvršava 1.77 instrukcija u jednom taktu procesora što je solidno.
- `branches` - govori koliko grananja i petlji postoji unutar programa, odnosno koliko ih je odrađeno: lijevo je njihov broj, a desno je njihov broj u sekundi (milijuna/sekundi).
- `branch-misses` - označava koliko je predikcije grananja, na razini *CPUa* bilo pogrešno, pa su se morala ponoviti. Lijevo je broj promašaja a desni dio je izražen u postotku. Sve vrijednosti do nekoliko postotaka su u redu.
- `L1-dcache-loads` i `L1-dcache-load-misses` - odnose se na to koliko se podataka učitalo ili nije dohvaćeno iz *CPU L1 cache* memorije, na što se može utjecati znatnijim optimizacijama programa.
- `LLC-loads` i `LLC-load-misses` - odnose se na *last-level cache* memoriju (obično *L3*) odnosno koliko se podataka učitalo u nju, odnosno koliko podataka nije dohvaćeno iz nje. Napomena je ista kao i za prethodna dva polja.

I na kraju imamo vrijeme koliko je bilo potrebno da se naša naredba (koju pratimo) izvrši.
Važno je razumjeti kako se u trenutku pokretanja vašeg testa, na sustavu vjerojatno pokreću i drugi programi (procesi), koji mogu utjecati na mjerenja, pa ako je moguće, zaustavite sve programe koji bi mogli utjecati na ova mjerenja.
Ovdje naročito mislimo na pokazatelje koji se tiču *CPU cache memorije* (*L1-dcache-loads*, *L1-dcache-load-misses*, *LLC-loads* i *LLC-load-misses*) te predikcije grananje unutar *CPU-a* (*branch-misses*).

**Statička analiza općih statistika za cijeli operacijski sustav**
Pogledajmo sveukupne statistike cijelog operativnog sustava, s time da ćemo statistike prikupljati samo 5 sekundi:

```
perf stat -a sleep 5

160320.347229   task-clock (msec)       #   32.057 CPUs utilized
       10,250   context-switches        #    0.064 K/sec
          284   cpu-migrations          #    0.002 K/sec
        3,602   page-faults             #    0.022 K/sec
5,015,973,413   cycles                  #    0.031 GHz                     [83.35%]
5,640,706,458   stalled-cycles-frontend #  112.45% frontend cycles idle    [83.34%]
5,060,632,995   stalled-cycles-backend  #  100.89% backend  cycles idle    [66.68%]
1,157,010,314   instructions            #    0.23  insns per cycle
                                        #    4.88  stalled cycles per insn [83.34%]
  248,416,922   branches                #    1.550 M/sec                   [83.34%]
    8,285,615   branch-misses           #    3.34% of all branches         [83.32%]
  5.001172231 seconds time elapsed
```

Na ispisu su sada vidljive opće statistike sustava, koje smo već objasnili, ali ne za specifičan program, već za cijeli sustav.

Međutim moguće je tražiti i samo specifične statistike, poput sljedećeg primjera:

```
perf stat -e cycles,instructions,cache-references,cache-misses,bus-cycles -a sleep 5
```

U ovom primjeru tražimo samo statistike iz područja: `cycles`,`instructions`,`cache-references`,`cache-misses`,`bus-cycles` s time da se statistike prikupljaju 5 sekundi. Pogledajmo sada, samo statistike cijelog sustava, ali vezane isključivo za datotečni sustav **ext4** i to sve statistike ovog datotečnog sustava:

```
perf stat -e 'ext4:*' -a sleep 5
```

**Statistike za pokrenuti proces/program prema njegovom *PID* broju.**
Sada ćemo prikupiti mrežne statistike, ali za specifičan pokrenuti program/proces. U našem slučaju se radi o programu/procesu, za koji smo saznali kako je njegov ***PID*** broj `2205`, na `10` sekundi, i to sâmo za mrežne statistike:

```
perf stat -e 'net:*' -p 2205 -a sleep 10
```

Ovdje je vidljivo koliko podataka je poslano i primljeno, a koliko se nalazilo u redu odnosno u nîzu za dohvaćanje (*net_dev_queue*) i slično.

**Izvori informacija: (233),(643),(644).**

### 15.1.2. *perf record*

Naredba `perf record` koristi se za statičku analizu programa koji želimo snimiti u datoteku zbog kasnije analize.
Kako bi se kernel simboli mogli prikazati, potrebno je promijeniti sljedeću sistemsku postavku:

```
echo 0 > /proc/sys/kernel/kptr_restrict
```

U novijim inačicama Linuxa, ova vrijednost je već postavljena na nula (0). Sada pogledajmo primjere.

**Praćenje određene vrste statistika, za sve programe/procese**
Pratimo pristup blok uređajima (diskovnom podsustavu), na pet (5) sekundi te sve snimimo u datoteku zbog dalje analize:

```
perf record -e block:* -a sleep 5
```

Moguće je i ne definirati vrijeme snimanja (`sleep 5`), ali će se tada statistike snimati, sve dok ne prekinemo proces snimanja sa kombinacijom tipki `CTRL c`.
Analitika (statistika) se snima u binarnu datoteku imena `perf.data` u trenutni direktorij u kojem se nalazimo.
Naredba `perf report` koristi se za izradu analitike iz datoteke `perf.data`.
Pogledajmo što nam kaže analitika u našem primjeru:

```
perf report
```

U našem slučaju, dobit ćemo ovakav ispis:

```
Available samples
0 block:block_touch_buffer
129 block:block_dirty_buffer
0 block:block_rq_abort
0 block:block_rq_requeue
205 block:block_rq_complete
205 block:block_rq_insert
205 block:block_rq_issue
0 block:block_bio_bounce
0 block:block_bio_complete
8 block:block_bio_backmerge
0 block:block_bio_frontmerge
213 block:block_bio_queue
205 block:block_getrq
0 block:block_sleeprq
2 block:block_plug
2 block:block_unplug
0 block:block_split
213 block:block_bio_remap
0 block:block_rq_remap
```

Važno je razumjeti, kako se u svakom redu nalaze statistike, za svaku vrstu pristupa *BLOK* uređajima.

Ako sa kursorskim tipkama odaberemo recimo `block:block_rq_complete` i stisnemo tipku **ENTER**, dobit ćemo statistike za tu vrstu upita prema disku. Pogledajmo i ovu statistiku:

```
Samples: 205  of event 'block:block_rq_complete', Event count (approx.): 205
89.76%  kworker/u98:1  [kernel.kallsyms]  [k] blk_update_request
 2.44%        swap
per  [kernel.kallsyms]  [k] blk_update_request
 1.95%   kworker/19:2  [kernel.kallsyms]  [k] blk_update_request
 1.46%   kworker/17:3  [kernel.kallsyms]  [k] blk_update_request
 0.98%   ksoftirqd/22  [kernel.kallsyms]  [k] blk_update_request
 0.98%   kworker/18:2  [kernel.kallsyms]  [k] blk_update_request
 0.98%   kworker/20:0  [kernel.kallsyms]  [k] blk_update_request
 0.98%   kworker/21:2  [kernel.kallsyms]  [k] blk_update_request
 0.49%   kworker/22:2  [kernel.kallsyms]  [k] blk_update_request
```

Na ovom ispisu vidimo kako je proces `kworker` zauzimao najviše CPU vremena za operaciju ***block_rq_complete*** prema diskovnom podsustavu. Naime ovaj program (proces) je zadužen za mnoge funkcionalnosti kernela. Proces `kworker` u nazivu ima identifikatore, poput `/19:2`, koji se inače nazivaju, prema principu: **`kworker/%u:%d%s (cpu, id, priority)`**. Dakle ime ovog programa (procesa) je, kako je i vidljivo: `kworker/19:2`.

**Ispis prikaza izvještaja**

Za sve primjere upotrebe naredbe `perf record` moguće je koristiti i drugu naredbu (`perf timechart`) koja također snima statistike u datoteku, ali u formatu koji se onda može eksportirati u vektorsku sliku (*SVG* format). Na gornjem primjeru, napravit ćemo istu statistiku te ju eksportirati u *SVG* datoteku koju možemo kasnije otvoriti s bilo kojim web preglednikom (*Firefox*, *Chrome*, …). Pogledajte kako (ovisno o inačici *perf* programa) to napraviti s naredbom `perf timechart`:

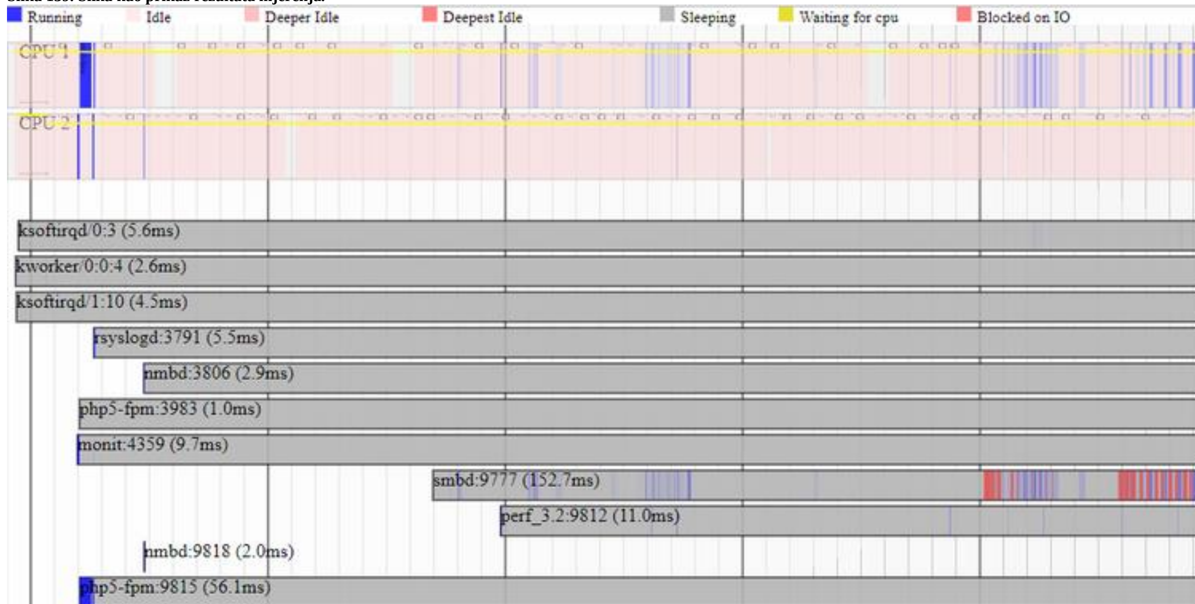
```
perf timechart record -e block:* -a sleep 5
```

Dobit ćemo statistike zapisane u datoteci. Potom ih trebamo obraditi i konvertirati u *SVG* datoteku, pomoću sljedeće naredbe:

```
perf timechart
```

I dobivamo datoteku imena: `output.svg` koju možemo kopirati na svoje računalo i otvoriti ju.
Pogledajmo kako ona izgleda u našem primjeru, na slici 139.

**Slika 139. Slika kao prikaz rezultata mjerenja.**




Ovdje na lijep vizualan način vidimo statistike diskovnog sustava na *blok* razini, u trenutku kada smo nešto kopirali s diska.
Na nekim inačicama programa *perf* imamo sâmo nekoliko prekidača tijekom snimanja, ako se koristi:
`perf timechart record`:

- `-P` - samo snima podatke vezane za *CPU power*.
- `-p` ***PID*** - snima podatke sâmo od procesa koji pratimo (prema njegovom ***PID*** broju).
- `-T` - samo podaci od procesa/programa.
- `-I` - samo ulazno/izlazni (I/O) podaci.

Moguće je snimati sve događaje (Engl. *evente*) i to ovako:

```
perf timechart record
```

Napomena: Snimanje prekidamo s tipkama `CTRL c`.

Sada opet možemo izraditi *SVG* vektorsku sliku (fotografiju) kao rezultat snimanja (analize rada):

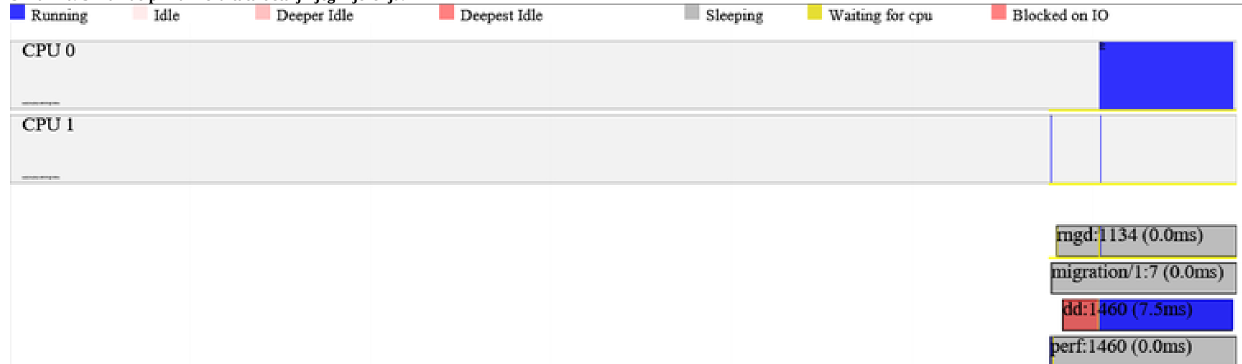
```
perf timechart
```

Moguće je snimiti i statistike rada nekog programa koji pokrećemo, za koji opet možemo izraditi *SVG sliku*.

```
perf timechart record dd if=/dev/zero of=/root/test/file.dd bs=1M count=10
```

U gornjem primjeru, pozivamo naredbu: `dd if=/dev/zero of=/root/test/file.dd bs=1M count=10` koja kreira datoteku imena: `/root/test/file.dd` veličine 10 MB.
U ovom koraku možemo izraditi izvještaj:

```
perf timechart
```
I dobili smo ga u datoteci `output.svg`.
Pogledajmo kako ovaj izvještaj izgleda kao slika (slika 140):

Slika 140. Slika kao prikaz rezultata detaljnijeg mjerenja.



Na slici 140 vidljivo je kako je naša naredba za kopiranje (`dd`) prvo čekala na diskovni podsustav: u crvenom je (• ***Blocked on IO***), a potom je krenula sa zapisivanjem: u plavom je (• *Running*). Moguće je odrediti i dužinu snimanja s opcijom `sleep` kao u primjeru dolje u kojemu definiramo dužinu snimanja od pet (5) sekundi:
```
perf timechart record sleep 5
```

**Praćenje svih statistika, za sve pokrenute programe/procese**
U ovom slučaju uvesti ćemo još jedan napredni prekidač `-F 99` koji označava da će se uzorkovanje umjesto svakih 100Hz (100 puta u sekundi) raditi na 99Hz, kako bi se potencijalno drugim intenzivnim aktivnostima smanjio utjecaj na uzorkovanje statistika. Stoga pokrenimo naredbu `perf record` na sljedeći način:
```
perf record -F 99 -a -g -- sleep 5
```

Sada možemo analizirati dobivenu datoteku s naredbom `perf report`:
```
perf report
```
Dobit ćemo nešto poput ovog ispisa:
```
Samples: 1K of event 'cycles', Event count (approx.): 14246501253
+   54.92%    0.00%           java  libpthread-2.17.so    [.] start_thread
+   30.24%    0.00%           java  libjvm.so             [.] 0x0000000000872be2
+   29.65%    0.00%           java  libjvm.so             [.] 0x00000000009bfee2
+   24.66%    0.00%           java  libjvm.so             [.] 0x0000000000480450
+   24.55%    0.00%           java  libjvm.so             [.] 0x00000000006420ca
+   23.54%    0.00%           java  libjvm.so             [.] 0x000000000047f5e6
+   23.11%    0.03%        swapper  [kernel.kallsyms]     [k] cpu_startup_entry
+   22.90%    0.00%        swapper  [kernel.kallsyms]     [k] start_secondary
+   20.72%    0.00%        swapper  [kernel.kallsyms]     [k] arch_cpu_idle
+   20.64%    0.00%        swapper  [kernel.kallsyms]     [k] cpuidle_idle_call
```
Kursorskim tipkama možemo odabrati željeni proces, ući u njega (s **ENTER**) te kada uđemo u njega, možemo ponovno ući u njegove nîti (*threadove*) i analizirati ih, te potom ući još dublje u pojedine funkcije i vidjeti njihove statistike.
Postoji i drugi način ispisa odnosno odabira načina prikaza izvještaja, ako `perf report` pozovemo na sljedeći način:
```
perf report --stdio
```
Dobit ćemo ispis svih programa/procesa u obliku stabla što je ponekad korisno, ali je uglavnom preveliko za pregledavanje.
Mi smo ispis maksimalno skratili:
```
#
# Children      Self          Command          Shared Object     Symbol
# ........  ........  .................  ..............  ......
#
    54.92%    0.00%             java      libpthread-2.17.so  [.] start_thread
                        |
                        --- start_thread
    30.24%    0.00%             java      libjvm.so           [.] 0x0000000000872be2
                        |
                        |--21.42%-- 0x7fc8471febe2
                        |           start_thread
                        |--20.92%-- 0x7f3917061be2
                        |           start_thread

    29.65%    0.00%             java      libjvm.so           [.] 0x00000000009bfee2
                        |
                        |--21.31%-- 0x7fc84734bee2
                        |           0x7fc8471febe2
                        |           start_thread
                        |--20.90%-- 0x7f39171aeee2
                        |           0x7f3917061be2
```
I ovdje možemo snimati statistike pojedinog procesa, prema njegovom ***PID*** broju, pomoću prekidača `-p` ***PID***.

### 15.1.3. *perf top*

Naredba perf top koristi se za prikaz (analitiku) statistika cijelog sustava u realnom vremenu, poput naredbe top, ali sa svim statistikama opterećenja sustava na vrlo niskoj razini, odnosno prema raznim pod komponentama sustava. Standardno, moguće je snimanje cijelog sustava:

```
perf top
```

Tada ćemo vidjeti sve programe/procese i sve sistemske komponente, poput ovoga na ispisu dolje (skratili smo ispis):

```
Events: 11K cycles
  3.85%  [tg3]                        [k] tg3_poll_work
  3.39%  [ip_tables]                  [k] ipt_do_table
  2.50%  [tg3]                        [k] tg3_read32
  2.02%  [kernel]                     [k] do_raw_spin_lock
  1.94%  [kernel]                     [k] nf_iterate
  1.78%  [kernel]                     [k] __netif_receive_skb
  1.78%  [kernel]                     [k] tcp_ack
  1.58%  [nf_conntrack]               [k] tcp_packet
  1.52%  [kernel]                     [k] native_read_tsc
  1.39%  [kernel]                     [k] delay_tsc
  1.21%  [kernel]                     [k] vlan_do_receive
  1.17%  [e1000e]                     [k] e1000_irq_enable
  1.12%  [nf_conntrack]               [k] nf_conntrack_in
  1.00%  [kernel]                     [k] ip_route_input_common
  0.89%  [kernel]                     [k] __pskb_trim_head
  0.88%  [kernel]                     [k] iowrite8
  0.85%  [kernel]                     [k] arch_local_irq_restore
  0.81%  [kernel]                     [k] kmem_cache_alloc
  0.77%  [bonding]                    [k] bond_handle_frame
```

Na primjeru od gore, radio se prijenos podataka preko mreže, pa su među prvim najopterećenijim sustavima; drugi stupac [ ] :

- tg3 - ovo je mrežna kartica (tj. naziv njenog *kernel modula*) koja radi dohvaćanje mrežnih paketa *(polling)*.
- ip_tables - ovo je linux vatrozid (*firewall*).
- kernel - ovdje imamo razne funkcije kernela, uglavnom vezane za mrežu.

Dakle ovo je normalno ponašanje, kada se preko 1Gbps mrežnog sučelja kopiraju podaci iz memorije (RAM) preko mreže. Dodatno, i u ovom pregledu moguće je s kursorskim tipkama ući u pojedini proces te ulaziti sve dublje i pratiti detalje, poput sistemskih poziva i funkcija koje poziva određena komponenta ili nît vršnog procesa. Osim ovakvog sveukupnog praćenja cijelog sustava, moguće je u realnom vremenu pratiti i sâmo željene funkcije klasično s prekidačem -e (iz prijašnjih perf opcija) kao i željenu naredbu ili proces (prekidač -p ***PID***). Dodatno, moguće je i raznoliko filtriranje i sortiranje ispisa.
Za više detalja o opcijama i parametrima upišite:

```
perf top --help
```

Jedan od korisnih brzinskih primjera je jednostavno filtriranje prema procesu koji zauzima najviše *CPU* vremena, uz prikaz samo imena tog procesa i simbola/objekata koje koristi:

```
perf top -F 49 -ns comm,dso
```

**Za više primjera, pogledajte: (228),(234), man perf.**

### 15.1.4. *perf* i detaljnija analiza programskog koda

U slučajevima kada nam je potrebna detaljnija analiza programa, sve do razine *disassemblera*, odnosno kada želimo vidjeti koji *assemblerski* programski kôd je izvršavala koja funkcija programa koji analiziramo, to možemo postići na dva načina:

- Upotrebom već poznate naredbe perf report ili
- Upotrebom perf annotate naredbe.

U oba slučaja prvo moramo snimiti analitiku pomoću naredbe perf record s opcijama koje nam već trebaju jer se *disassembling* radi iz snimljene analitike, dakle standardno iz datoteke: perf.data.
Uzmimo za primjer snimanje statistika za kreiranje datoteke veličine 10MB pune nula (0), pomoću naredbe dd s naredbom:

```
perf record dd if=/dev/zero of=/root/test/file.dd bs=1M count=10
```

Nakon toga ćemo dobiti izlaznu datoteku u kojoj se snimalo stanje diskovnog podsustava na *blok* razini za 5 sekundi.
Sada možemo pokrenuti analizu upotrebom naredbe perf report:

```
perf report

Samples: 37  of event 'cycles', Event count (approx.): 25266778
  30.82%  dd  [kernel.kallsyms]  [k] copy_user_enhanced_fast_string
  12.14%  dd  [kernel.kallsyms]  [k] __clear_user
   9.20%  dd  [kernel.kallsyms]  [k] mark_page_accessed
   6.34%  dd  [kernel.kallsyms]  [k] kmem_cache_alloc
   5.92%  dd  [kernel.kallsyms]  [k] _raw_spin_lock
   4.62%  dd  [kernel.kallsyms]  [k] __mem_cgroup_commit_charge
   3.29%  dd  [kernel.kallsyms]  [k] __list_del_entry
   3.22%  dd  [kernel.kallsyms]  [k] xfs_bmap_search_multi_extents
   3.21%  dd  [kernel.kallsyms]  [k] __lru_cache_add
```

Potom možemo ući u pojedinu funkciju poput ove prve (`copy_user_enhanced_fast_string`) za koju imamo 30.82% uzoraka od ukupnog CPU vremena. Odaberimo ju s kursorskim tipkama te stisnimo slovo **a** što znači ***annotate***.
Sada ćemo dobiti programski kôd označene funkcije (`copy_user_enhanced_fast_string`) u *assembleru*.
Konkretno, dobit ćemo nešto poput:

```
       │    Disassembly of section load2:
       │
       │    ffffffff81325e40 <load2+0x325e40>:
       │      data16 xchg %ax,%ax
       │      and    %edx,%edx
       │    ↓ je     b
       │      mov    %edx,%ecx
100.00 │      rep    movsb %ds:(%rsi),%es:(%rdi)
       │ b:   xor    %eax,%eax
       │      data16 xchg %ax,%ax
       │    ← retq
```

Lijevo je ponovno postotak uzoraka vremena CPUa odnosno relativno CPU vrijeme, a desno je programski kôd u *assembleru*, odnosno instrukcije u *assembleru* koje su se izvršile u našoj funkciji koju konkretno i promatramo: *copy_user_enhanced_fast_string*. Sličnu stvar možemo dobit i s naredbom `perf annotate` koju sada nećemo objašnjavati.

---

# 15.2. Linux *strace* naredba

Naredba `strace`(226),(227) koristi se za praćenje sistemskih poziva i signala koje koristi program koji želimo analizirati pomoću ove naredbe. Naime svaki program u radu poziva cijeli nîz sistemskih poziva te koristi mnoge sistemske signale koje možemo vidjeti pomoću naredbe `strace`. Pogledajmo neke od primjera iz kojih će vam biti jasnija upotreba i namjena ove naredbe.
Provjerimo koje sve sistemske pozive, poziva naredba `ls` prilikom izlistavanja sadržaja direktorija (mapa):

```
strace ls
```

Dobit ćemo nešto poput:

```
execve("/bin/ls", ["ls"], [/* 22 vars */])
futex(0xbfcceaa4, FUTEX_WAKE_PRIVATE, 1) = 0
futex(0xbfcceaa4, FUTEX_WAIT_BITSET_PRIVATE|FUTEX_CLOCK_REALTIME, 1, NULL, bfcceab4) = -1
EAGAIN (Resource temporarily unavailable)
rt_sigaction(SIGRTMIN, {0xe5d7d0, [], SA_SIGINFO}, NULL, 8) = 0
rt_sigaction(SIGRT_1, {0xe5dcd0, [], SA_RESTART|SA_SIGINFO}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
getrlimit(RLIMIT_STACK, {rlim_cur=10240*1024, rlim_max=RLIM_INFINITY}) = 0
uname({sys="Linux", node="localhost.localdomain", ...}) = 0
statfs64("/selinux", 84, {f_type="EXT2_SUPER_MAGIC", f_bsize=4096, f_blocks=785908,
f_bfree=426542, f_bavail=385787,
```

Ovdje je vidljivo puno detalja:

- U prvom redu se vidi koja se izvršna datoteka poziva, a ona je u našem slučaju `/bin/ls`.
- Nadalje se vide svi sistemski pozivi koji se pozivaju, kao i datoteke; poput biblioteka i drugih koje se čitaju i otvaraju

U slučaju kada je primjerice naredba postavljena da koristi takozvane [Linux (*capabilities*) mogućnosti](), tada bi trenutak u kojem se one čitaju (`capget`) ili postavljaju (`capset`) bio vidljiv kao:

```
capget({version=_LINUX_CAPABILITY_VERSION_3,pid=0},{effective=1<<CAP_CHOWN|1 . . .
capset({version=_LINUX_CAPABILITY_VERSION_3,pid=0},{effective=0,
permitted=1<<CAP_NET_ADMIN| 1<<CAP_NET_RAW, inheritable=0}) = 0a
```

Za detalje pogledajte poglavlje: **4.4.3. To nije sve: mogućnosti izvršavanja datoteka i programa (*capabilities*).**

**Upit za specifičnim sistemskim pozivom**
Moguće je prema programu koji testiramo poslati i upit, koristi li on specifični sistemski poziv poput `open` poziva koji znači: *„daj mi sve sistemske zahtjeve u kojima se traži otvaranje (**open**) datoteka“*. Pogledajmo već navedenu naredbu `ls`, ali ovako:

```
strace -e open ls
```

Dobit ćemo ispis, koje sve datoteke je naša naredba `ls` otvorila, prilikom svog pokretanja odnosno izvršavanja:

```
open("/etc/ld.so.cache", O_RDONLY)      = 3
open("/lib/libselinux.so.1", O_RDONLY)  = 3
open("/lib/librt.so.1", O_RDONLY)       = 3
open("/lib/libcap.so.2", O_RDONLY)      = 3
open("/lib/libacl.so.1", O_RDONLY)      = 3
open("/lib/libc.so.6", O_RDONLY)        = 3
open("/lib/libdl.so.2", O_RDONLY)       = 3
open("/lib/libpthread.so.0", O_RDONLY)  = 3
open("/lib/libattr.so.1", O_RDONLY)     = 3
open("/proc/filesystems", O_RDONLY|O_LARGEFILE) = 3
open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = 3
open(".", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_CLOEXEC) = 3
```

Dakle vidimo koje sve datoteke naredba `ls` otvara tijekom pozivanja, ako smo ju pozvali samo s `ls`.
Možemo testirati i koje sve datoteke ova naredba otvara, ako ju pozivamo s nekim prekidačem poput `ls -alh`:

```
strace -e open ls -alh
```

Sada bi dobili i druge otvorene datoteke, jer našu naredbu pozivamo s drugačijim parametrima (ili opcijama).

Pogledajmo jedan sličan primjer u kojem nas zanima, kada pozovemo program `netstat` bez ikakvih opcija i parametara, koje sve on datoteke čita iz posebnog direktorija */proc*.

```
strace -e open netstat 2>&1 > /dev/null | grep /proc

open("/proc/net/tcp", O_RDONLY)          = 3
open("/proc/net/tcp6", O_RDONLY)         = 3
open("/proc/net/udp", O_RDONLY)          = 3
open("/proc/net/udp6", O_RDONLY)         = 3
```

Ispis smo dodatno filtrirali i skratili, s preusmjeravanjem (`2>&1 > /dev/null`) te filtriranjem s naredbom `grep`.
Vidljivo je kako se pozivanje naredbe `netstat` bez opcija i prekidača čita samo nekoliko datoteka iz */proc* direktorija:

- `/proc/net/tcp` - sadrži statistike o TCP prometu.
- `/proc/net/udp` - sadrži statistike o UDP prometu.
- ...

Drugi korisni prekidači naredbe `strace` su:

- `-e` - navedi koje sistemske pozive želimo pratiti.
  - `open` - prikaži koje su otvorene datoteke (koje je program otvorio).
  - `write` - prikaži datoteke koje su u stanju zapisivanja.
  - `network` - za mrežne konekcije programa.
- `-f` - prati i sve pôd procese trenutnog procesa (i one procese koje pokreće naš proces).
- `-o` *DATOTEKA.txt* - ispiši sve u datoteku imena *DATOTEKA.txt*.
- `-p` ***PID*** - prati sistemske pozive za već pokrenuti Linux proces, **PID** (*Process ID*) brojem ***PID.***
- `-t` - ispiši i točno vrijeme u kojem je pokrenut koji sistemski poziv.
- `-r` - ispiši relativno vrijeme u kojem je pokrenut (i koji) sistemski poziv.
- `-s` - povećaj veličinu *stringa* (podataka) koji će biti prikazani.
- `-T` - ispiši vrijeme koliko je bilo potrebno da se izvrši određeni sistemski poziv (razlike u vremenu od kada je sistemski poziv pozvan, do vremena kada je odrađen).
- `-c` - ispiši statistički izvještaj o sistemskim pozivima.

Pogledajmo primjer sa statističkim izvještajem, za svaku pojedinu vrstu sistemskih poziva, za pokretanje naredbe `ls`:

```
strace -c ls
```

Dobit ćemo nešto poput:

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0        10           read
  0.00    0.000000           0         2           write
  0.00    0.000000           0        12           open
  0.00    0.000000           0        14           close
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         2 access
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         1           rt_sigprocmask
  0.00    0.000000           0         1           getrlimit
  0.00    0.000000           0        26           mmap2
  0.00    0.000000           0        12           fstat64
  0.00    0.000000           0         2           getdents64
  0.00    0.000000           0         1           fcntl64
  0.00    0.000000           0         2         1 futex
  0.00    0.000000           0         1           set_thread_area
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   109         3 total
```

Ovdje za svaku vrstu sistemskog poziva vidimo koliko sistemskog vremena troši, koliko je bilo pojedinih sistemskih poziva (*calls*), te koliko je bilo grešaka, i to za svaki od tih poziva (*errors*) kada se pozove/pokrene naredba `ls`.

**Praćenje mrežnih aktivnosti procesa**

U primjeru koji slijedi pratit ćemo proces ***PID***-a `4758`, što je naš **http** poslužitelj. Pokrenuli smo `pidof httpd` te odabrali jedan od ***PID*** brojeva *httpd* procesa, koji ćemo sada analizirati (prema ***PID*** broju).

```
strace -f -e network -p 4758  -s 10000
```

Pošto smo povećali veličinu podataka koji se prikazuju unutar svake linije na *10.000* dobiti ćemo više podataka koji su poslani mrežom, poput (skratili smo ispis):

```
Process 4758 attached
accept4(3, {sa_family=AF_INET, sin_port=htons(57743),
sin_addr=inet_addr("192.168.100.182")}, [16], SOCK_CLOEXEC) = 9
getsockname(9, {sa_family=AF_INET, sin_port=htons(80),
sin_addr=inet_addr("192.168.100.254")}, [16]) = 0
```

Napomena: u ovom primjeru je `192.168.100.254` adresa poslužitelja (*HTTP*) a `192.168.100.182` je adresa našeg računala koje se spaja na web stranicu poslužitelja, s web preglednikom.

**Izvor informacija:** (226),(227), **`man strace`**.

## 15.3. Linux *ltrace* naredba

Naredba `ltrace`[235],[236],[237] koristi se za pronalaženje funkcija iz sistemskih i drugih (vanjskih) biblioteka koje koristi program koji želimo analizirati pomoću ove naredbe. Naime svaki program obično koristi cijeli nîz raznih funkcija čiju listu možemo vidjeti pomoću naredbe `ltrace`. Pogledajmo koje sve funkcije koristi naredba `ls` čija lokacije je: `/usr/bin/ls`:

```
ltrace /usr/bin/ls
```

Dobit ćemo cijeli nîz funkcija koje ona koristi:

```
write_unlocked("proc-net-test.sh", 1, 16, 0x7fd698079400)      = 16
__errno_location()                                              = 0x7fd6988c4690
__ctype_get_mb_cur_max()                                        = 6
fwrite_unlocked("test.dd.bin", 1, 11, 0x7fd698079400)           = 11
__errno_location()                                              = 0x7fd6988c4690
__ctype_get_mb_cur_max()                                        = 6
fwrite_unlocked("vmlinuz-3.10.0-514.21.2.el7.x86_"..., 1, 34, 0x7fd698079400) = 34
__errno_location()                                              = 0x7fd6988c4690
__ctype_get_mb_cur_max()                                        = 6
fwrite_unlocked("work", 1, 4, 0x7fd698079400)                   = 4
free(0x194ac00)                                                 = <void>
free(nil)                                                       = <void>
free(0x194abd0)                                                 = <void>
exit(0 <unfinished ...>
__fpending(0x7fd698079400, 0, 64, 0x7fd698079eb0)               = 223
fileno(0x7fd698079400)                                          = 1
__freading(0x7fd698079400, 0, 64, 0x7fd698079eb0)               = 0
...
```

Također je moguće vidjeti koje sve funkcije iz vanjskih biblioteka, koristi već pokrenuti program tj. proces prema njegovom ***PID*** broju. Dakle ovdje ćemo vidjeti pozivanje funkcija u realnom vremenu kako ih program koji pratimo i poziva.

Pogledajmo primjer u kojem pratimo koje sve funkcija poziva pokrenuti program s ***PID** brojem*: `43521`

```
ltrace -p 43521
```

Praćenje prekidamo sa kombinacijom tipki `CTRL` `c`.

**Izvor informacija:** (235),(236),(237), **`man ltrace`**.

## 15.4. Linux *dstat* naredba

Naredba `dstat` nije zamišljena za analizu rada programa već cijelog sustava, a ona kombinira razne korisne statistike koje nam daju naredbe `vmstat`, `netstat`, `iostat`, `mpstat` i `ifstat`. Naime u slučajevima kada želimo pratiti opterećenje cijelog sustava: od CPUa, mreže, softverskih i hardverskih signala prekida i diskovnog podsustava, sve to možemo pratiti s ovom naredbom. Međutim ovu naredbu je obično potrebno prvo instalirati, što ćemo i napraviti:

```
yum install dstat
```

Ako sâmo pokrenemo ovu naredbu bez prekidača dobiti ćemo sumariziranu sveopću statistiku rada sustava. Mi ćemo ju ipak pokrenuti s opcijama `2 10` koje znače kako će se statistike prikupljati svake dvije sekunde (`2`) i to deset puta za redom:

```
dstat 2 10

----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw
                       |           |           |           |
  4   1  96   0   0   0| 150k  215k|   0     0 |  60B  994B| 16k   24k
  9   1  90   0   0   0|   0     0 | 126k  114k|   0     0 | 22k   33k
  7   1  92   0   0   0|   0     0 | 126k  116k|   0     0 | 23k   26k
  7   1  93   0   0   0|   0    26k| 127k  114k|   0     0 | 25k   25k
  7   1  93   0   0   0|   0     0 | 126k  116k|   0     0 | 21k   25k
  7   1  92   0   0   0|   0   106k| 126k  114k|   0     0 | 27k   25k
  7   1  92   0   0   0|   0     0 | 126k  116k|   0     0 | 25k   25k
  7   1  92   0   0   0|   0     0 | 126k  114k|   0     0 | 23k   25k
  7   1  92   0   0   0|   0    38k| 126k  115k|   0     0 | 24k   25k
  7   1  92   0   0   0|   0     0 | 126k  114k|   0     0 | 27k   25k
  7   1  92   0   0   0|   0   120k| 127k  116k|   0     0 | 23k   25k
```

Statistika ispisana na prethodnoj stranici je podijeljena u pet osnovni stupaca, koje ćemo objasnit redom:

- `-total-cpu-usage-` nam daje statistike za CPU (postotak):
  - `usr` – označava CPU zauzeće za korisničke (normalne programe).
  - `sys` – označava CPU zauzeće za sistemske programe.
  - `idl` – označava slobodne CPU resurse.
  - `wai` – označava postotak vremena CPUa na čekanje na disk (*I/O*).
  - `hiq` – označava postotak vremena CPUa na čekanje na hardverske signale prekida (*IRQ*).
  - `siq` – označava postotak vremena CPUa na čekanja na softverske signale prekida (*soft IRQ*).

- `-dsk/total-` nam daje statistike za diskovni podsustav:
  - `read` – označava čitanje podataka s diska (k=kB) u sekundi.
  - `writ` – označava zapisivanje podataka na disk (k=kB) u sekundi.

- `-net/total-` nam daje statistike za mrežni podsustav:
  - `recv` – označava primanje podataka s mreže.
  - `send` – označava slanje podataka na mrežu.

- `-paging-` nam daje statistike za sustav virtualne memorije; konkretno za upotrebu ***swap*** diska:
  - `in` – označava snimanje na *swap* disk odnosno particiju.
  - `out` – označava čitanje sa *swap* diska odnosno particije.

- `-system-` nam daje statistike za sustav:
  - `int` – označava broj softverskih signala prekida u sekundi (*soft IRQ*).
  - `csw` – označava broj *context switch* operacija (prebacivanja između procesa ili procesnih nîti) u sekundi.

Ovu naredbu možemo i pokretati i sâmo s prekidačima s kojima ograničavamo što želimo pratiti:

- `-c` označava sâmo CPU statistike.
- `-d` označava sâmo statistike za diskovni podsustav.
- `-n` označava sâmo statistike za mrežni podsustav.
  - -N označava točno definirano mrežno sučelje (pr. `-n -N eth0`) koje želimo pratiti.
- `-g` označava sâmo statistike za *swap* particiju odnosno njenu uporabu.
- `-y` označava sâmo statistike za sustav:
  - `--aio` - uključujući i asinkrone diskovne (*I/O*) operacije.
  - `--fs` - uključuje i statistike za datotečni sustav (broj otvorenih datoteka, *inode*).
  - `--socket` - uključuje i statistike za sockete.
  - `--tcp` - uključuje i statistike za TCP stanja mrežnih konekcija.
  - `--udp` - uključuje i statistike za UDP stanja mrežnih konekcija.
  - `--vm` - uključuje i statistike sustava virtualne memorije.
  - …
- `-i` označava sâmo statistike za signale prekida (hardverski *IRQ*)
- `-m` označava sâmo statistike za sustav virtualne memorije.
- `--output` označava kako želimo ove statistike snimiti u datoteku (pr. `--output /var/log/dstat.out.csv`)
- … moguće je koristiti i module za dodatne željene statistike.



Pogledajte i sljedeće naredbe odnosno njihovu uporabu, koja je opisana u poglavljima:

**10.7.2.3.2 Naredba top.**
**10.7.2.3.5 Naredba vmstat za CPU.**
**10.7.2.3.6 Naredba mpstat za CPU.**
**14.2 Praćenje performansi I/O sustava.**
**25.7.4 Naredba *netstat*.**
**25.7.5 Naredba *nstat*.**

**Izvori informacija: (573),(574),(575),(576), `man dstat`,`man 7 aio`.**

# 16. Kernel Dump/Crashdump/core dump

**Slijedi napredno poglavlje (16.x.)!**
U određenim situacijama može doći do drastičnih problema u radu Linux sustava, koje rezultiraju fatalnim greškama i problemima u radu Linux kernela te eventualnog blokiranja cijelog računala. Srećom Linux ima mogućnost riješiti ove probleme tako da nam omogućava snimanje stanja cijele (RAM) memorije koje je snimljeno u trenutku navedenih grešaka te pohranom iste na lokalni disk ili preko mreže na neko drugo računalo. Takvo stanje memorije se naknadno može analizirati te otkriti uzrok greške u radu sustava, koja je uzrokovala ove probleme.
**Upoznajmo se prvo s pojmom *Kernel panic.***
*Kernel panic* je definirana radnja koju pokreće operativni sustav u trenutku kada detektira interne fatalne greške od kojih se ne može oporaviti. Ovaj pojam se koristi kako za Linuxe tako i za sve *UNIXoidne* operativne sustave. Ekvivalent ovakvim greškama iz *Windows* svijeta je takozvani *Stop Error* znan i kao *Blue Screen of Death (plavi ekran smrti)*.
U slučajevima kada se Linux kernel sruši u slučaju neke fatalne greške sustava `kdump` servis je u mogućnosti sačuvati konzistenciju sustava podizanjem drugog posebnog Linux kernela. Ovaj "posebni" Linux kernel se obično naziva: **dump-capture kernel**, a koristi se za snimanje stanja RAM memorije od kernela koji se srušio. Ovo snimanje stanja memorije se zove *Memory dump*. S obzirom na činjenicu da se u trenutku rušenja primarnoga Linux kernela nešto fatalno dogodilo, ne bi bilo pametno pokušavati snimiti stanje memorije korištenjem istog tog kernela jer je upitno kako bi to završilo i bi li se uopće takva operacija mogla izvršiti, jer se isti kernel i srušio zbog nekog većeg problema. Naime `kdump` je funkcionalnost Linux kernela koja kreira *Crash Dump* datoteku u slučaju kada se kernel sruši ili zablokira, uslijed neke fatalne greške. Kada je ova funkcija pokrenuta, `kdump` izvozi (eksportira) sliku memorije (Engl. *Memory image*) na medij prema izboru, i to automatski:

- Na datotečni sustav na lokalnom tvrdom disku u datoteku `/proc/vmcore` ili u `/var/crash/XY/vmcore`
- Ili na neki drugi sustav (računalo), dostupno preko mreže ili mrežnog dijeljenog datotečnog sustava (pr. ***NFS-a***).

U svakom slučaju ova slika stanja memorije se izvozi (eksportira) kao "***Executable and Linkable Format***" (*ELF*) objekt, te se naknadno može analizirati. Slika odnosno kopija memorije koju izrađuje `kdump` se zove `vmcore`.
Dakle naš `kdump` koristi metodu dvostrukog kernela koji u radu koristi `kexec` metodu za pokretanje našeg "*dump-capture kernel*"-a isti tren kada se dogodi neka fatalna greška na trenutno aktivnom (primarnom) linux kernelu. Naime `kexec` ima mogućnost pokretanja novog (ovog posebnog) Linux kernela preko postojećega, koji se srušio i to bez potrebe da se to radi na standardan način kao tijekom pokretanja cijelog sustava učitavanjem *Boot loadera* i svih standardnih procedura od strane *BIOSa* ili *UEFIja*. Učitavanje novog kernela preko postojećeg znači da se postojeći kernel koji se srušio ne dira niti se diraju memorijske adrese koje su bile u upotrebi od strane srušenog kernela i svih pokrenutih programa (procesa), jer bi to značilo nekonzistenciju podataka te bi cijela ova procedura bila uzaludna. U praksi postoje čak dva moguća modela ovakvog rada:

- Već opisani s učitavanjem novog posebnog kernela koji služi sâmo navedenoj svrsi.
- Poseban slučaj u kojemu se primarni kernel može ponovno iskoristiti (Engl. *Reuse*), ali samo na arhitekturama koje podržavaju Tzv. "*relocatable kernels*" metodu rada.

U većini slučajeva se koristi prvi pristup odnosno način rada s ovim posebnim kernelom.

## 16.1. Kako to radi

Sadržaj *RAM* memorije se prilikom pokretanja operativnog sustava popunjava te se u nju prvo učitava primarni kernel koji podiže ostatak operativnog sustava. Istovremeno se učitava i onaj mali *dump-capture* kernel u svoj izolirani dio RAM memorije koji se rezervira **SAMO** za njega. Stoga je nemoguće da RAM memorija koja je rezervirana i zauzeta od strane ovog posebnog kernela uopće bude dostupna primarnom Linux kernelu ili njegovim procesima odnosno programima, bilo namjerno ili slučajno. Ovo je važno jer želimo funkcionalan zaseban kernel koji nitko ne može "pobrkati" čak niti u slučaju kada se dogodi *rušenje* primarnog kernela. Dodatno, neke arhitekture procesora poput `x86` i `ppc64` traže da se pri inicijalizaciji sustava unaprijed rezervira fiksna veličina RAM memorije za proces pokretanja kernela bez obzira o poziciji gdje se ta memorija nalazila. U tom slučaju `kexec` kopira i taj dio RAM memorije tako da je on dostupan našem *dump-capture* kernelu. Veličina ove pred alocirane RAM memorije, kao i opcionalno njena pozicija, mogu se definirati u kernel "***boot***" parametrima sustava u opciji naziva: `crashkernel`.

## 16.2. Koji je proces pokretanja

Nakon što se primarni kernel pokrenuo, `kexec` naredba učitava i *dump-capture* kernel (kernel datoteku) i njegov pripadajući inicijalni RAM disk odnosno *initrd image* s korijenskim (*root*) datotečnim sustavom, koji se učitavaju u gore navedeni rezervirani i izolirani dio RAM memorije namijenjen za drugi kernel. Postoje i dodatni programi koji podržavaju `kdump` mehanizme. `Kdump` funkcionalnost, zajedno sa `kexec`-om je postala standardno integrirana s Linux kernelom od inačice 2.6.13 koja je razvijena 29.08.2005.g. Međutim kako bi sve ovo radilo, potrebno je zadovoljiti tri uvjeta:

- Minimalna inačica Linux kernela mora biti 2.6.13.
- Kernel mora biti konfiguriran da podržava `kexec`.
- Moraju biti instalirani pripadajući alati koji dolaze u programskom (softverskom) paketu imena: `kexec-tools`.


Vezano za inicijalni RAM disk i kreiranje inicijalnog korijenskog stabla direktorija, pogledajte poglavlje:
**10.7.1.2.1. initrd i/ili initramfs i inicijalizacija sustava.**

# 16.3. Kernel učitava kernel

Pokretanje odnosno učitavanje novog kernela iz postojećeg kernela nije u potpunosti podržano u kernelima starijim od 2.6.26 jer je do tada bilo moguće pokretanje kernela s jedne jedine adrese u memoriji, koja je naravno bila zauzeta s prvim kernelom koji se učitao. Dodatni problem je i u tome što je novi (dodatni) kernel koji se mora učitati, morao imati i zaštitu od direktnog pristupa RAM memoriji (*DMA* pristup). Osim toga novi *dump-capture* kernel mora imati mogućnost pristupa određenim informacijama starog kernela i to od trenutka kada se stari kernel srušio. Prema tome novi kernel mora imati i informaciju na kojim memorijskim lokacijama (adresama) se nalazi stari kernel. Za sve ove kao i mnoge druge radnje je zadužena naredba `kexec` kao i njeni pripadajući sistemski pozivi (`kexec_load`) preko kojih ona dolazi do potrebnih informacija.
*Kexec* na kraju krajeva i učitava odnosno pokreće novi kernel u rezervirano i izolirano područje memorije.
Konfiguracija `kexec` i `kdump` pâra programa odnosno servisa je specifična za svaku distribuciju Linuxa.
Mi ćemo se držati ***RedHat/CentOS*** distribucija, mada i među njima postoje male razlike u konfiguraciji.

**Početak**
Za početak je potrebno instalirati navedene alate potrebne za ovakav rad. Ovi alati/programi obično dolaze unutar softverskog paketa koji se obično zove `kexec-tools`. U daljem tekstu navesti ćemo primjere za *RedHat Enterprise Linux* (**RHEL**) pošto je on u najvećoj mjeri pripremljen za ovakav rad. Što se tiče *CentOS Linuxa* i kod njega je moguće konfigurirati sustav na ovakav način, ali uz dodatne korake koji prethode ovima za *RedHat Enterprise Linux*, ali i ovise o točnoj inačici.

**1.** Provjerimo trenutnu inačicu kernela upotrebom naredbe `uname` na sljedeći način:
```
uname -r

2.6.32-642.1.1.el6.x86_64
```
Dakle imamo kernel `2.6.32-642.1.1.el6.x86_64`, što znači da je trenutni kernel **2.6.32.x.**

**2.** Provjerimo je li kernel kompiliran sa svim potrebnim opcijama (za svaki slučaj).
```
egrep "KEXEC|CRASH|CAL_S" /boot/config-`uname -r`

CONFIG_KEXEC=y
CONFIG_KEXEC_AUTO_RESERVE=y
CONFIG_CRASH_DUMP=y
CONFIG_KEXEC_JUMP=y
CONFIG_PHYSICAL_START=0x1000000
CONFIG_CRASH=y
```
Sve tražene opcije su uključene (imaju postavljeno `=y`) u trenutni kernel pa idemo dalje.

**3.** Provjerimo s kojim opcijama vezanim za "*Crash kernel*" je naš kernel pokrenut:
```
grep crash /proc/cmdline

BOOT_IMAGE=/vmlinuz-2.6.32-642.1.1.el6.x86_64 root=UUID=fe0f5d06-afb2-42a6-b653-
e93f0397538d ro crashkernel=auto nomodeset rhgb quiet LANG=en_US.UTF-8
```

**4.** Vrijeme je za instalaciju `kexec-tools`-a na slijedeći način:
```
yum -y install kexec-tools
```

**5.** Sada je potrebno uključiti `kdump` servis tijekom podizanja sustava te ga odmah i pokrenuti.

**Za RedHat 6.x ili CentOS 6.x**
```
chkconfig kdump on
service kdump start
```
Provjerimo je li se sve pokrenulo kako treba:
```
/etc/init.d/boot.kdump status

kdump kernel loaded              running
```
Dakle sve je u redu, ako imamo status: `running`.

**Za RedHat 7.x ili CentOS 8.x+** ćemo sve pokrenuti na sljedeći način:
```
systemctl start kdump
systemctl enable kdump
```
Provjerimo je li se sve pokrenulo kako treba:
```
systemctl status kdump

kdump.service - Crash recovery kernel arming
Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset:
enabled)
Active: active (exited) since Mon 2016-05-30 20:15:28 CEST; 1 weeks 3 days ago
Main PID: 2526 (code=exited, status=0/SUCCESS)
CGroup: /system.slice/kdump.service
May 30 20:15:28 server01 systemd[1]: Starting Crash recovery kernel arming...
May 30 20:15:28 server01 kdumpctl[2526]: kexec: loaded kdump kernel
May 30 20:15:28 server01 kdumpctl[2526]: Starting kdump: [OK]
```
Vidimo da sve izgleda u redu odnosno da je servis pokrenut (`loaded`) i aktiviran prilikom pokretanja sustava (`enabled`).

**6.** Provjerimo imamo li sve potrebne naredbe:

```
which kexec

/sbin/kexec
```

i

```
which kdump

/sbin/kdump
```

Sjetimo se da `kdump` *startup* skripta tijekom pokretanja sustava učitava "*crash kernel*" u memoriju.

**7.** Možemo restartati sustav te provjerimo nalazi li se stvarno u RAM memoriji potrebni *crash kernel.*

```
grep -i crash /proc/iomem

  03000000-0b0fffff : Crash kernel
```

Vidimo memorijske adrese na kojima se nalazi učitan *crash kernel*, prema tome sve je u redu te je sve spremno za upotrebu.

**Konfiguracija kdump-a.**

Konfiguracija `kdump`-a se nalazi u datoteci `/etc/kdump.conf` tako da je moguće konfigurirati ili optimizirati sve potrebno, ako vaše potrebe ili želje odstupaju od standardnog.
Trenutna konfiguracija imena *core* datoteke je također vidljiva u *sysctl* varijabli: `kernel.core_pattern`.



Standardno se ***Core dump*** datoteke snimaju u direktorij: `/var/crash/XY/` na lokalni tvrdi disk (`XY`= obično datum). Osim sâme *core dump* datoteke `vmcore` snima se i ispis *dmesg* poruka sustava odnosno datoteka: `vmcore-dmesg.txt`.

**Ručno pozivanje *crash dump-a***

Za ručno pozivanje *crash dumpa* tj. snimanje stanja cijele RAM memorije sa svim učitanim programima, od trenutnog kernela (potencijalno problematičnog) je moguće sa ***SysRQ*** pozivima.



Pogledajte i poglavlje: **10.3.3.2 Tipke SysRq i Print Scrn.**

**Ukratko, to možemo napraviti ovako:**

Uključimo *SysRQ* funkcionalnost.

```
echo 1 > /proc/sys/kernel/sysrq
```

I ovako pokrećemo kernel *crash* te *crash dump* proceduru zapisivanja na disk:

```
echo c > /proc/sysrq-trigger
```

→Ovo je nasilna metoda koja ne zaustavlja pokrenute procese niti snima sve potrebno na disk.

Nježnija metoda, umjesto `echo c > /proc/sysrq-trigger` bi bila:

**1.** Uredno gašenje svih programa (`e`)

```
echo e > /proc/sysrq-trigger
```

**2.** Ugasimo i programe koji se nisu uspjeli lijepo i uredno ugasiti (ako ih ima) (`i`).

```
echo i > /proc/sysrq-trigger
```

**3.** Sinkronizirajmo sve zaostale podatke u RAM memoriji, na disk (`s`).

```
echo s > /proc/sysrq-trigger
```

**4.** Te na sâmom kraju odradimo *crash dump* metodu (`c`).

```
echo c > /proc/sysrq-trigger
```

Ovo može potrajati ovisno o tome koliko RAM memorije je bilo zauzeto jer se sva mora zapisati na tvrdi disk.
Kada se sve završi sustav će se restartati.

**Izvori informacija: (684),(685),(686),(K-14),** `man kexec`, `man kdump.conf`, `man kdumpctl`, `man 5 coredump.conf`, `man 5 core`, `man 5 procfs`.

## 16.4. NMI (*Non-Maskable Interrupts*) stanja

U slučajevima kada je sustav toliko zaglavljen da nije moguće poslati ***SysRQ*** naredbe, odgovor je reagiranje na *NMI* signale. *NMI* signali prekida odnosno "*Interrupti*" su posebni hardverski signali prekida s najvišim prioritetom koji se ne mogu ignorirati niti na koji način. Ovo su obično signali koji se kreiraju uslijed neke velike greške na hardveru ili slično. Stoga je dobro konfigurirati mogućnost da se u trenutku kreiranja *NMI* signala pokreće i *kernel dump*. Ako je ova mogućnost već uključena tada će u posebnoj datoteci: `/proc/sys/kernel/unknown_nmi_panic` biti postavljena vrijednost `1`. Provjerimo koju vrijednost imamo postavljenu na našem sustavu:

```
cat /proc/sys/kernel/unknown_nmi_panic

1
```

Ovo je omogućе i trajno postaviti zapisivanjem vrijednosti `kernel.unknown_nmi_panic = 1` u datoteku `/etc/sysctl.conf`.

# 17. Log datoteke

Zbog potrebe za praćenjem stanja i statusa rada raznih komponenti sustava logiraju odnosno snimaju se razne aktivnosti na sustavu u tzv. "*log*" odnosno *zapisničke* ili *dnevničke* datoteke. Naime, u Linuxu je moguće zapisivati informacije o više-manje svemu, na vrlo jednostavan način koji se svodi na zapisivanje u neku od za to predviđenih datoteka.
Ovakve dnevničke (zapisničke/log) datoteke sadrže poruke o sustavu, uključujući poruke od trenutka pokretanja sustava, poruke kernela, sistemskih servisa i aplikacija. Pri tome postoje različite dnevničke datoteke za različite potrebe. Primjerice, postoji točno zadana dnevnička (*log*) datoteka samo za poruke sustava prilikom pokretanja računala, zasebna datoteka za sigurnosne poruke sustava i tako dalje. Dnevničke (*log*) datoteke mogu biti vrlo korisne kada se pokušava riješiti neki problem na sustavu, kao što je pogreška prilikom pokretanja sustava, pogreška prilikom učitavanja upravljačkog programa (*kernel modula*) ili kada se traže neovlašteni pokušaji prijave na sustav i slično.



Standardno se dnevničke (*log*) datoteke pohranjuju u `/var/log/` direktoriju. Međutim ***journald*** sve dnevničke (log) datoteke snima u `/var/log/journal/` direktorij u binarnom formatu, koji se može pretraživati s naredbom `journalctl`.

Veći dio sistemskih dnevničkih (*log*) datoteka dohvaća i zapisuje servis koji se zove `rsyslogd`. On pohranjuje poruke raznih komponenti sustava u sljedeće tekstualne datoteke (o njima dolje u tekstu): `/var/log/messages`, `/var/log/secure`, `/var/log/maillog`, `/var/log/cron`, `/var/log/boot`, … Osim ovog servisa, i drugi servisi, programi, ali i komponente sustava zapisuju podatke u svoje dnevničke datoteke. To radi primjerice i sâm kernel, kao i razni poslužiteljski servisi; pr. *mariaDB, Apache, SSHD, Cron, Postfix,* ali i drugi sistemski servisi poput ***Init*** ili ***Systemd*** servisa i slično.
Od **Red Hat 7+ d**nevničkim (*log*) datotekama također upravlja i `journald` servis koji je komponenta `systemd` servisa. On također dohvaća sve *syslog* poruke, *kernel log* poruke, poruke inicijalnog RAM diska i poruke tijekom ranog pokretanja sustava, kao i poruke sa standardnog izlaza i standardnog izlaza za pogreške svih programa i servisa te ih indeksira i zapisuje.



Za detalje o **(r)syslog** servisu, pogledajte poglavlje:
**25.8.9. (R)syslog servis i usluga.**

Za detalje o **journald** servisu i naredbi za čitanje tih poruka, pogledajte primjere u poglavljima:
**7.3.2.3. Osnovni servisi (*daemoni*) unutar *systemd* softverskog paketa.**
**17.5. Journald, journalctl i dnevničke (log) datoteke.**

Većina dnevničkih (*log*) datoteka je u tekstualnom formatu (uz nekoliko iznimki). Pogledajte i kratku listu *log* datoteka s opisima:

- `/var/log/boot.log` – sadrži informacije koje se zapisuju tijekom pokretanja (engl. *Boot*) sustava tj. podizanja svakog pojedinog servisa (*daemona)* tijekom pokretanja sustava.
- `/var/log/cron` – sadrži informacije o pokretanju ***cron*** skripti, a koje zapisuje *crond* servis.
- `/var/log/dmesg` – sadrži informacija koje prosljeđuje kernel od trenutka pokretanja sustava. Ove poruke mogu se vidjeti i s naredbom `dmesg` ili s `journalctl`, ako koristite ***systemd***.
- `/var/log/grubby` – sadrži poruke koje kreira ***GRUB/GRUB2 boot loader***, tijekom pokretanja sustava s diska.
- `/var/log/messages` – sadrži globalne sistemske poruke koje se spremaju nakon trenutka pokretanja sustava, odnosno poslije inicijalizacije (r)*syslog servisa* ili se mogu vidjeti s naredbom `journalctl`, ako koristite ***systemd***.
- `/var/log/lastlog` – prikazuje informacije o zadnjem *logiranom* korisniku. Naredbom `lastlog` se može vidjeti sadržaj ove datoteke.
- `/var/log/maillog` – sadrži log informacije mail poslužitelja (poslužitelja elektroničke pošte).
- `/var/log/mcelog` – sadrži log informacije **MCE** sustava o hardverskim greškama računala.
- `/var/log/Xorg.x.log` – sadrži log informacije o *X Window* sustavu.
- `/var/log/btmp` – sadrži informacije o pogrešnim pokušajima *logiranja* na sustav, ali nije u tekstualnom formatu. S naredbom `last` može se vidjeti formatirana statistika iz ove datoteka, kao i s naredbom: `utmpdump`.
- `/var/log/cups` – prikazuje sve informacije vezane za pisače ili print servis (*daemon)* (***CUPS***).
- `/var/log/anaconda.log` – sadrži sve informacije o procesu instalacije Linuxa (od servisa ***Anaconda***).
- `/var/log/yum.log` – sadrži informacije koje logira YUM menadžer paketa tijekom svake instalacije softverskih paketa. Međutim ***DNF*** (nova inačica YUM-a u ***RedHat/CentOS*** 8x) snima sve svoje log poruke u drugu datoteku, imena: `/var/log/dnf.log`.
- `/var/log/secure` – sadrži sve informacije vezane za sigurnost koje mogu popunjavati razni servisi: primjerice *SSH* servis ovdje zapisuje sve poruke oko *logiranja* korisnika na ***ssh*** servis (kao i na sustav direktno). Ova datoteka sadrži i podatke o kreiranju, brisanju ili promjenama korisnika (korisničkih računa) ili korisničkih grupa.
- `/var/log/wtmp` – sadrži zapise o *logiranju* na sustav. Ova datoteka nije u tekstualnom formatu!. Naredba `who` čita iz ove datoteke. Ipak, sadržaj ove datoteke možemo vidjeti i s naredbom: `utmpdump /var/log/wtmp`.

## (R)syslog servis detaljnije

Vidjeli smo kako kernel i ostale komponente sustava, kao i razni programi zapisuju dnevničke odnosno *log* datoteke. Primarni servis za tu namjenu jest ***(r)syslog***.



Za detalje o **(r)syslog** servisu, pogledajte poglavlje:
**25.8.9. (R)syslog servis i usluga.**

## Način zapisivanja log datoteka

Svaka dnevnička (*log*) poruka je kategorizirani prema objektu (tzv. *facility*) odnosno definiciji: koji podsustav kreira poruku te prema prioritetu odnosno važnosti poruke, kako je vidljivo u tablici.

Pri tome *rsyslog* koristi objekt (***facility***) i prioritet kao parametre na osnovu kojih određuje kako postupati s dnevničkim porukama odnosno kamo ih spremati. Pravila o tome konfiguriraju se datoteci `/etc/rsyslog.conf` i u bilo kojoj datoteci s ekstenzijom **.conf** u `/etc/rsyslog.d/` direktoriju.

Pogledajmo tablicu s detaljima o objektima *syslog* poruka:

| **Objekt** (*Facility code*) | **Ključna riječ** | **Opis** |
|---|---|---|
| **0** | **`kern`** | Kernel poruke |
| **1** | **`user`** | Poruke s razine korisnika |
| **2** | **`mail`** | Mail sustav |
| **3** | **`daemon`** | Sistemski servisi |
| **4** | **`auth`** | Sigurnosne ili autentikacijske poruke |
| **5** | **`syslog`** | Poruke kreirane interno od strane *syslog* servisa |
| **6** | **`lpr`** | Poruke sustva za ispis (printanje) |
| **7** | **`news`** | Network news podsustav |
| **8** | **`uucp`** | UUCP podsustav |
| **9** | **`cron`** | Cron podsustav |
| **10** | **`authpriv`** | Autentikacijske poruke |
| **11** | **`ftp`** | FTP servis |
| **12** | **`ntp`** | NTP podsustav |
| **13** | **`security`** | Sigurnosne poruke |
| **14** | **`console`** | Log upozorenja |
| **15** | **`solaris-cron`** | Solaris cron poruke |
| **16–23** | **`local0 – local7`** | Lokalno korišteni objekti (facilities) |

Preslikavanje između kôda objekta (**facility**) i ključne riječi nije ujednačeno u različitim operativnim sustavima i implementacijama *rsyslog* servisa.

Popis važnosti odnosno ozbiljnosti (***severity***) također je opisan standardom i vidljiv u tablici:

| Vrijednost | Važnost (Severity) | Ključna riječ | Opis | Uvjet/razlog |
|---|---|---|---|---|
| **0** | Emergency | `emerg` | Sustav je neupotrebljiv | Panična poruka sustava. Pogledajte: **10.7.3. Machine Check Architecture (MCA).** **12.5. *Out of Memory* stanje rada i konfiguracija *OOM killera.*** |
| **1** | Alert | `alert` | Traži se hitna analiza sustava | Događaj koji se mora hitno analizirati i popraviti, poput korupcije podataka nekog servisa, baze podataka i sl. |
| **2** | Critical | `crit` | Kritični događaj | Hardverska greška. |
| **3** | Error | `err` | Error conditions | |
| **4** | Warning | `warning` | Važno upozorenje | |
| **5** | Notice | `notice` | Nomalni događaj | Događaj koji nije veća greška, ali bi potencijalno na njega trebalo obratiti pažnju. |
| **6** | Informational | `info` | Informacijska poruka | Informacija u radu programa (koji uredno radi). |
| **7** | Debug | `debug` | Debug-level poruke | Sadrže ekstemno detaljne informacije koje se koristi prilikom "debugging" porcesa analize programa (ili sustava) |

Sada pogledajmo sekciju s pravilima (*rules*) *rsyslog* centralne konfiguracijske datoteke: `/etc/rsyslog.conf`, uz naše komentare prije svakog retka konfiguracije:

```
#### RULES ####
# Logira se sve (osim: mail,authpriv,cron) od razine "info" na više
*.info;mail.none;authpriv.none;cron.none                /var/log/messages

# Za sve poruke koje se tiću sigurnosti
authpriv.*                                              /var/log/secure

# Logiranje svih mail poruka
mail.*                                                  -/var/log/maillog

# Logiranje svih poruka Cron servisa
cron.*                                                  /var/log/cron

# Za poruke tijekom pokretanja sustava
# Osim toga ovo je lokalna kategorija
local7.*                                                /var/log/boot.log
```

← U ovom (*Rules*) odjeljku je vidljiva definicija prema principu:
**Facility.Važnost datoteka**

Analizirati ćemo jedan unos:

```
authpriv.*        /var/log/secure
```

 |        |              |
***Facility*** ***Važnost*** ***Odredišna datoteka***

Moguće je i ručno poslati poruku, koju će *rsyslog* servis prihvatiti i pohraniit, primjerice ovako:

```
logger -p local7.notice "Probna Log poruka"
```

← ovdje smo u ***Facility*** imena (***local7***) za važnost razine **"notice"** poslali poruku "*Probna Log poruka*".

S obzirom da je u *rsyslog* konfiguraciji definirano:

```
local7.*                                                    /var/log/boot.log
```

← To znači da ***facility*** (*local7*) prihvača sve (.*) važnosti poruka te da će one biti pohranjene u datoteku: `/var/log/boot.log`. To što facility (*local7*) prihvača sve (.*) važnosti poruka znači da smo odmah mogli poslati poruku željene važnosti, primjerice važnosti (**error**):

```
logger -p local7.error "Log Grška"
```

**Oblik syslog poruka**

Syslog poruke se pohranjuju kako je definirano u **RFC 3164** odnosno u novijem **RFC 5424**.
Sadržaj poruke se pri tome kodira pmoću **UTF-8**, dok se poruka sastoji od zaglavlja i poruke, odnosno:

***Vremenske oznake – Imena računala – Aplikacije (i PID broja) – Sadržaja syslog poruke***

**Syslog kao mrežni protokol**

Važno je razumjeti da je *syslog* i mrežni protokol koji standardno koristi UDP kao transportni protokol na portu 514, a može se koristiti i TCP port 514, kako je definirano prema **RFC 6587**,
Osim toga **RFC 6587** definira i mogućnost kriptiranja *syslog* poruka upotrebom TLS-a, kako je definirano u: **RFC 5425**
Nadalje, moguća je i upotreba TCP porta 601, kako je navedeno u **RFC 3195** iako je ovo nešto stariji standard koji se u praksi rijeđe koristi. Prema definiciji *syslog* mrežnog protokola, *syslog* poruka se sastoji od:

- Zaglavlja u kojem su definirani: objekt (***facility***) i važnost (***severity***) poruke.
- Tijela poruke koje sadrži standardnu syslog poruku (prema **RFC 5424**):
  - Vrijeme poruke i ime računala s kojeg se šalje poruka
  - Aplikacije koja šalje poruku i njenog *PID* broja.
  - Sadržaja poruke.

Postoje i neke druge implementacije *syslog* protokola osim onog definiranog u standardnom **RFC 5424**, a to su:

- *BSD stil* syslog poruka, kako je definirano u **RFC 3164**.
- *Common Event Format* (**CEF**). **CEF** je važan jer pruža standardizirani format za bilježenje događaja povezanih sa sigurnošću, što olakšava integraciju dnevničkih (*log*) datoteka. To može biti osobito korisno za sigurnosne sustave poput: *security information and event management* (**SIEM**) sustava.

Neki drugi sustavi nude i drugačije formatiranje *syslog* poruka u:

- *Comma Separated Value* (**CSV**) formatu, koji je dobar stoga što se definiraju stupci odnosno polja koja želimo da se šalju na udaljeni poslužitelj, u sadržaju poruke. U *CSV* formatu, podaci su razdvojeni zarezima ili nekim drugim znakom za razgraničenje (delimiter znakom). *CSV* format je jednostavan i čitljiv, ali može biti manje pogodan za strukturirane podatke jer se polja mogu mijenjati ovisno o događaju.
- *Key-value formatu*. U *key-value* formatu, svaki događaj se sastoji od ključeva i vrijednosti koji su odvojeni znakom jednakosti ili nekim drugim znakom za razdvajanje. Ovaj format je fleksibilniji jer omogućuje uključivanje samo relevantnih informacija za svaki događaj.

**Primjer ručnog slanja mrežne syslog poruke na udaljeni syslog poslužitelj**

S našeg računala (IP: **192.168.100.201**) koje je konfigurirano da šalje *syslog* poruke na udaljeni *syslog* poslužitelj (IP: **192.168.100.1**) šaljemo jednu *syslog* poruku:

```
logger -p user.notice "Log Greška"
```

← Prije nego pošaljem ovu poruku, syslog poslužitelj s druge strane mora biti konfiguriran za primanje *syslog* poruka s mreže, kako je opisano u poglavlju: **25.8.9. (R)syslog servis i usluga.**

Pogledajmo ovu mrežnu *syslog* poruku snimljenu s programom **tcpdump** (*analizu radimo od OSI sloja tri [3]*):

```
(3) Internet Protocol Version 4, Src: 192.168.100.201, Dst: 192.168.100.1
(4) User Datagram Protocol, Src Port: 53483, Dst Port: 514
     Syslog message: USER.NOTICE: Apr 06 18:03:34 rocky9 root[4898]: Log Greška
     0000 1... = Facility: USER - random user-level messages (1)
     .... .101 = Level: NOTICE - normal but significant condition (5)
     Message: Apr 06 18:03:34 rocky9 root[4898]: Log Greška
          Syslog timestamp (RFC3164): Mar 29 13:03:34
          Syslog hostname: rocky9
          Syslog process id: root
          Syslog message id: [4898]: Log Greška
```

← Na OSI sloju tri (**3**) vidimo IP adrese pošiljatelja i primatelja. I konačno u transportnom sloju (**4**) vidimo da se koristi UDP protokol te da se paket šalje na port 514 (`Dst Port: 514`). I konačno vidimo *syslog* poruku, koja je sljedeće kategorije: ***Facility: USER → Level: NOTICE*** . I potom vidimo i samu poruku "`Log Greška`".

Kada bismo željeli da ova poruka bude poslana na udaljeni *syslog* poslužitelj i to u njegov ***facility*** (*local7*) tada bismo ju mogli poslati ovako:

```
logger -p local7.notice "Log Greška"
```

**Izvori informacija: (687),(688),(689),(690),(691),(K-14),** `man 8 logrotate`, `man 5 utmp`, `man rsyslogd`, `man rsyslog.conf`, `man systemd`, `man logger`, `man 8 systemd-journald.service`, `man journalctl`, `RFC 3164`, `RFC 3195`, `RFC 5424`, `RFC 5425`, `RFC 6587`.

## 17.1. Servis *logrotate*

Vidjeli smo kako većina *Unix*/Linux servisa zapisuje svoje aktivnosti i status rada u *log* datoteke koje se nalaze unutar vršnog direktorija: `/var/log/`. Svaki od tih servisa može zapisivati veću ili manju količinu podataka odnosno više ili manje detalja, sve ovisno o tome kako je konfiguriran. Stoga je moguće da log datoteke prepune disk, pa nam tu u pomoć dolazi servis *logrotate*. Servis **`logrotate`** osmišljen je kako bi pojednostavio administraciju *log* datoteka na sustavu koji ili generira mnogo *log* datoteka ili su one popriličnе veličine. *Logrotate* omogućuje automatsku kompresiju *log* datoteka, uz njihovo rotiranje, uklanjanje i potencijalno slanje drugdje. *Logrotate* se dodatno može podesiti kako bi upravljao dnevnim, tjednim ili mjesečnim *log* datotekama do određene veličine. *Logrotate* je standardno već instaliran, a ako to nije slučaj, možemo ga instalirati sa:

```
yum -y install logrotate
```

Osnovna konfiguracijska datoteka *logrotate* servisa se nalazi u: `/etc/logrotate.conf`.
Pogledajmo što je definirano u njoj. Skratili smo ispis na osnove opcije, a i pobrojali neke dodatno korisne:

- `weekly` – ovaj redak (naredba) označava kako će se log datoteke rotirati na tjednoj bazi (svaki tjedan). Moguće je umjesto ovoga definirati rotaciju na dnevnoj bazi sa: `daily` ili na mjesečnoj bazi sa: `monthly`.
- `create` – ovaj redak (naredba) označava kako će se kada se popuni prethodna log datoteka, kreirati nova(e) sve do maksimalno njih; definirani broj u sljedećoj opciji (`rotate`).
- `rotate 4` – ovaj redak (naredba) označava kako će se čuvati **4** tjedna log datoteka unazad.
- `dateext` – ovaj redak (naredba) označava kako će se kod naziva novih rotacijskih datoteka dodavati datum kao sufiks.
- `#compress` – ovaj redak (naredba) označava da se log datoteke ne komprimiraju, a ako maknemo znak `#` onda će se komprimirati (osim trenutno aktivne log datoteke koja se nikada ne komprimira).
- `include /etc/logrotate.d` – ovaj redak (naredba) označava kako novo instalirani programski paketi (*RPM*) unutar navedenog direktorija mogu kreirati svoja pravila za svoje log datoteke.

Moguće je definirati i veličinu log datoteke u MB, koja se mora doseći i tek tada će se rotirati nova log datoteka.
Ova oznaka je `size XM`. Primjerice za veličinu 10 MB, bi to bilo: `size 10M`.
Standardno se s instalacijom *logrotate* servisa kreira ***crontab*** datoteka unutar `/etc/cron.daily/` direktorija pod nazivom `logrotate`. Kao što je slučaj i s ostalim ***crontab*** datotekama unutar tog direktorija, to znači kako će se svakodnevno pokretati servis *logrotate* i odrađivati zadane mu operacije s log datotekama.

Rotacija *log* datoteka znači sljedeće; pogledajmo naš log direktorij: `/var/log/` i datoteke koje u imenu imaju riječ `messages` jer nas sada zanimaju ove datoteke u koje se snimaju sistemske *log* poruke. Mi ovdje imamo sljedeće datoteke:

```
ls -al /var/log

-rw-------   1 root root           18468253 Jun 18 07:33 messages
-rw-------   1 root root            3803963 May 26 03:15 messages-20190526
-rw-------   1 root root            4438195 Jun  2 03:21 messages-20190602
-rw-------   1 root root            4445081 Jun  9 03:45 messages-20190609
```

Pri tome je trenutno aktivna log datoteka imena `messages` od trenutnog tjedna. Kada istekne prvi tjedan ova datoteka se kopira u datoteku imena: `messages-20190609` (2019.g, 06 mj. 9.dan)… i tako se za svaki tjedan čuva po jedna tjedna datoteka koja u nazivu ima datum do kojega se u njoj nalaze (log) podaci. Mi na našem sustavu vidimo log datoteke za zadnja četiri nîza tjedana te petu aktivnu datoteku za trenutni peti tjedan, sve kako je definirano u konfiguracijskoj datoteci servisa *logrotate*: `/etc/logrotate.conf`. Pošto je standardno definirano kako se unutar direktorija: `/etc/logrotate.d/` mogu kreirati nove konfiguracijske datoteke za svaki servis (*daemon*) zasebno, tako se ovdje nalaze datoteke za one servise koji trebaju dodatnu optimizaciju *logrotate* servisa. Naime konfiguracijske datoteke unutar ovog direktorija imaju veću važnost od globalne konfiguracije definirane u datoteci: `/etc/logrotate.conf`.
Odnosno ove vrijednosti i opcije će pregaziti one standardno postavljene. Pogledajmo jedan primjer konfiguracije *logrotate* datoteke: `/etc/logrotate.d/bootlog` za logiranje poruka prilikom pokretanja sustava (tzv. *boot log*):

```
/var/log/boot.log
{
    missingok
    daily
    copytruncate
    rotate 7
    notifempty
}
```

Objašnjenje redaka ove konfiguracije slijedi:

- `/var/log/boot.log` - označava kako će se *boot log* poruke snimati u navedenu datoteku.
- `missingok` - označava kako neće biti problema, ako nedostaje *log* datoteka (ako ih trenutno uopće nema).
- `daily` - označava kako će se *boot log* poruke rotirati dnevno (svaki dan).
- `copytruncate` - označava kako će se stara *boot log* datoteka kopirati u onu rotiranu (koja se čuva), a original potom isprazniti i početi popunjavati od početka s novim zapisima (od početka dana, na dalje).
- `daily` - označava kako će se *boot log* poruke rotirati dnevno (svaki dan).
- `rotate 7` - označava kako će se *boot log* poruke rotirati dnevno (svaki dan) i to do sedam (7) dana.
- `notifempty` - označava kako se *boot log* datoteke koje su prazne neće rotirati.

**Izvor informacija: `man logrotate`,`man 5 logrotate.conf`.**

# 17.2. Datoteka `/var/log/messages`

Datoteka `/var/log/messages` sadrži globalne sistemske poruke koje se spremaju nakon trenutka pokretanja sustava, ali tek poslije inicijalizacije *r/syslog* servisa. Dakle u ovu datoteku neće biti upisane poruke kernela u sâmom trenutku inicijalizacije sustava, ali će nakon toga u njoj završiti sve ostale sistemske poruke kao i poruke sâmog kernela, a koje se inače sve zapisuju u datoteku `/var/log/dmesg`. Pretpostavka je da imate instaliran *(r)syslog* servis, a ako nemate, instalirajmo ga:

```
yum -y install rsyslog
```

Ako nije bio instaliran, ne zaboravite ga pokrenuti i permanentno aktivirati.
***Od RedHat 8+ rsyslog servis se više ne koristi, već to odrađuje systemd-journald servis i pripadajuća naredba*** `journalctl`.
Ostale log datoteke ovise o konfiguraciji **`(r)syslogd`** servisa, koji se obično konfigurira u datoteci: `/etc/rsyslog.conf`.
**R/syslogd** servis moguće je konfigurirati da prima i sprema poruke od drugih računala preko mreže, preko *syslog* mrežnog protokola. ***Syslog protokol u mrežnom radu obično koristi UDP transportni protokol na portu 514***.



Za detalje o **(r)syslog** servisu i **journald** servisu, pogledajte poglavlja:
**25.8.9. (R)syslog servis i usluga.**
**17.5. Journald, journalctl i dnevničke (log) datoteke.**

**Krenimo s primjerima.**
Pošaljimo poruku koja će biti vidljiva unutar ljuske (*shella)* te će se zapisati u sistemsku log datoteku `/var/log/messages` koju možemo pregledati i s naredbom `dmesg`. Poruka koju šaljemo neka bude "`–TEST–`".
Naredba za ovu namjenu je `logger`, a njen prekidač `-s` označava kako će osim zapisa u datoteci: `/var/log/messages` poruka biti i prikazana u naredbenoj ljusci (*shellu*).

```
logger -s "--TEST--"
```

Provjerimo je li se sve upisalo u sistemsku log datoteku (`/var/log/messages`).

```
tail /var/log/messages

Sep 20 17:01:24 Server1 root: --TEST--
```

Vidimo kako je poruka ovdje, što smo i očekivali.

**Izvor informacija:** **`man logger`**,**`man rsyslogd`**.

---

# 17.3. Datoteka `/var/log/dmesg`

Druga datoteka, odnosno u slijedu pokretanja sustava prva, sadrži poruke kernela koje se zapisuju tijekom učitavanja kernela, odnosno u ranoj fazi pokretanja operativnog sustava, ali i dalje tijekom podizanja ostatka operativnog sustava.
Sve ove poruke zapisuju se u datoteku: `/var/log/dmesg`.
Da bi na sustavu imali ovu datoteku, potrebno je imati instaliran softverski paket imena ***initscripts***.
**Međutim od *Red Hat/CentOS 8.x.* ova datoteka se više ne koristi, već je integrirana u *journal log* datoteku.**

Dodatno, postoji i metoda direktnog slanja željene poruke kernelu, ili slanja poruka sâmog kernela, a koja će završiti zapisana u takozvanom kernel *message bufferu* odnosno posebnoj memoriji namijenjenoj za spremanje poruka sustava.
Sve poruke koje se trebaju logirati, a koje kernel šalje, šalju se preko posebnog uređaja (engl. *character device*): `/dev/kmsg` koji predstavlja takozvani *printk buffer* odnosno kernelovu međumemoriju za poruke (engl. *message buffer)*.
Zbog toga je moguće i iz aplikacijske razine poslati poruku koja će završiti logirana u kernel *message bufferu*.

Pogledajmo i kako. U primjeru ćemo poslati poruku „*`Ovo je test`*" na ovu međumemoriju za poruke:

```
echo Ovo je test > /dev/kmsg
```

Ova poruka će biti odmah vidljiva u kernel *memoriji za poruke (message bufferu)*. Ispišimo zadnjih par poruka od sustava:

```
dmesg | tail -n1
```

Važno je razumjeti kako poruke koje završe u kernel *memoriji za poruke*, nakon što se sustav pokrene, neće biti zapisane u datoteku `/var/log/dmesg` jer u tu datoteku se zapisuju samo log poruke tijekom pokretanja sustava.

Konkretno sve poruke tijekom pokretanja sustava (u *booting* fazi) koje završe u kernel *memoriji za poruke* se zapisuju u datoteku `/var/log/dmesg`. Sve naknadne poruke kernela, koje završe u kernel *memoriji za poruke*, zapisuju se konstantno u datoteku: `/var/log/messages`, koju smo opisali u prethodnom poglavlju.



Naredba `dmesg` čita trenutne poruke iz kernel *message buffera*, a ne iz datoteke: `/var/log/dmesg`.

Na distribucijama Linuxa koje koriste **systemd** umjesto **init**-a (*RedHat/Centos* **7.x+**) postoji i naredba `journalctl` s kojom možemo pregledavati ove poruke pomoću prekidača `-k` koji znači da se prikazuju i sve kernel poruke.
Na sustavima koji koriste ***systemd*** servis (***systemd-journald*** konkretno), više ne postoji datoteka `/var/log/dmesg`, pa je potrebno koristiti naredbu:

```
journalctl -k
```

Pomoću naredbe `dmesg` možemo napraviti i kopiju (backup) trenutnih *log* podataka kernela:

```
dmesg > backup.dmesg.log
```

Moguće je i isprazniti *kernel memoriji za poruke (kernel message buffer)* na sljedeći način:

```
dmesg -c
```

Na razini cijelog sustava moguće je i definirati do koje razine želimo da se logiraju (spremaju) poruke kernela.

Prvo pogledajmo koje sve razine logiranja poruka postoje[(220)], prema važnosti odnosno *težini* poruka:

- `0` - *KERN_EMERG* - sustav je neupotrebljiv.
- `1` - *KERN_ALERT* - potrebno je ekstremno hitno nešto poduzeti.
- `2` - *KERN_CRIT* - kritična greška, potrebno je nešto učiniti.
- `3` - *KERN_ERR* - dogodila se neka greška.
- `4` - *KERN_WARNING* - poruka važnijeg upozorenja.
- `5` - *KERN_NOTICE* - poruka običnog upozorenja.
- `6` - *KERN_INFO* - informacijska poruka.
- `7` - *KERN_DEBUG* - *debug* razina poruka (detaljni ispis).

Definicija razine *logiranja* kernel poruka se zapisuje u posebnu datoteku: `/proc/sys/kernel/printk`[(221)].
Sadržaj ove datoteke izgleda obično ovako:

```
cat /proc/sys/kernel/printk

4        4        1        7
```

Objasnit ćemo što ova datoteka sadrži:

- Prvi broj (`4`) u primjeru gore označava konzolni *log level (razinu detaljnosti kod logiranja).*
- Drugi broj (`4`) u primjeru gore je standardni (*default*) *log level (razina logiranja).*
- Treći broj (`1`) u primjeru gore je minimalni *log level (razina logiranja).*
- Četvrti broj (`7`) u primjeru gore je *log level (razina logiranja)* kod pokretanja sustava.

Kako promijeniti *konzolni log level* odnosno razinu detaljnosti tijekom logiranja poruka:

Prva metoda s kojom možemo podići razinu logiranja na pet (5) je:

```
echo 5 > /proc/sys/kernel/printk
```

Druga metoda, upotrebom naredbe `dmesg`:

```
dmesg -n 5
```



Pogledajte i poglavlje:
**7.3.2.5. Drugi korisni programi i komponente unutar *systemd* paketa.**

**Izvor informacija: (219), (220), (221),** `man dmesg`, `man journalctl`.

### 17.3.1. Kratka analiza problema na sustavu

**Slijedi napredna cjelina!**
U ovoj cjelini vidjet ćemo nekoliko poruka s greškama koje smo susreli u radu te s njihovim objašnjenjima odnosno rješenjima. Naime u radu našeg sustava, moguće je da se s vremena na vrijeme pojave i određene greške; neke od njih mogu biti samo softverske, a neke mogu ukazivati i na neke dublje hardverske probleme, koje je nužno otkloniti.
Dakle gledat ćemo poruke koje smo mogli dobiti prvenstveno u datoteci: `/var/log/dmesg`.

## Problemi s mrežnom karticom

Pogledajmo jedan nîz poruka vezanih za greške na mrežnoj kartici čiji je upravljački program `be2net`:

```
1 kernel: be2net 0000:04:00.4: Unrecoverable Error detected in the adapter
2 pci 0000:04:00.4: BAR 6: no space for [mem size 0x00040000 pref]
  pci 0000:04:00.4: BAR 6: failed to assign [mem size 0x00040000 pref]
3 pci 0000:04:00.4: BAR 7: trying firmware assignment [mem 0xffffc000-0x10005bfff]
  pci 0000:04:00.4: BAR 7: [mem 0xffffc000-0x10005bfff] conflicts with reserved [mem 0xff800000-0xffffffff]
  pci 0000:04:00.4: BAR 7: failed to assign [mem size 0x00060000 64bit]
  be2net: Please reboot server to recover
```

Nakon restarta poslužitelja dobivamo i sljedeće poruke, ali tek nakon nekog vremena njenog ispravnog rada:

```
kernel: be2net 0000:04:00.4: Error detected in the adapter
kernel: be2net 0000:04:00.4: eno1: Link down
```

A nakon toga mreža više ne radi. U ovom konkretnom slučaju radi se o mrežnoj kartici: ***Emulex Corporation OneConnect 10Gb NIC*** i njenom upravljačkom programu: `be2net`. U svakom slučaju na poziciji (1) vidimo da je detektirana greška koja se nije mogla ispraviti. Radi se o mrežnoj kartici sa **PCI** identifikatorom: `04:00.4`. čije detalje bi mogli pronaći s naredbom:

```
lspci -s 04:00.4
```

Nadalje se spominje problem vezan za *firmware* mrežne kartice (3). Ova mrežna kartica je vidljiva kao `eno1`. U konkretnom slučaju, detaljnijim istraživanjem više izvora informacija (***Google, Emulex***, ***RedHat***, ***CentOS***, ***HP***) utvrđeno je kako se radi o grešci unutar *firmware*-a mrežne kartice.



Vezano za *firmware* pogledajte poglavlje:
**11.1.2.1.2. Nadogradnja *firmwarea* uređaja**.

Dakle rješenje je bilo nadogradnja *firmware*-a, a dodatno smo napravili i nadogradnju kernela, a sâmim time i upravljačkog programa mrežne kartice.



Vezano za detalje oko nadogradnjuekernela, pogledajte poglavlje:
**11.1.1. Rad s kernelom**.

U nekim ekstremnim slučajevima grešaka na mrežnim karticama, a pogotovo, ako plaćate podršku za Linux (pr. ***RedHat***) imate i pravo otvoriti tehnički incident, za koji je moguće izraditi i tzv. *firmware dump* odnosno analizu *firmware*-a mrežne kartice, sa sljedećom naredbom:

```
ethtool -d eno1 raw on > /root/lan.eno1.dump.raw
```

Tada centru za tehničku podršku (pr. *Red Hat*) treba proslijediti datoteku koju smo izradili: `/root/lan.eno1.dump.raw`.

## Druga mrežna kartica – drugi problem

U ovom slučaju imamo ***Intelovu*** gigabitnu mrežnu karticu koja u nekom trenutku prestaje raditi.

Njem upravljački program (kernel modul) je `igb`. Pogledajmo koje poruke smo dobivali:

```
kernel: igb 0000:05:00.0: Intel(R) Gigabit Ethernet Network Connection
kernel: igb 0000:05:00.0: eth0: (PCIe:5.0Gb/s:Width x4) ec:9e:cd:11:45:b2
kernel: igb 0000:05:00.0: Using MSI-X interrupts. 8 rx queue(s), 8 tx queue(s)
kernel: ADDRCONF(NETDEV_UP): eth0: link is not ready
```

Dakle na sustavu u jednom trenutku mrežna kartica `eth0` više ne radi. Nakon intenzivne pretrage interneta (***Google, Intel***, ***RedHat***/***CentOS/Rocky***) te otvaranja tehničke podrške na ***RedHat***-u, utvrđeno je da se radi o grešci unutar upravljačkog programa mrežne kartice.

Ovaj problem je riješen nadogradnjom sustava na novi kernel i pripadajući upravljački program (`igb`) za navedenu mrežnu karticu, mada bi rješenje bilo samo nadogradnja upravljačkog programa. Ipak jednostavnije, ali i preporučeno je bila i nadogradnja kernela s kojom je dolazila novija inačica upravljačkog programa mrežne kartice.

Točnu inačicu upravljačkog programa mrežne kartice (pr. `eth0`) možemo vidjeti sa sljedećom naredbom:

```
ethtool -i eth0
```

Tada ćemo dobiti osnovne informacije o upravljačkom programu poput (skratili smo ispis):

```
driver: igb
version: 5.6.0-k
firmware-version: 1.2.11
bus-info: 0000:05:00.0
```

Nadalje, u slučajevima kada vam je potrebno, a upravljački program to podržava, moguće je povećati razinu *logiranja* razniih poruka u radu mrežne kartice. Primjerice za `igb` mrežnu karticu, za to imamo parametar imena `debug`, koji se definira prilikom učitavanja upravljačkog programa mrežne kartice.

Listu parametara koje podržava mrežna kartica odnosno njen upravljački program, možemo dobiti sa sljedećom naredbom:

```
modinfo igb | grep parm
parm:      max_vfs:Maximum number of virtual functions to allocate per physical function (uint)
parm:      debug:Debug level (0=none,...,16=all) (int)
```

Za konkretni upravljački program `igb` imamo samo dva parametra: `max_vfs` i `debug`.

To sve možemo postići ako prvo maknemo upravljački program mrežne kartice te ga ponovno učitamo, ali na sljedeći način:

```
rmmod igb && modprobe igb debug=16
```

Tada smo maksimalno podigli razinu logiranja svih poruka ove mrežne kartice (zbog potreba detaljnije analize rada iste).



*Vezano za firmware pogledajte poglavlje:*
***11.1.2.1.2. Nadogradnja firmwarea uređaja***

## Tainted poruke kernela

Sljedeći slučaj su takozvane ***Tainted*** poruke, poput:

```
Kernel is Tainted
nvidia: module license 'NVIDIA' taints kernel
Raw taint value as int/string: 01/' G/P '
```

To znači kako je kernel izbacio poruku (koja može biti: sâmo upozorenje ili greška), kako je upravljački program odnosno kernel modul `nvidia` (za ***Nvidia*** grafičku karticu) pisan/razvijan kao zatvoreni kôd.
Dakle ovo je sâmo poruka o različitoj licenci s kojom je objavljen ovaj upravljački program, od one s kojom je objavljen kernel Linuxa.



Pogledajte poglavlje:
**11.1.1.6. Tainted kernel.**

## IPMI kartica

Potom imamo slučaj u kojemu upravljački program za mrežnu menadžment karticu **IPMI** izbacuje sljedeće poruke/greške:

```
IPMI message handler: BMC returned incorrect response, expected
Hardware name: ARTESYN ACPI5-A/ACPI5-A, BIOS 1.5.06 X64 12/14/2015
Call Trace:
<IRQ> [<ffffffff959135d4>] dump_stack+0x19/0x1b
 [<ffffffff95346412>] __report_bad_irq+0x32/0xd0
 [<ffffffff95346832>] note_interrupt+0x132/0x1f0
 [<ffffffff95343f65>] handle_irq_event_percpu+0x55/0x80
```

Detaljnijim istraživanjem više izvora informacija (**Google, *RedHat*, *CentOS***) utvrđeno je kako se radi o grešci unutar upravljačkog programa za ***IPMI*** karticu/sustav, koja se pojavila u trenutnoj inačici ***CentOS*** Linuxa 7.2.
Ova greška ispravljana je u novoj inačici (ili s novijim kernelom i njegovim pripadajućim upravljačkim programom za konkretni ***IPMI*** sustav).

## Disk kontroler

Potom imamo sljedeći slučaj, s porukama poput:

```
kernel: isci 0000:8:00.0: sci_remote_node_context_start_io: invalid state
RNC_TX_RX_SUSPENDED
kernel: isci 0000:8:00.0: sci_remote_node_context_start_io: invalid state
RNC_TX_RX_SUSPENDED
kernel: isci 0000:8:00.0: sci_remote_node_context_start_io: invalid state
RNC_TX_RX_SUSPENDED
```

Ove poruke se, ako pregledamo cijelu log datoteku, odnose (što smo utvrdili prema **PCI** identifikatoru: `8:00.0`) na:

```
isci: Intel(R) C600 SAS Controller Driver - version 1.2.0
isci 0000:08:00.0: driver configured for rev: 6 silicon
scsi host5: isci
```

Ovdje vidimo da kernel komponenta (`isci`) na **PCI** sabirnici, na adresi: `8:00.0` svako malo, izbacuje poruku: „`sci_remote_node_context_start_io: invalid state RNC_TX_RX_SUSPENDED`“.

Dakle radi se o ***Intel c600*** SAS disk kontroleru koji koristi upravljački program `isci`, koji koristi *SCSI* diskovni podsustav, da kada od hardvera primi poruku `TX_RX_SUSPENDED` da tada prelazi u novi kontekst (unutar upravljačkog programa) i tada se dogodi greška. Problem je riješen prelaskom na noviji kernel i sâmim time na noviju inačicu upravljačkog programa (kernel modula) za navedeni disk kontroler.

## NTP Servis

Slijedeće poruke se odnose na ***NTP*** servis odnosno greške koje on izbacuje. Pogledajmo ih:

```
ntpd[4200]: 0.0.0.0 0618 08 no_sys_peer
root: NTP daemon is not connected to any NTP peer! Please check server1
```

Ovdje vidimo grešku koju nam je dao *NTPD* servis u kojoj kaže: „`no_sys_peer`“ što znači kako se ne može spojiti niti na jedan udaljeni **NTP** poslužitelj, a što se potvrđuje u sljedećoj poruci:
„`NTP daemon is not connected to any NTP peer`“.

Prva stvar ovdje bi bila provjeriti na koje udaljene *NTP* poslužitelje smo konfigurirani da se spajamo:

```
ntpq -pn
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
-192.168.1.100  161.53.123.8     3 u    3   16  377    0.299   -0.040   0.010
-192.168.1.20   213.251.52.217   3 u    -   16    0    0.000    0.000   0.000
```

Vidimo kako je naš sustav konfiguriran da koristi lokalne *NTP* poslužitelje (`192.168.1.100` i `192.168.1.20`), ali da su oba nedostupna, što vidimo po oznaci ispred njih (`-`).



Za detalje o **NTP** protokolu pogledajte poglavlje**:**
**26.6. NTP.**

U ovom slučaju uzrok je bio problem s mrežom te su stvarno oba konfigurirana *NTP* poslužitelja bila nedostupna. Inače bi bilo potrebno odabrati minimalno dva nova (druga) ***NTP*** poslužitelja koji su dostupni i stabilni te ih upisati u datoteku: `/etc/ntp.conf`, a potom restartati ***ntpd*** servis.

**Dodatni izvor informacija:** **`man dmesg`**,**`man lspci`**,**`man ethtool`**.

## 17.3.2. SOS reporting

*Sos reporting* odnosno program za izradu izvješća je alat naredbenog retka u ***Red Hat*** kompatibilnim distribucijama Linuxa kao što su ***CentOS***, ***Fedora***, ***Rocky***, ***Oracle Linux*** i druge, ali koriste ga i druge grane distribucija Linuxa poput ***Ubuntu Linuxa***.
On prikuplja informacije o konfiguraciji i dijagnostici vašeg sustava kao što su:

- *Boot* informacije: ***initrd*** i **GRUB2** te informacije tijekom učitavanja kernela i pokretanja računala.
- Informacije o blok uređajima i particijama te datotečnim sustavima i montiranim diskovima.
- Detalji o memoriji (RAM) i pokrenutim programima (procesima) te listu svih otvorenih datoteka i druge detalje.
- Ime računala, sistemska IP adresa, postavke vatrozida, tablica usmjeravanje (*routing* tablica*)* i ARP tablica.
- Detalji o mreži (sva mreža sučelja s parametrima rada, IP adrese.
- Inačica OS-a i kernela, učitani kernel moduli (upravljački programi) i detalji o njima, ***SELinux***, **Systemd**, …
- Lista instaliranih programskih paketa.
- Listu PCI i USB uređaja, svog hardvera na sustavu i ***udev*** sustav i parametre rada.
- Konfiguracijske datoteke sustava i sistemskih servisa te sve važne log datoteke sustava.

Ova naredba također pokreće i dodatne programe i naredbe za prikupljanje daljnjih informacija te ih sve zajedno pohranjuje u arhivsku datoteku. *Sos report* se može koristi na sljedeći način odnosno primijeniti za:

1. **Generiranje izvještaja** – korisnik pokreće naredbu `sosreport` ili `sos report` u terminalu (naredbenom retku) kako bi generirao izvještaj. Ova naredba će pokrenuti skriptu koja će prikupiti informacije o sustavu, uključujući konfiguracijske datoteke, log datoteke, mrežne postavke, informacije o procesima i mnoge druge dijagnostičke informacije koje smo već spomenuli.
2. **Analizu problema** – nakon što se izvještaj generira, može se analizirati kako bi se identificirali problemi ili potencijalni uzroci problema na sustavu. Izvještaj može pružiti korisne informacije o konfiguraciji sustava, mrežnim postavkama, opterećenju procesora, memoriji i drugim resursima, kao i o događajima koji su se dogodili na sustavu.
3. **Pomoć pri rješavanju problema** – korisnici mogu dijeliti generirane izvještaje s tehničkom podrškom ***Red Hat-a*** ili zajednicom kako bi dobili pomoć pri rješavanju problema. Izvještaj pruža detaljne informacije koje mogu pomoći u dijagnozi problema i pronalaženju rješenja.
4. **Poboljšanje performansi i sigurnosti** – prikupljene informacije iz izvještaja mogu se koristiti za poboljšanje performansi i sigurnosti sustava. Analiza izvještaja može otkriti potencijalne probleme ili sigurnosne propuste koji se mogu adresirati kako bi se poboljšala stabilnost i sigurnost sustava.



*Sos reporting* nije bio dostupan u drugim ***Red Hat*** kompatibilnim distribucijama Linuxa (pr. ***CentOS***) sve do inačice 6.x. *Sos reporting* je dostupan od komercijalne inačice ***Red Hat Entrprise Linuxa*** v.4.x., ali se naziv alata mijenjao s vremenom. Od ***Red Hat Entrprise Linuxa*** v.4.x do v.7. ovaj program (naredba) se zvao `sosreport`, dok se od v.8.x. on zove `sos report`. Odnosno naredba je `sos` s dodatnim argumentima, među kojima je i argument `report`.
**Ubuntu** distribucija Linuxa od **v.20.04+** koristi naredbu `sosreport`.

Važno je znati da je *sos report (*izvješće*)* potrebno kada želite otvoriti slučaj (*ticket*) ***Red Hat-ovoj*** tehničkoj podršci. Važno je razumjeti da kada otvorite slučaj na tehničkoj podršci, inženjeri za podršku *Red Hata* zahtijevat će SOS izvješće vašeg poslužitelja u svrhu analize i rješavanja problema.

**Instalacija i upotreba**
Za upotrebu *sos* alata potrebno ga je prethodno instalirati, što ćemo postići sa:

```
yum -y install sos
```

Da bismo kreirali *sos izvještaj*, potrebno je pokrenuti (za ***Red Hat 9.x*** odnosno konkretno za ***Rocky Linux 9.3.***):

```
sos report

sosreport (version 4.6.1)
This command will collect system configuration and diagnostic
Press ENTER to continue, or CTRL-C to quit.
```

Sada trebamo potvrditi nastavak (stisnite tipku *ENTER*).

```
Optionally, please enter the case id that you are generating this report for []:
```

Ako imate broj slučaja (*ticketa*) koji ste otvorili tehničkoj podršci (***Red Hat***, ***Rocky***, ***Oracle***, …) ovdje ga upišite ili samo nastavite odnosno stisnite tipku *ENTER*.
I kada sve bude gotovo dobit ćete poruku poput ove:

```
Your sosreport has been generated and saved in:
        /var/tmp/sosreport-Rocky-9-2024-03-10-gkatids.tar.xz
 Size   10.02MiB
```

Na kraju je kreirana *sos arhivska* datoteka: `/var/tmp/sosreport-Rocky-9-2024-03-10-gkatids.tar.xz`.

Arhivske datoteke snimaju se automatski u direktorij `/var/tmp/`, a uobičajeni naziv im je:
*sosreport-IME_RAČUNALA-DATUM-BROJ_TICKETA(ili oznaka)**.tar.xz***
Pri tome se arhiva može komprimirati i s drugim algoritmima, pa može imati i drugačiju ekstenziju (pr. `.gz`, `.bz2` ili `.xz`)

U slučaju kada plaćate komercijalnu podršku za ***Red Hat Enterprise*** Linux i otvorili ste slučaj (*ticket*) tehničkoj podršci, tada je osim ručnog kopiranja *sos reporta*, moguće (ako imate direktan pristup internetu s vašeg poslužitelja) to učiniti i automatski prilikom kreiranja izvještaja. Odnosno moguće je sve učiniti u jednom koraku:

```
sos report --upload
```

← Pri tome je obavezno navesti broj slučaja (*ticketa*), a autentikacija je uobičajena za prenošenje izvještaja na udaljeni poslužitelj ili platformu. To obično uključuje unos korisničkog imena i lozinke, što se također može učiniti i ovako:

```
sos report --upload --upload-user KORISNIK --upload-pass LOZINKA
```

Konfiguracija *sos sustava* se nalazi pohranjena u datoteci: `/etc/sos/sos.conf`. Navedene parametre *autentikacije* (korisničko ime, lozinka ili token), uz druge opcije, moguće je postaviti u navedenu konfiguracijsku datoteku ili u varijable okruženja sustava (tzv. *environment varijable*).

**Modularnost**
Naredba `sos report` ima modularnu strukturu i omogućuje korisniku da omogući ili onemogući željene profile i module te odredi opcije modula putem naredbenog retka. Za popis dostupnih profila i tzv. *plug-inova* možemo koristiti sljedeću naredbu:

```
sos report -l
```

***Profili*** u *sos reportu* su skupovi konfiguracijskih postavki koje definiraju koje informacije treba prikupiti prilikom generiranja izvještaja. Svaki profil može sadržavati različite skupove modula i *plugin-a* koji će biti uključeni u izvještaj.
***Plugini*** su skripte ili moduli koji obavljaju specifične zadatke prikupljanja informacija o sustavu. Postoje *plugini* za različite dijelove sustava kao što su: mreža [`network`], kernel [`kernel`], memorija [`memory`], hardver [`hardware`], sustav za pohranu [`storage`], sigurnost [`security`], virtualizacija [`virt`] i mnogi drugi. Kada generirate izvještaj *sosreport* automatski pokreće odgovarajuće *plugin-e* za prikupljanje informacija koji su definirani u odabranom profilu.
Ako primjerice u sos izvještaju ne želimo *plugine:* `samba`, `selinux` i `soundcard`, možemo ih isključiti ovako:

```
sos report -n  samba, selinux,soundcard
```

**Kriptiranje i dekriptiranje sos izvještaja**
Moguće je i kriptirati (šifrirati) *sos izvještaj*, sa željenom lozinkom, primjerice ovako:

```
sos report --encrypt
```

Prilikom kriptiranja *sos izvještaja* sustav će nas pitati za način kriptiranja:

```
Specify encryption method [P]assphrase, [K]ey, [E]nv vars, [N]o encryption:
```

Mi želimo ovu datoteku kriptirati simetričnom lozinkom, pa upišite `P` i stisnite *ENTER*, a potom upišite i lozinku.
← Dobit ćemo novu datoteku *sos izvještaja* sa `.gpg` ekstenzijom, koja je kriptirana s našom lozinkom.
Sada su nam zbog *GPG* sustava potrebna dva programska paketa koja ćemo instalirati (za dekriptiranje):

```
yum -y install pinentry pinentry-tty
```

Dekriptiranje možemo napraviti ovako (u jednom retku):

```
gpg --output /tmp/decrypted-sosreport.tar.gz --decrypt /var/tmp/secured-sosreport-Rocky-9-2024-03-10-gkatids.tar.xz.gpg
```

I sada nas sustav traži lozinku s kojom smo kriptirali *sos izvještaj*, pa ju upišite nakon sljedećeg upita:

```
Please enter the passphrase for decryption:
```

**Prikrivanje osjetljivih podataka**
*Sos report* program nudi rutinu za prikrivanje potencijalno osjetljivih podataka, kao što su korisnička imena, imena računala, IP ili MAC adrese ili druge ključne riječi koje možemo odrediti. Pri tome izvorni *sos izvještaj* (`sos report`) ili `sos collect` ostaju nepromijenjeni, i kreira se nova datoteka (`*-obfuscated.tar.xz`).
Da bismo to napravili, prvo moramo imati prethodno izrađen *sos izvještaj*. a tek tada možemo pokrenuti:

```
sos report --clean
```

← Pri tome možemo koristiti sljedeće opcije:

- `--keywords` **`XY`** – ako želimo obrisati ključne riječi ili listu ključnih riječi (da se ne vide u izvještaju).
- `--usernames` **`XY`** – ako želimo obrisati točno određene korisničke račune (da se ne vide u izvještaju). Inače se kod ove metode (`--clean`) standardno obavlja brisanje ispisa svih korisničkih računa koji imaju **UID** broj veći od **1000**.

**Klaster**
U slučajevima kada imamo klaster u kojem se nalazi više poslužitelja, moguće je izraditi *sos izvještaj* i za njih, a za to se koristi naredba `sos collect`. U primjeru imamo tri poslužitelja u klasteru (`server-1`, `server-2` i `server-3`):

```
sos collect --nodes=server-1,server-2,server-3
```

← I ovdje možemo koristiti sve dostupne opcije i parametre koje želimo.

**Izvori informacija: (1599),(1600),(1601),(1602),(1603),(1604), `man sos`, `man sosreport`, `man sos-collect`, `man 5 sos.conf`.**

# 17.4. Datoteka `/var/loge/secure`

Datoteka `/var/log/secure` u Linuxu pohranjuje informacije o autentifikaciji i autorizaciji. To znači da se u nju zapisuju aktivnosti vezane za sigurnost odnosno podaci o prijavama korisnika, pokušajima prijave i drugim aktivnostima vezanim za kontrolu pristupa na sustav.

Neki od uobičajenih zapisa koje ćemo vidjeti u datoteci `/var/log/secure` su:

- Prijave ili odjave korisnika – to uključuje informacije poput korisničkog imena, vremena i datuma prijave ili odjave, te IP adrese s koje se prijava dogodila. Primjerice ovo je poruka prilikom odjave korisnika sa sustava:

```
Apr  6 09:24:27 Rocky2 sshd[13847]: pam_unix(sshd:session): session closed for
user root
```



Za detalje o **PAM** sustavu, pogledajte:
**7.1.6.6. Linux PAM (Pluggable Authentication Modules).**

- Pokušaji prijave – zapisuje se broj pokušaja prijave, korisničko ime i IP adresa s koje su se pokušaji prijave na sustav dogodili. Primjerice poruka poput:

```
Apr  6 09:21:53 Rocky2 sshd[13847]: Accepted password for root from
192.168.100.10 port 39955 ssh2
```

- Aktivnosti *root* korisnika ili privilegirane operacije – zapisuje se aktivnost *root* korisnika, uključujući i promjene konfiguracije sustava. Primjrice kada korisnik izvodi privilegirane operacije kao što su promjena privilegija (npr. prebacivanje na korisnika *root*), pokretanje programa koji zahtijeva više privilegije itd.
Primjerice ovo je jedna takva poruka:

```
Apr 6 10:40:20 localhost sudo: user1 : TTY=pts/0 ; PWD=/home/user1 ; USER=root ;
COMMAND=/bin/bash
```

← Ovdje vidimo aktivnost u kojoj korisnik (`user1`) koristi naredbu `sudo` te postaje korisnik `root`, a zatim izvršava naredbu `/bin/bash` odnosno pokreće *bash* ljusku.

Dok se ovdje vidi da korisnik `hrvoje` pomoću naredbe `sudo` pokušava promijeniti lozinku za korisnika `root`.

```
Apr  6 09:42:21 Rocky2 sudo[14009]:  hrvoje : TTY=pts/0 ; PWD=/home/hrvoje ;
USER=root ; COMMAND=/bin/passwd root
```



Za detalje o prebacivanju iz jednog korisnika u drugog, pogledajte poglavlje:
**7.1.5. Izvršavanje naredbi i prebacivanje rada s jednog na drugog korisnika.**

- Autentikacijski protokoli – zapisuju se informacije o korištenim autentikacijskim protokolima, poput PAM-a i SSH-a. Primjerice prilikom spajanja preko ***SSH*** protokola:

```
Apr  6 09:21:53 Rocky2 sshd[13847]: pam_unix(sshd:session): session opened for
user root(uid=0) by (uid=0)
```

- Greške autentikacije – zapisuju se greške pri autentikaciji, poput pogrešnih lozinki ili nepostojećih korisničkih imena. Primjerice:

```
Apr  6 10:32:15 localhost sshd[1234]: Failed password for invalid user testuser
from 192.168.100.10 port 54321 ssh2
```

**Izvori informacija: `man rsyslogd`,`man dmesg`**

# 17.5. Journald, journalctl i dnevničke (log) datoteke

Prethodne inačice ***Red Hat***-a, poput v.6. i starijih, koristile su tradicionalni sustav za upravljanje inicijalizacijom poznat kao **SysVinit**, odnosno takozvani *Init* koji je koristio ***(r)syslogd*** servis za dohvaćanje i spremanje dnevničkih (*log*) poruka sustava i aplikacija. Iako novije inačice *Red Hat* kompatibilnih distribucija Linuxa i dalje koriste ***(r)syslogd***, pošto se tradicionalni sustav za upravljanje inicijalizacijom **SysVinit** odnosno takozvani *Init* zamijenio s noviji ***systemd*** servisom, došlo je do promjena i po pitanju servisa zaduženog za rad s dnevničkim (*log*) datotekama. To se dogodilo od ***Red Hat 7+*** koji je uveo upotrebu inicijalizacijskog servisa ***systemd*** te je uveo i novi servis za rad s dnevničkim datotekama koji se zove ***journald***.

Naime, ***systemd*** je trebao naprednije funkcionalnosti od starog ***(r)syslog*** servisa koju je bilo teško dobiti kroz postojeće implementacije. *Journald* koristi datoteke za pohranu u binarnom formatu, a ne tradicionalne tekstualne datoteke kao što to radi *rsyslogd*. U skladu s ovim ciljem, jedna od glavnih promjena je bila zamjena jednostavnih tekstualnih datoteka s posebnim formatom datoteka optimiziranim za dnevničke (*log*) poruke. Ovaj format datoteke omogućuje učinkovitiji pristup relevantnim porukama. Ove binarne datoteke obično se nalaze u direktoriju `/var/log/journal/` i uobičajeno je da ih se pretražuje i čita pomoću posebnog programa odnosno naredbe `journalctl`.
U slučaju da navedeni direktorij ne postoji, možete ga kreirati i aktivirati na sljedeći način;

```
mkdir -p /var/log/journal
systemd-tmpfiles --create --prefix /var/log/journal
```

Pogledajmo popis osnovnih prednosti *journald* sustava u odnosu na tradicionalni *(r)syslogd*:

- **Indeksiranje** - *journald* koristi binarnu pohranu za zapise, gdje su podaci indeksirani. Pronalaženje traženih pojmova je puno brže nego s običnim tekstualnim datotekama.
- **Strukturirana pohrana** – iako je to moguće i sa *ryslogom*, ovdje je takav način zadan. Naime u kombinaciji s indeksiranjem, to znači da možete jednostavno filtrirati određene zapisnike (npr. s postavljenim prioritetom, u postavljenom vremenskom okviru i slično).
- **Kontrola pristupa** – prema zadanim postavkama, datoteke za pohranu odnosno pristup do njih podijeljen je i prema korisnicima, s različitim dopuštenjima za svakoga. Primjerice kao običan korisnik, nećete vidjeti sve što vidi *root* (administrator), ali ćete vidjeti svoje zapise.
- **Automatska rotacija dnevnika** – možete konfigurirati dnevnik da vodi zapise samo do ograničenja prostora ili na temelju slobodnog prostora diska, iako je ovo moguće na starom sustav, ali upotrebom dodatnog ***logrotate*** servisa.

Trenutno stanje (***Red Hat*** **v.9+)**, jest da je centralni servis za dohvaćanje sistemskih poruka i poruka aplikacija u dnevničke datoteke (tzv. *log* ili *zapisničke*) uz `rsyslogd` servis i novi servis `journald`, koji se preporuča koristiti, a koji će vrlo vjerojatno s vremenom i zamijeniti stariji servis i način zapisivanja dnevničkih (*log*) datoteka.
Važno je razumjeti da je *journald* sastavni dio ***Systemd*** servisa, a njegova konfiguracijska datoteka jest: `/etc/systemd/journald.conf`.

Pogledajmo samo neke od osnovnih opcija i pripadajućih vrijednosti koje možemo definirati o ovoj konfiguracijskoj datoteci:

- `Storage=` – ovdje se definira hoće li dnevničke (*log*) datoteke biti trajno pohranjene na disku (`persistent`) odnosno unutar direktorija `/var/log/journal/` ili u RAM memoriji (`volatile`) na lokaciji `/run/log/journal/`.
- `Compress=` – ovdje se definira hoće li dnevničke (*log*) datoteke biti komprimirane (`yes`) što je zadano, ili neće (`no`).
- `SplitMode=` – ovdje se definira hoće li dnevničke (*log*) datoteke biti razdvajane prema korisnicima što je zadano (`uid`) ili ako to izričito ne želimo, možemo postaviti na (`none`).
- `SystemMaxUse=` – ovdje se definira maksimalna ukupna veličina dnevničkih (*log*) datoteka na disku; primjerice `4G` označava maksimalnu ukupnu veličinu *log* datoteka od 4GB.
- `SystemKeepFree=` – ovdje se definira minimalna količina prostora na disku koja još smije ostati.

Za provjeru koliko zauzimaju sve dnevničke (*log)* datoteke na disku, možemo koristiti:

```
journalctl --disk-usage
```

Želimo li smanjiti ukupno zauzeće dnevničkih (*log*) datoteka brisanjem najstarijih, na primjerice 4GB, to možemo učiniti sa:

```
journalctl --vacuum-size=4GB
```

- `RateLimitIntervalSec=`, `RateLimitBurst=` – ovdje se definira ponašanje u kojem će *journald* izbaciti sve poruke pristigle od pojedinog servisa (ili programa) ako prijeđe određena ograničenja. Ako se, u vremenskom intervalu definiranom u `RateLimitIntervalSec=`, zabilježi više poruka nego što je navedeno u `RateLimitBurst=`, sve daljnje poruke unutar definiranog intervala se odbacuju, sve dok interval ne završi. Ova postavka ovisi i o dostupnom slobodnom prostoru na disku.

U slučaju ako radimo neke promjene u konfiguraciji (u navedenoj konfiguracijskoj datoteci), potrebno je osvježiti servis:

```
systemctl reload systemd-journald
```

Ili jednostavno možemo napraviti ponovno pokretanje servisa:

```
systemctl restart systemd-journald
```



Vezano za dnevničke (*log*) datoteke, pogledajte i poglavlja:
**17. Log datoteke.**
**25.8.9. (R)syslog servis i usluga.**

Za detalje o **journald** servisu i naredbi za čitanje tih poruka, pogledajte primjere u poglavljima:
**7.3.2.3. Osnovni servisi (*daemoni*) unutar *systemd* softverskog paketa.**

**Slijede primjeri upotrebe naredbe `journalctl` odnosno pretraživanja dnevničkih (*log*) datoteka.**
Pogledajmo primjere čitanja dnevničkih (*log*) datoteka pomoću programa `journalctl`:
1. Pogledajmo sve sistemske log datoteke (od trenutka inicijalizacije kernela, pa na dalje), sve u jednom ispisu:
```
journalctl
```

2. Pogledajmo samo kritične poruke na sustavu (ako ih ima).
Druge kategorije poruka (od najveće) su: `emerg`– važna upozorenja, `alert`– upozorenja, `crit`– kritično, `err`– greške, `warning`– klasična upozorenja, `notice`– obavijesti, `info`– informacije, `debug`– *debug* informacije.
```
journalctl -p crit
```

3. Pregled (*log*) poruka i to samo za servis *sshd* (SSH servis):
```
journalctl -u sshd
```

4. Pregled poruka samo za korisnika koji ima **UID** broj 1000:
```
journalctl _UID=1000
```

5. Pregled (*log*) poruka prema **PID** broju procesa; primjerice za **PID** broj 45878:
```
journalctl _PID=45878
```

6. Pregled (*log*) poruka s kontinuiranim praćenjem (kako se pojavljuju):
```
journalctl -f
```

7. Pregled (*log*) poruka za određeni uređaj, primjerice hard disk `/dev/sda`:
```
journalctl /dev/sda
```

8. Pregled (*log*) poruka kernela:
```
journalctl -k
```

9. Pregled (*log*) poruka kernela od određenog izvora (pr. servisa `systemd`):
```
journalctl -k -t systemd
```

10. Pregled (*log*) poruka vezanih za servis zadužen za mrežu i mrežna sučelja:
```
journalctl -u NetworkManager
```

11. Pregled (*log*) poruka koje sadrže točno određenu poruku „`Out of memory`“:
```
journalctl -g "Out of memory"
```

12. Pregled (*log*) poruka prilikom pokretanja sustava (*boot*):
```
journalctl -b
```

13. Pogledajmo (*log*) poruke o pokretanju sustava (kada je pokrenut ili restartan):
```
journalctl --list-boots
```

14. Pogledajmo (*log*) poruke koje su zabilježene između dva datuma (i vremena):
```
journalctl --since "2024-03-09 00:00:00" --until "2024-03-09 15:00:00"
```

15. Pogledajmo (*log*) poruke vezane za određenu naredbu (pr. za naredbu `sudo`):
```
journalctl _COMM=sudo
```

16. Za provjeru koliko zauzimaju sve dnevničke (*log*) datoteke na sustavu:
```
journalctl --disk-usage
```

17. Za smanjenje zauzeća dnevničkih (log) datoteka na disku na 4GB, i to automatskim brisanjem najstarijih *log* datoteka:
```
journalctl --vacuum-size=4GB
```

**Izvori informacija: (687),(688),(689),(690),(691),(1605),(1606),(K-14),** **`man systemd`**, **`man 8 systemd-journald.service`**, **`man journalctl`**, **`man journald.conf`**.

# 18. X Window sustav

***X Window*** sustav ponekad nazivan ***X11***, ***X*** ili ***X-Windows*** je grafičko sučelje (engl. ***GUI*** - *Graphical User Interface*) Unix i Linux sustava. Povijesno gledano, istraživanje upotrebe grafičkih elemenata za rad na računalu pokrenuto je u tvrtki ***Xerox*** unutar istraživačkog centra *Palo Alto Research Center* (***PARC***), početkom 1970. godine. *PARC* projekt je imao vrlo veliki utjecaj na razvoj prvog računala s upotrebljivim grafičkim sučeljem, dakle govorimo o ***Apple Lisa*** računalu te kasnije o ***Apple Macintosh*** te drugim računalima odnosno njihovim pripadajućim operativnim sustavima.
Razvoj *X Window* sustava započeo je 1984 od strane instituta *Massachusetts Institute of Technology* (***MIT***) pod oznakom protokola ***X11***. Većim djelom ovaj projekt su sponzorirale tvrtke ***DEC*** i ***IBM***. Godine 1986, tvrtke i institucije: ***MIT***, ***Apple***, ***AT&T***, ***DEC***, ***HP*** i ***SUN*** ulažu dodatna sredstva u daljnji razvoj projekta koji će voditi novoosnovani ***X*** konzorcij. Od 1987 do danas "*X.Org Foundation*": http://www.x.org/ je zadužen za njegov razvoj. Dakle ovaj sustav vizualno sliči grafičkom sučelju operativnih sustava ***Windows*** ili ***MAC*** **OS**. Važno je znati kako se svaki *X Window* sustav sastoji od nekoliko komponenti:

- ***X Window poslužitelja*** (*engl. X Server*) zaduženog za upravljanje hardverom, postavljenjem/preklapanjem prozora na radnoj površini, obradom ulaza (korisničkih), komunikacija s X klijentima, mrežna komunikacija i drugo.
- ***X Window klijenta*** (*engl. X Client*) zaduženog za prikaz grafičkog sučelja, interakciju s korisnikom, slanje zahtjeva *X Serveru* i komunikaciju s drugim klijentima, postavljanjem izgleda prozora (veličina/položaj/boje/ …).
- ***X Window Managera*** zaduženog za upravljanje prozorima i radnom površinom, događajima, …

*X Window* sustav koristi model: klijent - poslužitelj. To znači da *X Server* komponenta komunicira s raznim *X klijentskim* programima. S druge strane sâm ***X11*** protokol je zamišljen kao mrežni protokol. To znači da nije nužno da sve ono što bi trebali vidjeti na lokalnom monitoru odnosno zaslonu, nužno mora biti sâmo lokalno, već bilo gdje na mreži, sve dok s druge strane mreže postoji računalo koje ima instaliranu *X Server* komponentu koja ima kontrolu nad monitorom, mišem i tipkovnicom tog udaljenog računala. *X Server* omogućava spajanje više *X* klijenata (aplikacija) istovremeno i to svakog u svom zasebnom prozoru, što danas zvuči uobičajeno, ali sjetimo se od kada se ovaj sustav dizajnirao i razvijao. X klijent je zapravo bilo koja aplikacija koja treba koristiti resurse *X Servera*.
*X* klijent šalje *X Serveru* (poslužitelju) zahtjeve poput:

- Iscrtaj prozor na koordinatama: X,*Y ili* pomakni prozor s $X_1$,$Y_1$ koordinata na koordinate $X_2$,$Y_2$ i slično.

Dok s druge strane poslužitelj (*X Server*) vraća klijentu sve ostale interakcije poput: korisnik je kliknuo na dugme "*OK*".
Dodatno komunikacija nije ograničena sâmo na jedan (ili više) klijenata odnosno aplikacija prema jednom *X Serveru*, već je moguće imati razne kombinacije. Moguće je da se nekoliko X klijenata spaja na jedan *X Server* (pr. lokalno računalo), dok se istovremeno nekoliko drugih X *Window* klijenata odnosno aplikacija spaja ne neki drugi udaljeni *X Server* (negdje na mreži).
Prema tome, vrlo je važna komponenta unutar *X Window* sustava ova mrežna komponenta koja povezuje *X Server* i *X klijent* te koja mu daje vrlo veliku fleksibilnost u radu. Važno je i to da mrežna komunikacija nije razvijena naknadno, već ona pripada osnovnim funkcionalnostima ***X11*** protokola. Međutim *X Server* nije zadužen za iscrtavanje ičega na ekranu. On sve zahtjeve za konačno iscrtavanje prosljeđuje sljedećoj komponenti *X Window* sustava, a to je komponenta koja se zove ***Window Manager***. *Window Manager* je taj koji određuje na koji način će se prozor iscrtavati, pomicati i mijenjati te kako će u konačnici i izgledati. Kao i sve ostale detalje: kako će prozor uopće izgledati ili kako će se prozori povećavati, smanjivati, pomjerati i slično. U konačnici on je zadužen za sve ono što vidimo na zaslonu našeg monitora. *Window Managera* postoji na desetke, neki su manji i brži a drugi su veći, ali nude i napredne mogućnosti. Neki od njih vizualno izgledaju poput *Microsoft Windows*-a raznih inačica, neki poput *MAC OS*-a, dok su drugi potpuno drugačiji, a neki opet, poput kombinacije navedenih.
Pogledajmo listu sâmo nekih od češće korištenih *Window Managera*:

- ***Blackbox*** (http://blackboxwm.sourceforge.net/)
- ***Compiz*** (https://launchpad.net/compiz)
- ***Enlightenment*** (https://www.enlightenment.org/)
- ***Fvwm*** (http://www.fvwm.org/)
- ***Fluxbox*** (http://www.fluxbox.org/)
- ***IceWM*** (http://www.icewm.org/)
- ***GNOME*** (https://www.gnome.org/)
- ***KDE*** (https://www.kde.org)
- ***Openbox*** (http://openbox.org)
- ***TWM (Tab Window Manager) (https://en.wikipedia.org/wiki/Twm)***
- ***Window Maker*** (http://windowmaker.org/)
- ***Xfce*** (http://www.xfce.org/)
- ***LXDE*** (http://lxde.org/) i mnogi drugi

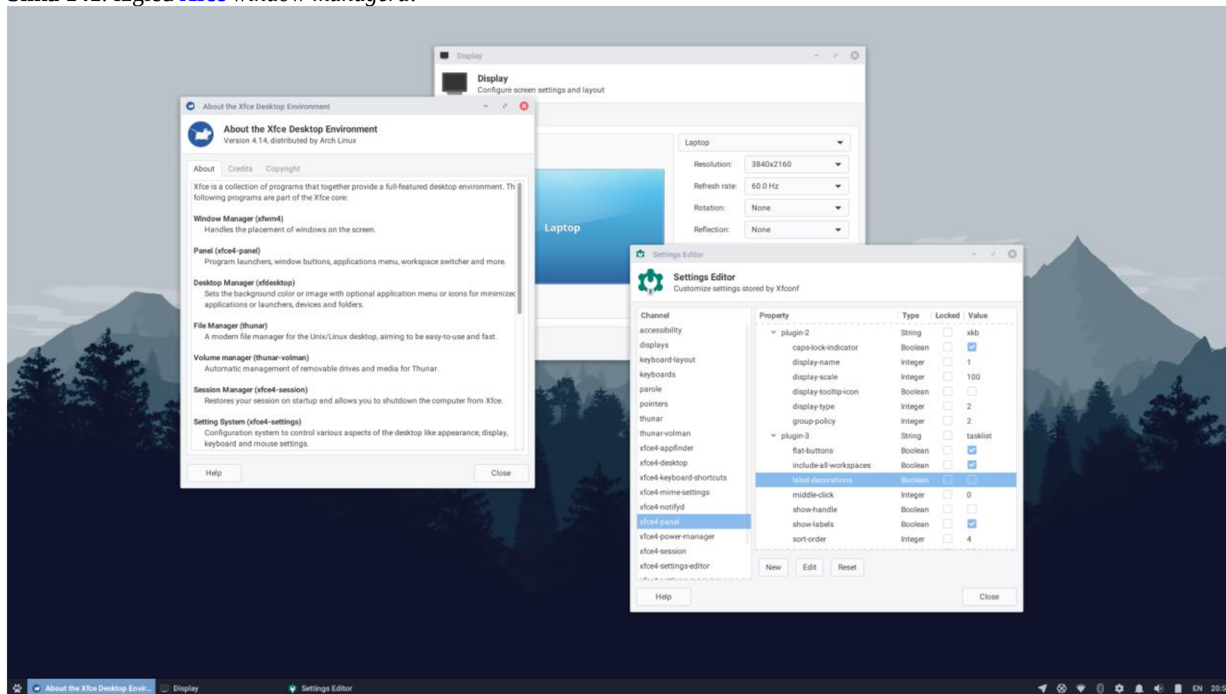
Zanimljivo je i to da na svaki Linux sustav možemo instalirati više *Window Managera* te da si svaki korisnik može konfigurirati koji od njih će koristiti.



U novije vrijeme uz ***Xorg*** (*X11*) kao *X Window* sustav, pokrenut je i po mnogima moderniji projekt imena ***Wayland*** koji na poprilično drugačiji način obavlja istu funkcionalnost. Dakle on koristi znatno moderniji i čišći dizajn te lakši način razvoja. On na mala vrata (2019.g.) ulazi u primjenu, za sada kao „probni" *X Window* sustav u nekim distribucijama Linuxa.

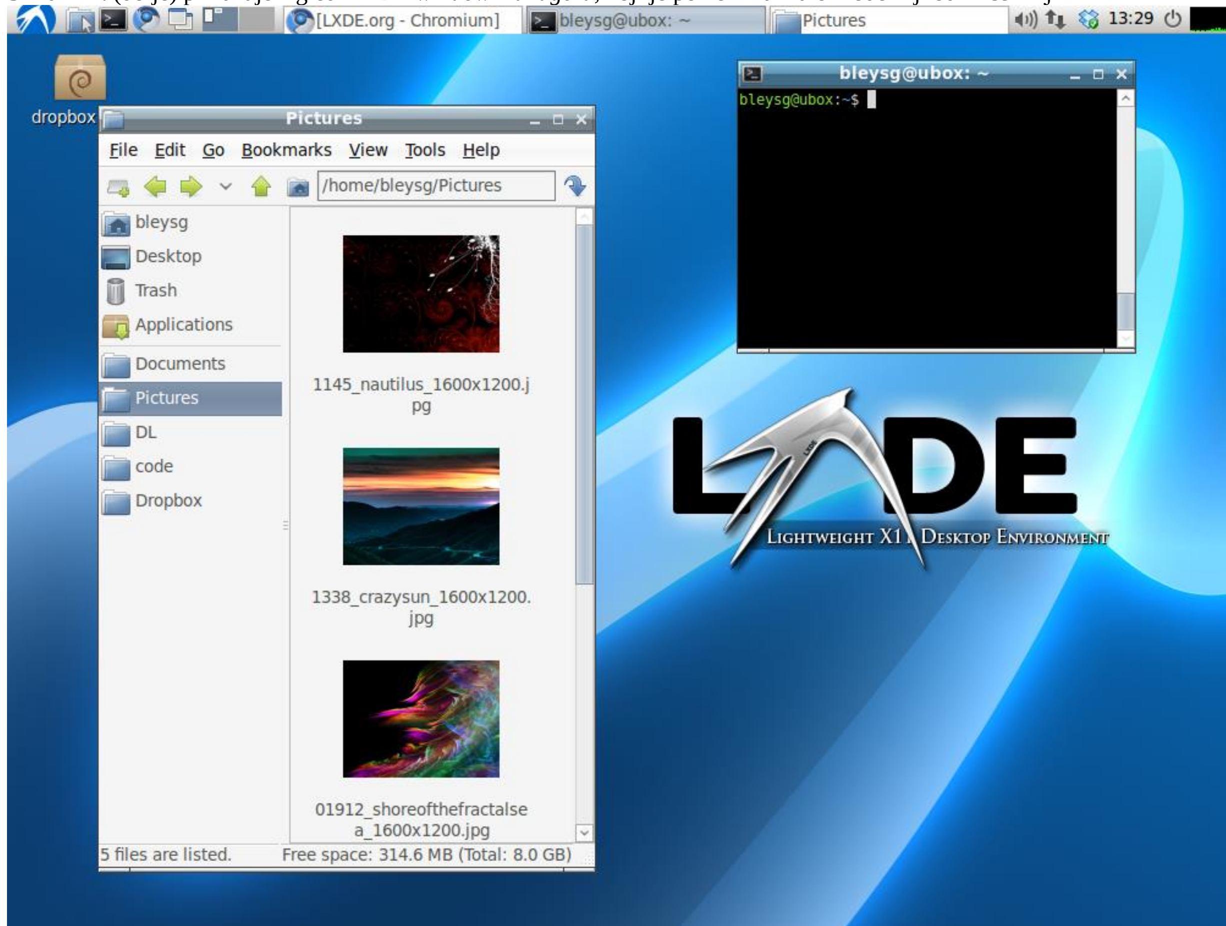
Pogledajmo kako izgledaju neki od ***X Window*** sustava odnosno grafičkih sučelja za Linux, na slikama: **(141),(142),(142.1)** i **(142.2)**.

**Slika 141**. Izgled **Xfce** *window managera*.



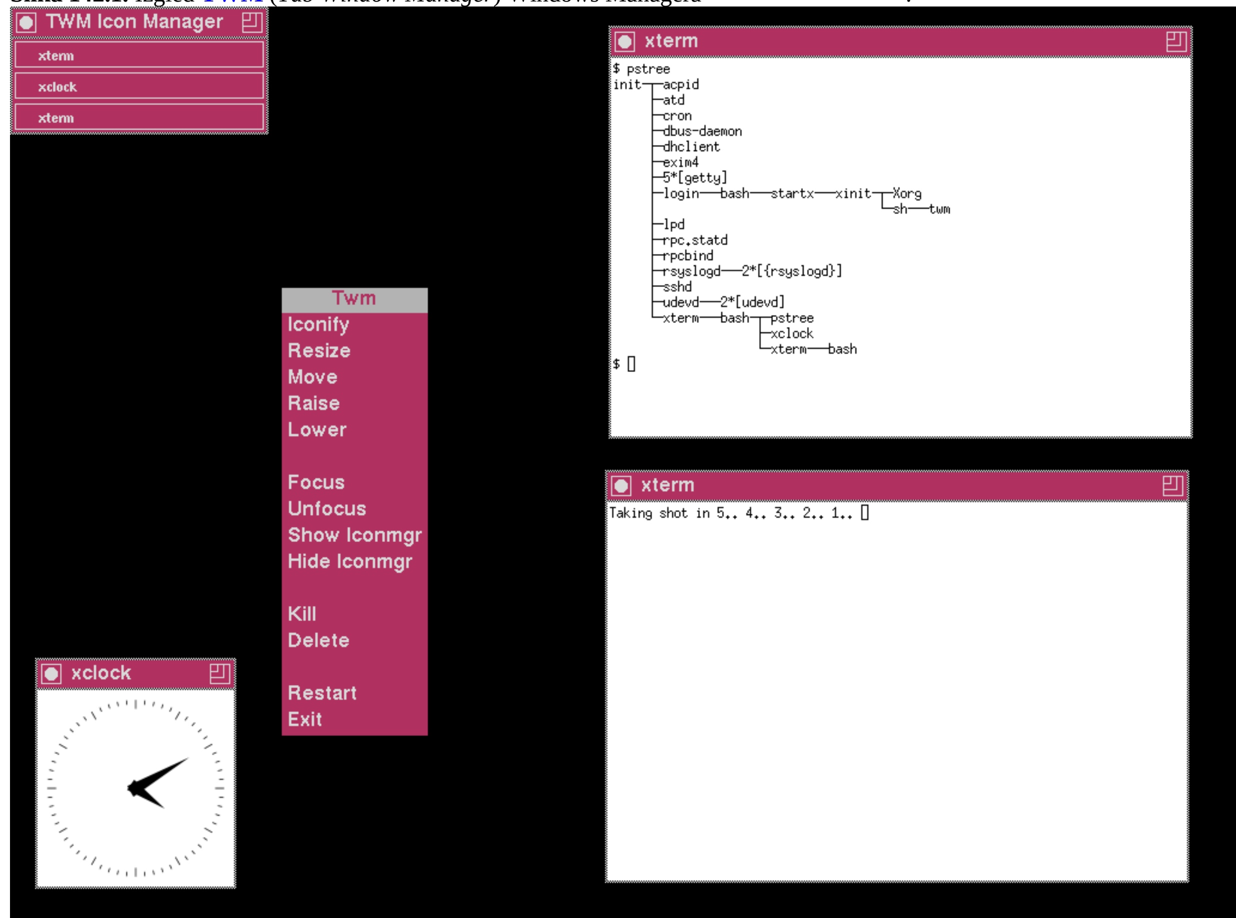

Izvor slike: https://xfce.org/about/screenshots

**Slika 142.** (dolje) prikazuje izgled **LXDE** *window managera*, koji je po nekima malo moderniji od **Xfce** ali je zadržao brzinu:



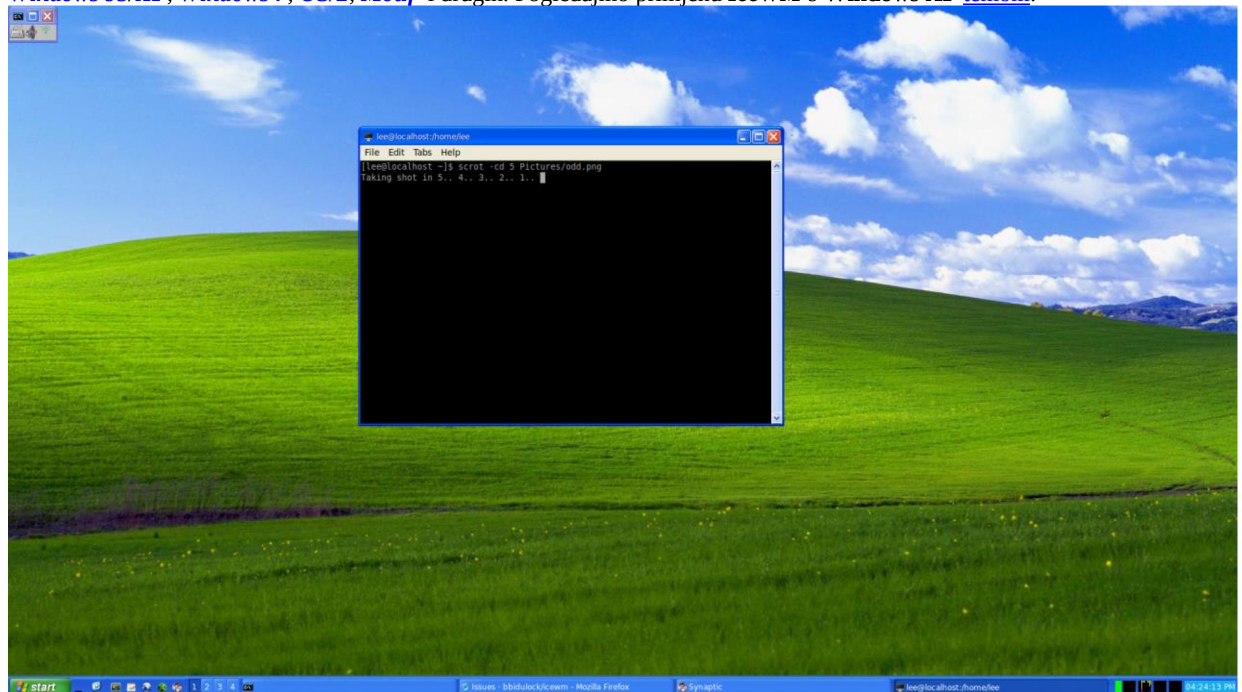

Izvor slike: https://wiki.lxde.org/en/File:LXDE_desktop_full.png

**Slika 142.1.** Izgled **TWM** (*Tab Window Manager*) Windows Managera(koji se razvija od 1987 godine).




Izvor slike: https://en.wikipedia.org/wiki/File:Debian_TWM_Maroon.png

**Slika 142.2.** Izgled **IceWM** window managera, koji je napravljen da pomoću takozvanih tema može imitirati grafička sučelja: ***Windows 95/XP***, ***Windows 7***, ***OS/2***, ***Motif*** i drugih. Pogledajmo primjenu ***IceWM*** s **Windows XP** temom:



Izvor slike: https://ice-wm.org/screenshots/

**Izvori informacija: (692),(693),(694)**

# Računalne mreže i Linux

# 19. Mrežni sustav

Da bismo se upoznali s mrežama, prvo se moramo upoznati s mrežnim protokolima i modelima prema kojim su nastali, kao i s osnovnim standardima vezanim uz mreže.

**U ovom poglavlju o mrežama proći ćemo uvod u osnove računalnih mreža te osnove i napredne dijelove mreže u Linuxu.** Međutim prvo se upoznajmo s najraširenijom mrežnom tehnologijom koja se zove ***Ethernet***.

## Ethernet

***Ethernet*** je naziv tehnologije za umrežavanje računala koja se obično koristi u lokalnim mrežama (**LAN**), mrežama za gradska područja (**MAN**), ali i mrežama koje pokrivaju široka zemljopisna područja (**WAN**). Komercijalno je uveden 1980. godine a prvi put je standardiziran 1983. godine kao ***IEEE 802.3***. Ethernet je od tada poboljšan da podržava veće brzine prijenosa, povezivanje većeg broja čvorova (računala i mrežnih uređaja) te znatno veće udaljenosti, ali je zadržao dosta kompatibilnosti sa starijim inačicama. Tijekom vremena, *ethernet* je u velikoj mjeri zamijenio konkurentske žične LAN tehnologije kao što su ***Token Ring***, ***FDDI*** i ***ARCNET***.
U današnje vrijeme *Ethernet* je najkorištenija tehnologija u lokalnim mrežama s iznimkama upotrebe: na superračunalima koja za umrežavanje osim ***Etherneta*** koriste i ***InfiniBand***, a ponekad i ***OmniPath***, te u **SAN** sustavima za pohranu (*Storage Area Network*) gdje se često koristi ***Fiber channel*** (tzv. *Fibre channel*) tehnologija (***poglavlje. 13.12.1.1.***).

### Povijest razvoja

*Ethernet* je razvijen u tvrtki **Xerox PARC** između 1973. i 1974. godine. Inspiriran je ***ALOHAnetom***, koji je *Robert Metcalfe* proučavao kao dio svoje doktorske disertacije. Izvorni *ethernet* radio je na 2.94 Mbit/s (Mbps). Godine 1975. godine tvrtka ***Xerox*** je patentirala ovu tehnologiju koju je razvio navedeni autor. Metcalfe je napustio **Xerox** u lipnju 1979. kako bi osnovao tvrtku **3Com**. Pri tome je uvjerio tvrtke ***Digital Equipment Corporation*** (**DEC**), ***Intel*** i ***Xerox*** da zajedno rade na promoviranju *Etherneta* kao standarda. Kasnije je s nekolicinom dugih napravio i poboljšanu inačicu *etherneta* koja je radila na 10Mbps, tijekom 1980. godine, koji je potom iskorišten kao standard. Kao dio procesa bolje promocije *Xerox* je pristao odreći se svojeg registriranog zaštićenog (patentiranog) naziva *'Ethernet'*.
Prvi standard objavljen je 30. rujna 1980. kao "*The Ethernet, A Local Area Network. Data Link Layer and Physical Layer Specifications*". Ovaj takozvani ***DIX*** standard (*Digital Intel Xerox*) specificirao je 10 Mbit/s *Ethernet*, s 48-bitnim izvorišnim i odredišnim adresama (tzv. **MAC** adrese) te globalnim 16-bitnim poljem naziva ***Ethertype***.
Tijekom 1980. godine, Institut inženjera elektrotehnike i elektronike (**IEEE**) započeo je projekt **802** za standardizaciju lokalnih mreža (*LAN*). Zatim je ***DIX*** radna skupina podnijela takozvanu "*Blue Book*" **CSMA/CD** specifikaciju kao kandidata za LAN specifikaciju. Inačica 2 objavljena je u studenom 1982. godine i definirala je ono što je postalo poznato kao **Ethernet II**.
Standard IEEE **802.3** CSMA/CD odobren je u prosincu 1982. godine, a IEEE je objavio standard 802.3 kao nacrt 1983. godine te kao finalni standard 1985. godine. Ethernet se u početku natjecao s *Token Ringom* i drugim vlasničkim protokolima.
Ipak, *Ethernet* se uspio prilagoditi potrebama tržišta počevši s **10BASE5** varijantom koja je koristila *debeli* koaksijalni kabel (9,5.mm. u promjeru**)**, prelazeći na jeftiniji *tanki* koaksijalni kabel s **10BASE2** (upotrebom **RG-58** kabela). Dok se od 1990. godine do sada koristi upotrebom sveprisutne prepletene (uvijene) parice, prvo s upotrebom **10BASE-T** te novijih standarda.

U oznakama **10BASE5**, **10BASE2** ili **10BASE-T** broj **10** označava brzinu od 10Mbps, **BASE** način signalizacije (*baseband*), dok za koaksijalne kablove **5** označava maksimalnu duljinu kabela od 500.m. to jest broj **2** duljinu od 200.m. Dok oznaka **T** znači da se za kabliranje ne koristi koaksijalni kabel već prepletena parica (engl. *twisted pair*) odnosno takozvani **UTP** kabel.

Do kraja 1980-ih, *Ethernet* je postao dominantna mrežna tehnologija. Naime, *Ethernet* se izvorno temeljio na ideji da računala komuniciraju preko zajedničkog koaksijalnog kabela koji djeluje kao medij za prijenos podataka (paketa). Korištena metoda bila je slična radijskim sustavima, primijenjeno ovdje: sa zajedničkim kabelom koji je osiguravao komunikacijski kanal koji se uspoređuje s *luminiferskim eterom* u fizici 19. stoljeća, a iz te je reference izveden naziv "*Ethernet*". S vremenom se *Ethernet* razvijao kako bi omogućavao što veću propusnost: danas čak do 400Gbit/s [Gbps], poboljšane metode kontrole pristupa medijima i upotrebu različitih fizičkih medija: od početka razvoja koaksijalni kabel je zamijenjen fizičkim vezama upotrebom *paričnih* kabela, a za povezivanje su se koristili *Ethernet* repetitori (tzv. ***HUB-ovi***), a kasnije i preklopnici (engl. *switch-evi*).
U novije vrijeme sve više se koriste i optička vlakna i pripadajuća optička mrežna oprema (o standardima kasnije).

### Način rada

Računala i mrežni uređaji (stanice) u komunikaciji upotrebom *Ethernet* tehnologije komuniciraju šaljući jedan drugom pakete podataka: odnosno blokove podataka koji se pojedinačno šalju i isporučuju. Kao i kod drugih IEEE 802 LAN tehnologija, mrežna sučelja (tzv. mrežne kartice) dolaze programirane s jedinstvenom 48-bitnom adresom (tzv. **MAC** adresa) tako da svaka *Ethernet* stanica ima jedinstvenu adresu. MAC adrese se koriste za određivanje odredišta i izvora svakog paketa podataka. To znači da *Ethernet* uspostavlja veze koje se mogu definirati pomoću odredišne i izvorišne adrese. Po prijemu podataka, prijamna strana koristi odredišnu adresu kako bi utvrdila je li prijenos relevantan za nju ili ga treba zanemariti. Naime, mrežno sučelje (mrežna kartica) obično ne prihvaća pakete upućene drugim *Ethernet* stanicama, već samo one upućene sebi (prema svojoj **MAC** adresi).
Dok navedeno polje ***EtherType*** u svakom mrežnom okviru (paketu) koristi operativni sustav na prijemnoj strani za odabir odgovarajućeg protokola više razine (npr. inačice internetskog protokola kao što je **IPv4**). Upravo zbog polja ***EtherTyp***e, kaže se da su *Ethernet* okviri *samoidentificirajući*. Ovakvi *samoidentificirajući* okviri omogućuju upotrebu više mrežnih protokola na istoj fizičkoj mreži te omogućuju svakom (pojedinom) računalu da koristi više mrežnih protokola istovremeno.

***Za sve detalje o načinima razvoja, rada i povezivanja uređaja pomoću Ethernet tehnologije, pogledajte poglavlja koja slijede!***

**Izvori informacija: (44),(1511),(K-6),(K-7),(K-10).**

# 19.1. Topologija mreže

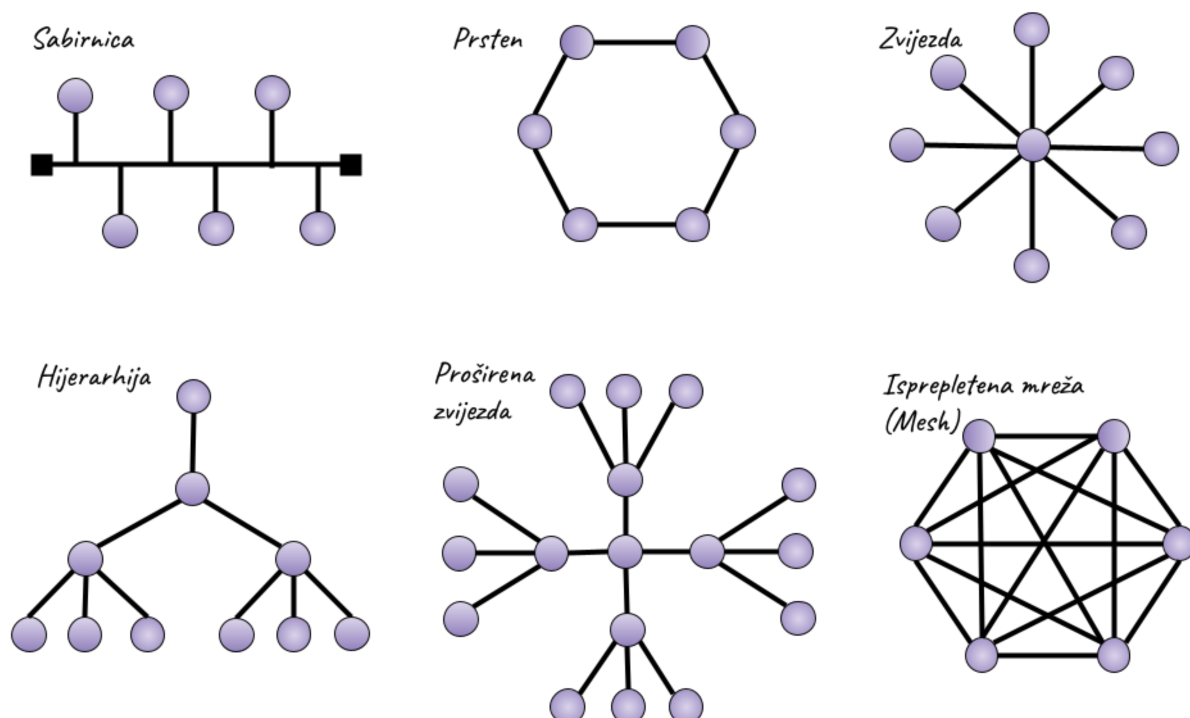
Mrežna topologija definira strukturu sâme mreže odnosno način međusobnog spajanja svih mrežnih komponenti i uređaja (pr. računala, poslužitelja i sl.), ako pričamo o fizičkoj topologiji. Logička topologija daje nam uvid u logičke veze između svih mrežnih komponenti i uređaja.

Prema fizičkoj topologiji razlikujemo sljedeće načine međusobnog povezivanja:

- ***Sabirnica*** (engl. *Bus*) koristi dijeljenu vezu. Svi su mrežni elementi (čvorovi) spojeni direktno na dijeljenu vezu odnosno sabirnicu. Ovakav način povezivanja koristio se upotrebom koaksijalnog kabela kao dijeljenog medija za umrežavanje kod 10Mbps *Ethernet* mreža (**10BASE5** i **10BASE2** *Ethernet*).
- ***Prsten*** (engl. *Ring*) – mrežni elementi se spajaju jedan na drugi (serijski), te zadnji element s prvim, što kreira zatvoreni prsten. Ovakav spoj se često koristi zbog redundancije (zalihosti) jer je kod njega dozvoljen ispad bilo kojeg (pojedinog) mrežnog elementa (mrežnog uređaja, računala, poslužitelja i sl.), bez ispada cijele mreže.
- ***Zvijezda*** (engl. *Star*) – kod ovakvog spoja svi mrežni elementi se spajaju na jednu centralnu točku. Ovako se spajaju mrežne komponente i uređaji pomoću ***HUB***-ova ili u današnje vrijeme pomoću preklopnika (*switcheva*). Kod ovakve veze nema dijeljenja mrežnog medija pa je samim time komunikacija brža i stabilnija (bez ***kolizije***).
- ***Proširena zvijezda*** (engl. *Extended Star*) - ovo je spoj u kojem se više topoloških zvijezda spaja u jednu točku. Ovako se spajaju mrežne komponente i uređaji pomoću preklopnika (*switcheva*), usmjerivača (*routera*) i drugih mrežnih elemenata (*vatrozida*, *IDS*/*IPS* sustava, *Load Balancera*, … ) u većim mrežama.
- ***Hijerarhijska topologija*** slična je proširenoj zvijezdi, ali mora se sastojati od minimalno tri hijerarhijska sloja: sloja *jezgre* mreže (Engl. *Core*), *distribucijskog* sloja (Engl. *Distribution*) i *pristupnog* sloja (Engl. *Access*), pri čemu svaki hijerarhijski sloj mreže ima svoju posebnu zadaću odnosno namjenu.
- ***Isprepletena mreža*** (engl. *Mesh*) koristi se kako bi omogućila najveću sigurnost od ispada bilo kojeg dijela ili komponente u mreži jer uvijek postoji više puteva do svake točke u mreži. Ovakvu topologiju najčešće koriste telekomi.

Slika 143 prikazuje gore navedene vrste mrežnih topologija.

**Slika 143. Topologije mreže**




**Izvori informacija: (1045), (K-10).**

### 19.1.1. Detaljnije o topologijama mreže

Topologija mreže odnosi se na strukturni raspored mreže.
U topologiji, mrežni uređaji su obično prikazani kao čvorovi, a veze između uređaja kao linije za izgradnju grafičkog modela.
Drugim riječima, topologija mreže označava način na koji je mreža uređena odnosno kako su čvorovi postavljeni u strukturi i međusobno povezani.
Potreba za razumijevanjem mrežnih topologija je važna jer (vaša) mreža može biti uređena na više različitih načina, pre čemu svaki od načina spajanja to jest topologija ima svoje prednosti i nedostatke. Na izbor topologije mreže utječu brojni čimbenici, od kojih su najvažniji veličina i opseg mreže kao i njena cijena. Međutim, potrebno je uzeti u obzir i dugoročne čimbenike, uključujući upravljanje konfiguracijom, praćenje i opću izvedbu mreže. Ključno je imati jasno razumijevanje topologije mreže jer će vam to omogućiti da odaberete onu koja najbolje odgovara vašim ciljevima i poslovnim zahtjevima.
Odabir prave mrežne topologije može vam pomoći da:

- Smanjite troškove rada i održavanja mreže.
- Povećajte performanse mreže.
- Osigurate optimalan rad mreže učinkovitom raspodjelom resursa.
- Brže locirate i otklanjajte pogreške u radu mreže, ali i cijelog IT sustava.

Topološka struktura mreže može se prikazati fizički ili logički.
Dakle postoje dvije osnovne kategorije mrežnih topologija: fizička topologija i logička topologija.

**Fizička topologija mreže**
Fizička topologija prikazuje fizički izgled vaše mreže. Odnosi se na smještaj različitih mrežnih uređaja kao što su: usmjerivači (*routeri*), preklopnici (*switchevi*), vatrozidi (*firewalli*), bežične pristupne točke (*Access point*), računala i drugo, uključujući metodu koja se koristi za povezivanje tih uređaja, tj. mrežnih kabela. To se odnosi na raspored kabela, mjesta čvorova i veza između čvorova i kabliranja. Poznavanje fizičke topologije vaše mreže važno je jer vam pomaže u planiranju proširenja, održavanju i zadacima osiguravanja sigurnog i pouzdanog rada mreže.

**Logička topologija**
Logička topologija odnosi se na logiku kako podaci prolaze unutar mreže. Ona opisuje kako je mreža postavljena, kako su čvorovi uključujući virtualne i resurse u oblaku, međusobno povezani i kako se podaci prenose kroz mrežu. Dakle logička topologija opisuje način na koji signali prolaze kroz mrežni medij ili način na koji podaci prolaze kroz mrežu s jednog uređaja na drugi, bez obzira na fizičku povezanost uređaja. Dobro razumijevanje logičke topologije ključno je za učinkovito upravljanje mrežom i njen nadzor, što osigurava učinkovitost i uredan (nesmetan) rad mreže.

**(Međusobne) veze**
Veza se odnosi na prijenosni medij koji se koristi za povezivanje čvorova u vašoj mreži. *Veza* je poznata kao *Ethernet* i obično uključuje *ethernet* kabele, optička vlakna i/ili bežično umrežavanje. Kao takve, *veze* spadaju u sljedeće kategorije:

- Žičane tehnologije, koje obuhvaćaju i optičke tehnologije povezivanja.
- Bežične tehnologije.

**Mrežni čvorovi**
Mrežni čvor je krajnja točka ili točka redistribucije mreže, koja može primati, stvarati, pohranjivati ili slati podatke duž mrežnih ruta do drugih mrežnih čvorova. Dakle mrežni čvorovi su točke povezivanja medija za prijenos (podataka/signala) s odašiljačima i prijemnicima električnih, optičkih ili radijskih signala koji se prenose u mediju.
U vašoj topologiji mreže, čvorovi su obično uređaji koji su povezani **vezama**.

Uobičajeni čvorovi koji se koriste u izgradnji računalne mreže su sljedeći:

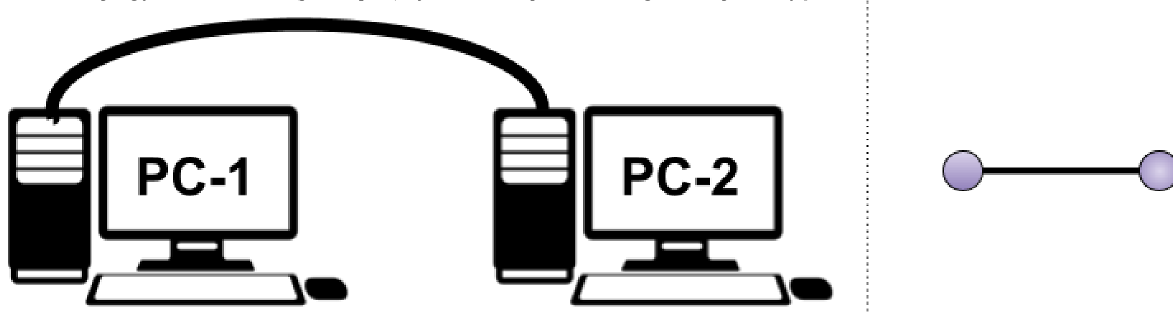
- **Mrežne kartice** (mrežna sučelja) odnosno kontroler mrežnog sučelja (Engl. ***NIC***) je računalni hardver koji računalu daje mogućnost pristupa mrežnom mediju i ima sposobnost obrade mrežnih informacija niske razine. Na primjer, mrežna kartica (**NIC)** može imati konektor za prihvaćanje kabela, ili antenu za bežični prijenos i prijem, te pripadajući elektronički sklop. Mrežna kartica odgovara na promet adresiran na mrežnu adresu namijenjenu za nju ili računalo u cjelini. U *Ethernet* mrežama, svaki kontroler mrežnog sučelja (mrežna kartica) ima jedinstvenu adresu kontrole pristupa mediju (tzv. **MAC** adresu), obično pohranjenu u trajnoj memoriji mrežnog kontrolera. Važno je razumjeti da kada povežete svoje računalo na mrežu ili prikažete računalo kao mrežni čvor, računalo nije ono koje funkcionira kao čvor. Upravo je mrežna kartica (**NIC**) unutar vašeg računala ona koja obavlja ovu funkciju.
- **Repetitor** (***repeater***) i **koncentrator** (***HUB***): repetitor je mrežni uređaj koji regenerira bežični (Wi-Fi) signal preko iste mreže. On prima slabe ili oštećene Wi-Fi signale i regenerira ih s izvornom snagom. ***Hub*** je samo repetitor s više mrežnih sučelja i obično se koristio u žičanim mrežama (10-100Mbps *Ethernet*).
- **Usmjerivač:** usmjerivači (*routeri*) su mrežni uređaji koji prenose pakete podataka između mreža. On prosljeđuje pakete podataka između mreža obradom informacija o usmjeravanju (na OSI 3 sloju) uključenih u te pakete.
- **Preklopnik:** preklopnik (*switch*) je mrežni uređaj koji omogućuje drugim uređajima na mreži da komuniciraju i dijele informacije. Drugim riječima, preklopnik povezuje različite uređaje kao što su računala, pisači, poslužitelji i druga oprema, zajedno u mrežu.
- **Mrežni most (***bridge***)** je mrežni uređaj koji filtrira te povezuje promet između dva mrežna segmenta.
- **Vatrozid (***firewall***)** je mrežni sigurnosni uređaj ili sustav, koji prati i kontrolira dolazni i odlazni mrežni promet.

## Uobičajene vrste mrežnih topologija

Uobičajene vrste mrežnih topologija su sljedeće.

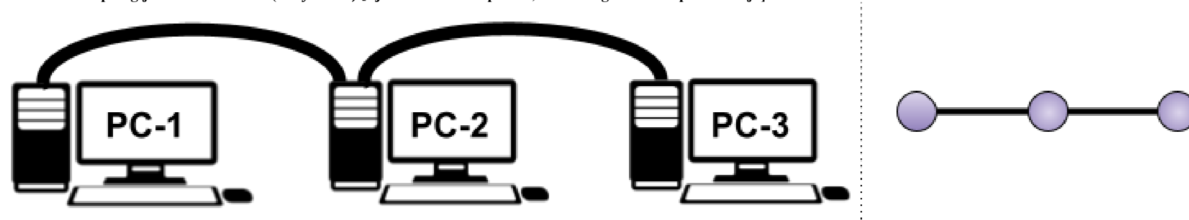
**1. Točka-točka (**Engl. *Point-to-point***):** kao što ime sugerira, *točka-točka* je mrežna topologija s namjenskom vezom između dvije krajnje točke (pr. računala ili mrežnih uređaja), stoga je to najjednostavnija topologija. Prednost takve mreže je što je sva dostupna propusnost mreže namijenjena dvama povezanim uređajima, koji ju dijele. Međutim, nije vjerojatno da ćete koristiti topologiju točka-točka u svojoj uredskoj ili kućnoj mreži. Ovakva topologija je vidljiva na slici **143.1**.

**Slika 143.1 Topologije mreže: točka-točka (point-to-point) [lijevo: simbolički prikaz, desno: logički način povezivanja]**



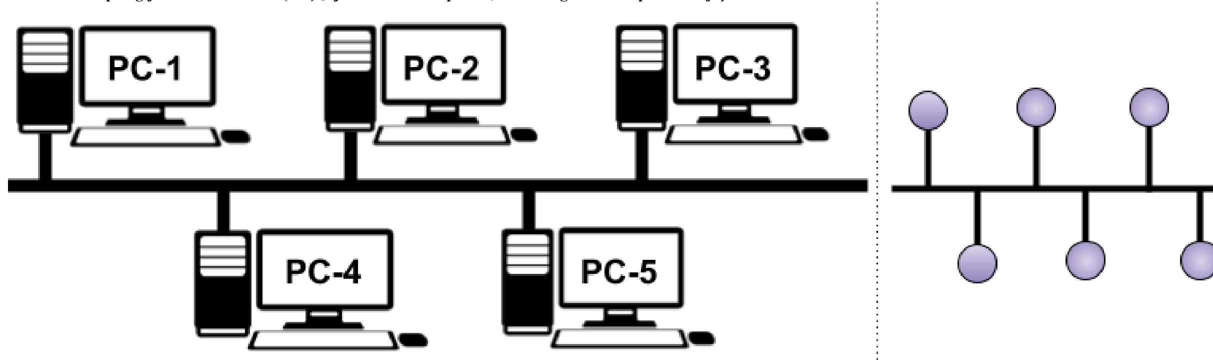

**2. Linearna veza i prstenasta veza: linearna veza (**znana i kao *Daisy chain*) je još jedna jednostavna mrežna topologija koja se stvara serijskim povezivanjem svakog čvora (pr. računala ili mrežnog uređaja). Kada se podaci prenose u mreži s linearnom vezom, svaki čvor ih prebacuje na slijedeći u nîzu, dok podaci ne stigne na odredište. Mreža u ovakvom spoju može biti u dva osnovna oblika: **linearna** (prikazano na slici **143.2**) i **prstenasta**, o kojoj ćemo govoriti kasnije.

**Slika 143.2 Topologije mreže: linearna (*daisy chain*) [lijevo: simbolički prikaz, desno: logički način povezivanja]**




**3. Sabirnička (**Engl. *Bus***)** odnosno topologija sabirnice sastoji se od jednog kabela, koji se također naziva sabirnica, koji se proteže od jednog do drugog kraja mreže. U ovakvom mrežnom spoju, svaki čvor (pr. računalo ili mrežni uređaj) je spojen na središnji kabel ili sabirnicu preko konektora sučelja za spoj na sabirnicu. Signal, koji sadrži adresu i podatke, koji se prenosi iz izvornog čvora putuje u oba smjera do svih čvorova dok ne dođe do odredišnog čvora, koji prihvaća podatke.
Ako se adresa isporučenog signala ne podudara s adresom prijemnog čvora, zanemaruje se podatkovni dio signala.
Pogledajte skicu ovakvog načina rada, na slici **143.3**.

**Slika 143.3 Topologije mreže: sabirnica (*bus*) [lijevo: simbolički prikaz, desno: logički način povezivanja]**




Prednosti ovakvog spoja su:

- Jednostavna izgradnja i cijena, u usporedbi s drugim mrežnim topologijama.

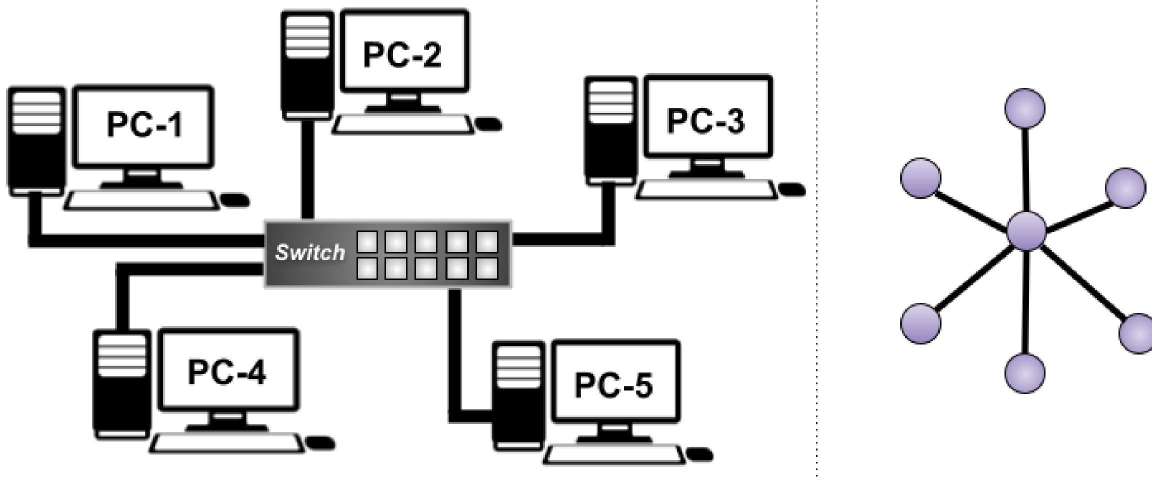
Mane ovog načina povezivanja:

- Dodavanje novih čvorova u mrežu je lakše i može se postići jednostavnim spajanjem dodatnih kabela s konektorima, na sabirnicu. Budući da cijela mreža ovisi o jednom kabelu za prijenos podataka, ako se taj kabel pokvari, cijela mreža neće raditi. Takva pojedinačna točka kvara nije idealna jer može uzrokovati puno zastoja i bit će skupa za obnavljanje.
- Topologije sabirnice se prikladno koristiti za male mreže u kojima nema previše uređaja, međutim veće mreže s velikom količinom prometa će patiti od ekstremno malih brzina prijenosa.
- Rješavanje i lociranje problema kod ovakvih mreža će biti dugotrajno za imalo veće mreže.

Ovakav spoj se u današnje vrijeme u pravilu više ne koristi, a koristio se prije, kada su se uređaji povezivali pomoću koaksijalnog kabela kao dijeljenog medija za umrežavanje kod 10Mbps *Ethernet* mreža (**10BASE5** i **10BASE2** *Ethernet*).

**4. Zvijezdasta (**Engl. *Star*) topologija je ona u kojoj je svaki periferni čvor (pr. računalo ili mrežni uređaj) povezan sa središnjim čvorištem ili preklopnikom. To je najčešće korištena mrežna topologija za ***LAN*** mreže jer se smatra najlakšom topologijom za dizajn i implementaciju, a osim toga na ovaj način se umrežavaju čvorovi (računala, poslužitelji, pisači i sl.) upotrebom preklopnika (*switcheva*). Središnje čvorište funkcionira kao poslužitelj za periferne čvorove ili klijente.
Sav mrežni promet prolazi kroz središnje čvorište i to je jedini uvjet da se topologija klasificira kao topologija zvijezda; pri tome mreža ne mora nalikovati zvijezdi u fizičkom spoju. Pogledajmo skicu ovakvog spoja, na slici **143.4**.

**Slika 143.4 Topologije mreže: zvijezda (*star*) [lijevo: simbolički prikaz, desno: logički način povezivanja]**




Prednosti ovakvog načina povezivanja su:

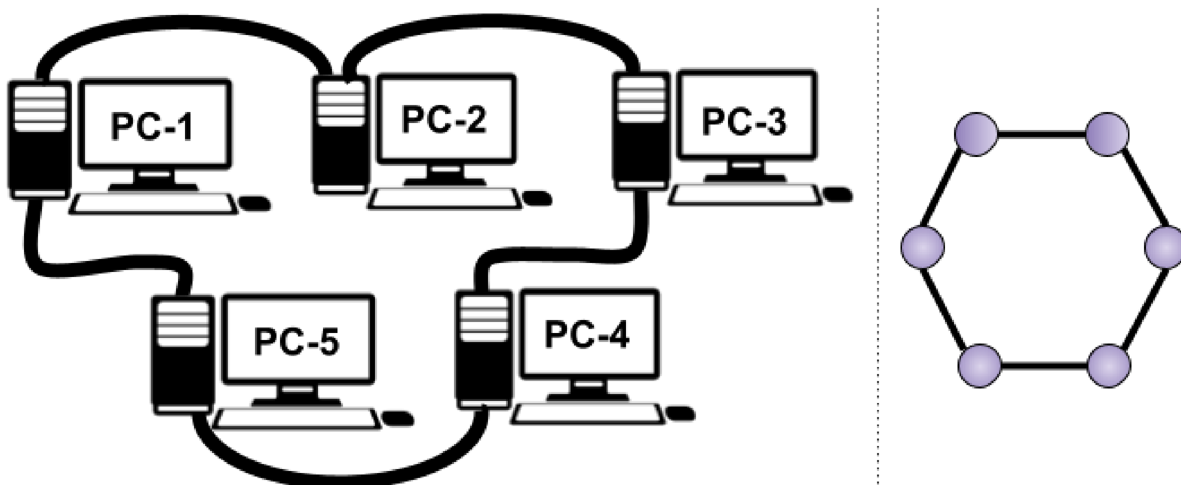
- Dizajn i izvedba su jednostavni.
- Koristi relativno manje kablova, stoga je manje radno intenzivan.
- Cijelom mrežom se lako može upravljati s jedne lokacije.
- Budući da su čvorovi neovisno povezani s čvorištem, problem s jednim čvorom neće utjecati na cijelu mrežu.
- Novi čvorovi se jednostavno mogu dodati ili ukloniti bez isključivanja cijele mreže.
- Rješavanje problema i održavanje mreže su lakši.

Mane ovog načina povezivanja su:

- Budući da sav promet mora proći kroz središnje čvorište, to je jedina točka kvara, što nije idealno.
- Izvedba mreže i ukupna propusnost ograničeni su tehničkim specifikacijama središnjeg čvorišta (obično preklopnika).

**5. Prstenasta** topologija (Engl. *Ring*) je slična linearnoj topologiji, ali sa zatvorenom petljom tako da su čvorovi (pr. računalo ili mrežni uređaj) raspoređeni u prsten ili krug. Svaki čvor ima točno dva susjedna čvora s kojima je spojen, a podaci putuju u jednom smjeru prolazeći kroz svaki među čvor u prstenu dok ne stignu do odredišnog čvora.
U određenim konfiguracijama, moguće je projektirati takav sustav da podaci mogu prolaziti u oba smjera; dodavanjem druge veze između mrežnih čvorova, stvarajući topologiju ***dvostrukog prstena***. Pogledajmo skicu ovakvog spoja, na slici **143.5**.

**Slika 143.5 Topologije mreže: prsten (ring) [lijevo: simbolički prikaz, desno: logički način povezivanja]**




Prednosti prstenaste topologije odnosno spoja:

- Budući da samo jedan čvor može prenositi podatke u isto vrijeme, što smanjuje kolizije paketa pa su topologije prstena učinkovitije u prijenosu podataka.
- Topologije prstena su isplative, a instalacija je relativno jeftina.
- Identificiranje i rješavanje problema je lakše, zbog same topologije.

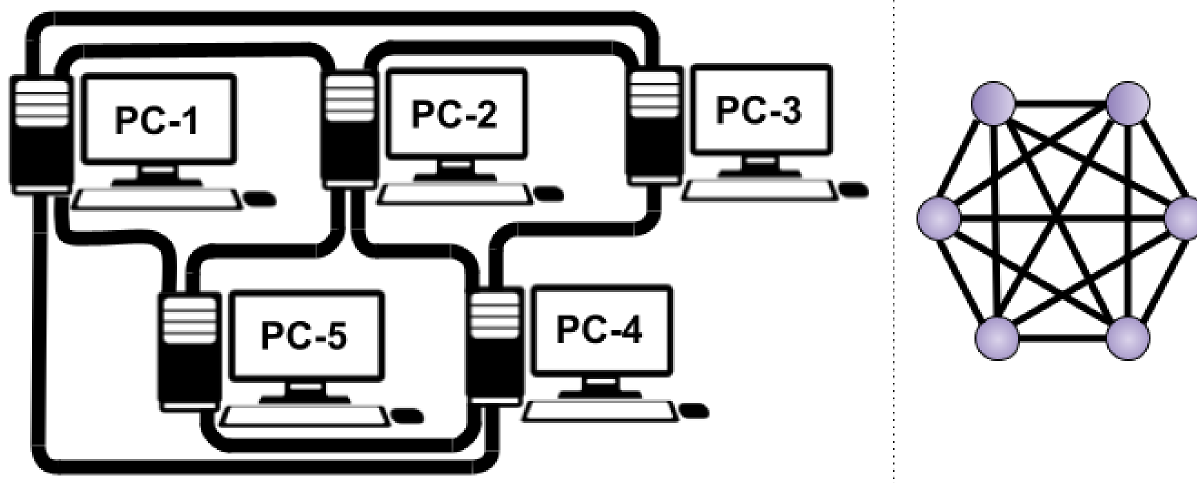
Mane prstenaste topologije odnosno povezivanja:

- Ako jedan čvor zakaže, cijela mreža će se pokvariti.
- Velike prstenaste mreže pate od sporijeg prijenosa podataka jer propusnost mreže dijele svi uređaji.
- Lako je preopteretiti mrežne resurse i kapacitet.
- Dodavanje ili uklanjanje čvorova zahtijeva da cijela mreža bude isključena tijekom dodavanja.

**6. Mrežasta** topologija (Engl. *Mesh*) odnosno topologija isprepletene mreže je ona u kojoj se čvorovi izravno i dinamički povezuju s mnogim drugim čvorovima. Sastoji se od razrađene strukture međupovezivanja od točke do točke među čvorovima. Moguće je imati i djelomičnu topologiju mreže, gdje neki čvorovi imaju dvije ili više veza, ili punu topologiju mreže, gdje su svi čvorovi u potpunosti povezani sa svakim drugim čvorom u mreži. Mrežasta topologija ima nehijerarhijsku strukturu i čvorovi surađuju u učinkovitom usmjeravanju podataka.

Zbog nedostatka ovisnosti o jednom čvoru ili ruti, svaki čvor može sudjelovati u prijenosu informacija. Pogledajmo skicu ovakvog spoja, na slici **143.6**.

**Slika 143.6 Topologije mreže: mrežasti spoj (mesh) [lijevo: simbolički prikaz djelomičnog mrežastog spoja, desno: logički način povezivanja u potpunom mrežastom spoju]**




Prednosti ovakvog spoja su sljedeće:

- Ovo je najstabilnija i najpouzdanija topologija mreže.
- Cijela mreža je otporna na kvar bilo kojeg čvora, zbog velikog stupnja međusobne povezanosti.
- Ne postoji jedinstvena točka ispada. Čak i ako se jedan ili dva čvora pokvare, mreža će i dalje raditi.

Mane ovog načina povezivanja:

- Ovakve mreže radno su intenzivne jer zahtijevaju puno kablova i mrežnih sučelja.
- Kabliranje, rad i vrijeme konfiguracije čine ju skupom.

**7. Hijerarhijska** ili tradicionalna troslojna topologija odnosno arhitektura sastoji se od tri sloja u implementaciji:

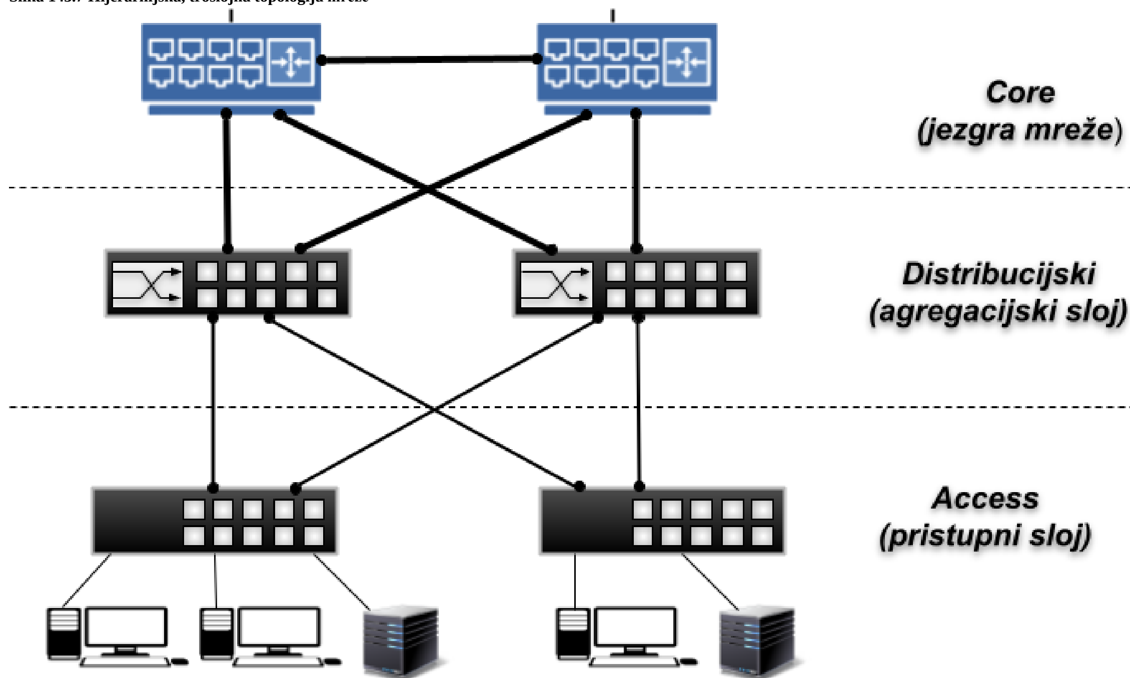
- Sloja jezgre (Engl. ***Core***),
- Distribucijskog odnosno *agregacijskog* sloja (engl. ***Distribution/Aggregation***)
- Pristupnog sloja (Engl. ***Access***).

Uređaji u svakom sloju međusobno su po mogućnosti povezani redundantnim vezama (zalihost) koja može stvoriti petlje u mreži. U prošlosti je većina prometa, a najviše u podatkovnim centrima, bila od poslužitelja do poslužitelja, ili od poslužitelja do sustava za pohranu podataka, što smatramo prometom "istok prema zapadu" (pogledajte sliku **143.7.**).

Model troslojne arhitekture obično je dizajniran za promet od istoka prema zapadu, tako da se paketi kreću kroz tri razine: do jezgre, usmjeravaju se do preklopnika sloja *agregacije*, a zatim se prosljeđuju pristupnom (*Access*) preklopniku na koji su krajnji uređaji spojeni.

Pogledajmo skicu odnosno logičku shemu ovakvog spoja, na slici 143.7.

Slika 143.7 Hijerarhijska, troslojna topologija mreže



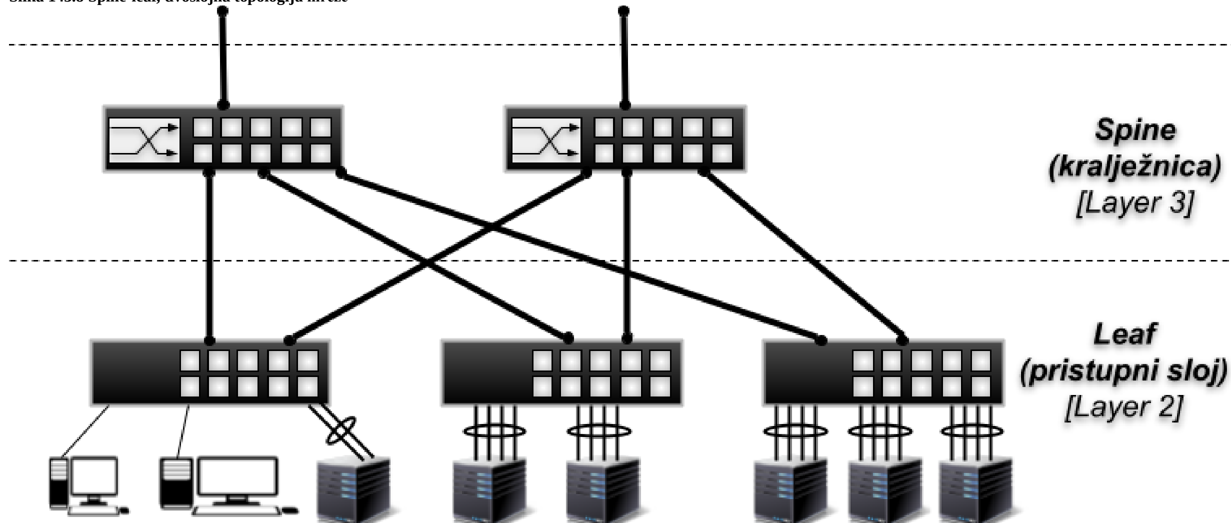

Međutim, s transformacijom podatkovnih centara, zahtijeva se više prolazaka podataka unutar podatkovnog centra dok se broj prolaznih uređaja (primjerice klasičnih (*Layer 2*), te *Layer 3* preklopnika i usmjerivača) povećava, što dodaje veću mogućnost i vjerojatnost za gubitak paketa te uvodi značajno kašnjenje. Dakle, ako se odvija intenzivan promet istok-zapad kroz ovu konvencionalnu arhitekturu, uređaji spojeni na isto komutacijsko sučelje (vezu prema uređaju višem u hijerarhiji), mogu se boriti za propusnost, što rezultira lošim vremenom odziva koje dobivaju krajnji korisnici.
Stoga ova troslojna arhitektura nije prikladna za moderne virtualizirane podatkovne centre u kojima se poslužitelji za virtualizaciju, kao i virtualna računala, ali i poslužitelji za pohranu, mogu nalaziti bilo gdje unutar topologije. Pogotovo s potrebama ekstremno brze mrežne komunikacije među njima, a naročito ako je kod njih potrebna redundancija (zalihost).
Kod dizajniranja ovakve mreže, vrlo važno je obratiti pažnju na propusnost komutacijskih veza, poput primjerice:

- ***Core*** → ***Core*** te ***Core*** → ***Distribution*** međuveza.
- ***Distribution*** → ***Access*** međuveza.
- ***Access*** veza prema krajnjim točkama to jest uređajima: poslužiteljima, računalima i slično.

Za više detalja o dizajnu i detaljima vezanim za dizajn mreže, pogledajte sljedeću topologiju (**Spine-Leaf**).

8. **Spine-Leaf** topologija je poboljšanje hijerarhijske troslojne topologije. Naime troslojni hijerarhijski dizajn u modernim podatkovnim centrima, ali i brzim, visoko opterećenim redundantnim **LAN** mrežama, dolazi do svojih sve većih ograničenja.
Kao što je vidljivo, ovakav dizajn sastoji se samo od dva sloja: sloja lista (***Leaf***) i sloja kralježnice (***Spine***), što smanjuje broj prolaznih (transportnih) mrežnih uređaja te jamči smanjeno kašnjenje. Dakle ovo je arhitektura gdje postoje samo dvije razine preklopnika između poslužitelja i jezgrene (***Core***) mreže.
Pogledajmo skicu ovakvog spoja, na slici 143.8.

Slika 143.8 Spine-leaf, dvoslojna topologija mreže



Ostale uobičajene razlike ove topologije uključuju sljedeće značajke:

- Povećana je upotreba preklopnika s fiksnim brojem sučelja (portova) u odnosu na modularne modele za mrežnu okosnicu.
- Više kabela za kupnju i upravljanje, s obzirom na veći broj međuveza.
- Znatno bolje skaliranje u slučaju povećanja infrastrukture.
- *Spanning Tree Protocol* (**STP**) se više ne koristi.

**Spine** sloj (sloj kralježnice) se sastoji od preklopnika (*switcheva*) koji odrađuju usmjeravanje (*Layer* 3 sloj), radeći kao okosnica mreže. **Leaf** sloj (sloj lista) uključuje pristupni preklopnik koji se povezuje s krajnjim točkama poput poslužitelja, uređaja za pohranu podataka, računala i slično, a u pravilu rade kao klasični preklopnici (*Layer 2*). U ovoj arhitekturi, svaki preklopnik je međusobno povezan sa svakim preklopnikom kralježnice (okosnice). S ovim dizajnom, svaki poslužitelj može komunicirati s bilo kojim drugim poslužiteljem s ne više od jedne staze između bilo koja dva pristupna (**leaf**) preklopnika.

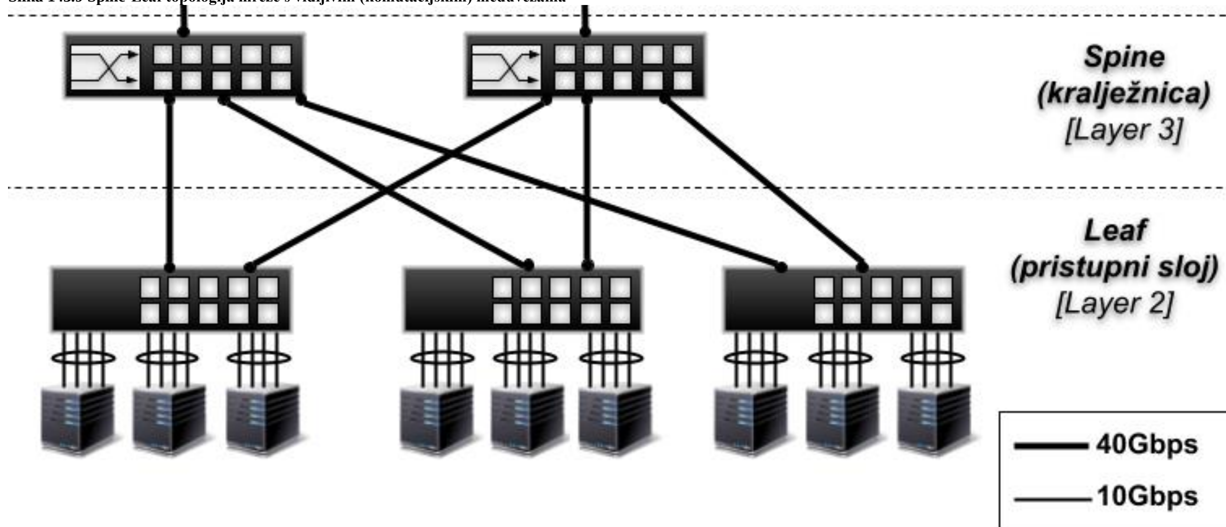
Pri dizajnu ovakvih mreža, potrebno je voditi računa o nekoliko faktora:

***Oversubscription Ratios*** odnosno omjeru prevelike/prekomjerne pretplate.
Prekomjerna pretplata je omjer zagušenja kada svi uređaji šalju promet u isto vrijeme. Može se mjeriti u smjeru sjever/jug (promet koji ulazi/izlazi iz podatkovnog centra tj. preko jezgrenih uređaja) kao i istok/zapad (promet između uređaja u podatkovnom centru odnosno lokalnoj mreži). Trenutni moderni dizajni mreža imaju omjere prekomjerne pretplate od **3:1** ili manje, što se mjeri kao omjer između uzvodne širine pojasa (prema preklopnicima) i kapaciteta prema dolje (prema poslužiteljima ili sustavima za pohranu podataka, računalima i slično).

Zamislimo topologiju kao na slici **143.9**, gdje je vidljivo da su međuveze između ***Spine*** preklopnika i ***Leaf*** preklopnika, veće propusnosti, konkretno 40Gbps, a pošto ih imamo šest, imamo: 6x40Gbps=**240Gbps** ukupne propusnosti na ovom sloju mreže.
Međutim, ako je svaki ***Leaf*** preklopnik spojen s dvije 40Gbps međuveze prema ***Spine*** preklopnicima, tada je propusnost pojedinog ***Leaf*** preklopnika prema jezgri mreže 2x40Gbps=**80Gbps**. Pri tome, ako svaki ***Leaf*** preklopnik ima 24x10Gbps mrežnih sučelja za spajanje poslužitelja, računala i drugih uređaja, tada se na njemu može generirati maksimalno: 24x10Gbps=**240Gbps** prometa. To nam sve zajedno govori, da prema pojedinom **Spine** preklopniku imamo **80Gbps** propusnosti, a prema dolje, to jest prema poslužiteljima i drugoj opremi, do maksimalno **240Gbps**, što je upravo i maksimalni dozvoljeni omjer **3:1** to jest: 240:80(Gbps).

To znači da smo se osigurali, da čak i ako neki od poslužitelja na pristupnom sloju (**Leaf**) generira maksimalan promet, odnosno čak i ako osam (8) poslužitelja koji imaju 10Gbps veze, kreiraju istovremeno maksimalan promet prema jezgri mreže ili preko jezgre mreže prema nekom drugom pristupnom (***Leaf)*** preklopniku, da neće moći doći do zagušenja.

**Slika 143.9 Spine-Leaf topologija mreže s vidljivim (komutacijskim) međuvezama**






Kod **dizajna** mreže, treba uvijek paziti da su komutacijske međuveze odnosno veze od jezgre mreže ([***Core***] ili [***Spine***]) prema pristupnom sloju mreže ([***Access***] ili [***Leaf***]), uvijek znatno veće propusnosti od propusnosti veza od pristupnog sloja mreže prema uređajima spojenim na njih (poslužitelji, računala i slično).
To je zbog toga kako ne bi došlo do zagušenja u komunikaciji. Slično kao što je situacija kod dizajna preklopnika: ***Blocking*** ili ***Non Blocking*** *dizajn* (***Poglavlje:*** **20.3. Ne blokirajući i blokirajući dizajn preklopnika**).

**9. Hibridna** topologija je topologija u kojoj se kombinira više vrsta topologija; primjerice:

- Sabirnica i hijerarhijska topologija.
- Sabirnica i proširena zvjezdasta topologija.
- Sabirnica i hijerarhijska i/ili linearna ili prstenasta topologija ili neke druge kombinacije topologija.

**Izvori informacija: (1045),(1046),(1047),(1048),(K-10).**

# 19.2. Mrežni pojmovi

Računalne mreže prema zemljopisnom području koje pokrivaju dijelimo na: **LAN** (*Local Area Network*), **WAN** (*Wide Area Network*) mreže, ali i **MAN** (*Metropolitan Area Network*) mreže.

- ***LAN*** - odnosno lokalna mreža je primjerice mreža unutar jedne zgrade, kao što je mreža unutar neke tvrtke ili kuće.
- ***WAN*** - mreža je mreža koja povezuje više ***LAN*** mreža između udaljenih lokacija poput dvije lokacije tvrtke ili vaše tvrtke s internetom. Ove mreže mogu pokrivati iznimno velike udaljenosti (i tisuće kilometara).

Naravno postoje i neke druge specifične vrste mreža poput: *SAN*, *VPN*, … ali nam nisu trenutno važne za uvod u mreže.
Važno je znati kako ***LAN*** mreže u praksi rade na puno većim brzinama od ***WAN*** mreža. Iznimke su velike korporacije, banke i telekomi koji si mogu priuštiti i ***WAN*** mreže brzina koje su inače u upotrebi u ***LAN*** mrežama.
Razlike u brzinama između prosječnog *LAN*-a i *WAN*-a koji se danas koriste, mogu biti:

- ***LAN***: 100Mbps, 1.000Mbps (1Gbps), 10.000Mbps (10Gbps), 20.000Mbps (20Gbps), 40.000Mbps (40 Gbps), 100.000 Mbps (100Gbps) te 200Gbps, 400Gbps i 800Gbps. Radi se i na razvoju 1.6 Tbit/s mreža [*očekuje se IEEE standard do 2025.g.*].
- ***WAN***: 10Mbps, 20Mbps, 30Mbps, 50Mbps, 100Mbps, 200Mbps, 1Gbps, 10Gbps, 25Gbps ili više.

Dakle prosječne brzine u *LAN* mrežama su 100 i više puta veće nego u *WAN* mrežama. Naravno u iznimnim slučajevima to se može i izjednačiti, ali uz ekstremno visoke troškove. Razlika, osim u brzini odnosno propusnosti je i u kašnjenju odnosno odzivu lokalnih (*LAN*) i udaljenih mreža (*WAN*), zbog zakona fizike (udaljenost odnosno putovanje signala do odredišta).

**Izvori informacija: (1011),(1012),(K-3),(K-6),(K-10).**

## 19.2.1. Kablovi i konektori

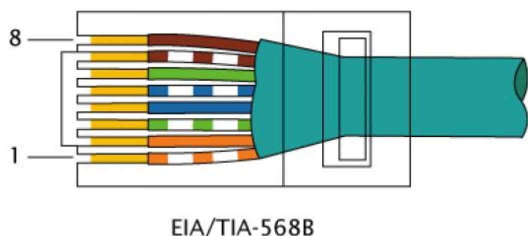
U ovom poglavlju upoznat ćemo se s dvije osnovne kategorije kabela za umrežavanje:

- Bakrenih kabela.
- Optičkih kabela (takozvanih svjetlovoda).

### 19.2.1.1. Bakar

Lokalnu (***LAN***) mrežnu opremu, poglavito preklopnike, ali i usmjerivače, povezujemo uglavnom s takozvanim ***UTP*** (Engl. *Unshielded twisted pair*) kablovima odnosno kablovima s prepletenim paricama. Ovakva vrsta kabela sastoji se od četiri uvijene parice, što čini ukupno osam (8) vodova odnosno vodiča (žila). Slika 144 prikazuje **RJ-45** utikač sa spojenim (*zakrimpanim*) *UTP* kabelom, i to spojenom prema ***ANSI/TIA/EIA 568B*** standardu:

**Slika 144. Pogled na RJ-45 utikač i žile UTP kabela**



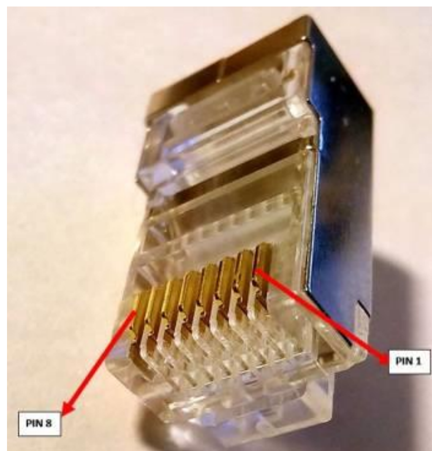

Svaki vodič (žila kabela) je obojan zasebnom bojom, pa tako unutar svake parice imamo sljedeće boje vodova:

- Narančasti i bijelo-narančasti (drugi par).
- Zeleni i bijelo-zeleni (treći par).
- Plavi i bijelo-plavi (prvi par).
- Smeđi i bijelo smeđi (četvrti par).

Kablovi se spajaju na posebne utikače s osam (8) nožica, znane kao **8P8C** (*8 position 8 contact*) naziva **RJ-45** (Engl. *Registered Jack*) ili na utičnice odnosno na prespojne panele (*patch panele)* koji na kraju opet imaju **RJ-45** utičnicu.

Slika 145 prikazuje ***RJ-45***, oklopljeni utikač (za *STP* ili *FTP* kabele) s vidljivim i označenim nožicama (*pinovima*).

**Slika 145. Pogled na RJ-45 utikač**




RJ-45 utikač na slici nije spojen odnosno utisnut na kabel (*zakrimpan*) te su njegove nožice stoga izvučene.
Tablica nam prikazuje shemu spajanja prema **ANSI/TIA/EIA 568 A** ili **B**:

| Vodič | Nožica prema 568 B | Nožica prema 568 A |
|---|---|---|
| Bijelo-narančasta | 1 | 3 |
| Narančasta | 2 | 6 |
| Bijelo-zelena | 3 | 1 |
| Zelena | 6 | 2 |
| Plava | 4 | 4 |
| Bijelo-plava | 5 | 5 |
| Bijelo-smeđa | 7 | 7 |
| Smeđa | 8 | 8 |

Što se boja tiče, razlika u spajanju je sâmo u tome što su zamijenjene pozicije drugog (2) i trećeg (3) para/parice tj. narančaste i zelene boje.
Vezano za spajanje i upotrebu, ako gledamo obje strane kabela odnosno utikača, tada imamo sljedeće stanje primjene.

| Jedna strana kabela | Druga strana kabela | Opis namjene kabela |
|---|---|---|
| **568 A** | **568 A** | Normalni kabel znan i kao prespojni (*patch)* ili *straight through* kabel. |
| **568 B** | **568 B** | Normalni kabel znan i kao prespojni (*patch)* ili *straight through* kabel. |
| **568 A** | **568 B** | Ukrižani odnosno takozvani *ethernet crossover* kabel. |

Naime, ako su obje strane kabela isto spojene na **RJ-45** utikaču (**A→A** ili **B→B**) dobivamo klasični (obični) kabel.
Ali ako izradimo kabel na kojemu su obje strane različite (**A→B** ili **B→A**), tada dobivamo ukrižani odnosno *crossover* kabel.
Ukrižani kabel obično se koristi za povezivanje istih uređaja ili mrežnih uređaja, poput primjerice:

- *Preklopnik (switch)* ←→ *Preklopnik* (*switch)*
- *Usmjerivač (router)* ←→ *Usmjerivač* (*router)*
- *Usmjerivač (router)* ←→ *Preklopnik* (*switch)*
- *Računalo 1* → *Računalo 2*

Kod ukrižanog odnosno *crossover* kabela su direktno spojeni: **TX(slanje)** na **RX(primanje)** nožice odnosno signale.

Konkretno to znači sljedeći spoj:
- *Pin (nožica)* 1 (**TX +**) → *Pin (nožica)* 3 (**RX +**)
- *Pin (nožica)* 2 (**TX -**) → *Pin (nožica)* 6 (**RX -**)

To znači da ono što se šalje (**TX**) s jedne strane, dolazi direktno na prijem (**RX**) s druge strane.
Slična vrsta kabela postoji i kod *serijskih* kabela, koji se nazivaju i ***null modem*** kabeli.

## Vratimo se kabelima

Svaka uvijena parica se u komunikaciji koristi za svoju namjenu:
- Za brzine **do 100 Mbps**: jedna parica je za slanje (*Transmit* - **TX**), a druga za primanje (*Receive* - **RX**) podataka (signala). Za brzine do 100 Mbps se koriste samo dvije od četiri parice (iako sve moraju biti uredno spojene prema standardima **ANSI/TIA/EIA 568A** ili **ANSI/TIA/EIA 568 B**). Tako da imamo veze sa sljedećim paricama:
  - *Pin (nožica)* 1 [**TX +**] i *Pin (nožica)* 2 [**TX -**].
  - *Pin (nožica)* 3 [**RX +**] i *Pin (nožica)* 6 [**RX -**].

- Za brzine **od 1.000 Mbps (1 Gbps) i više**, svaka parica se koristi za dvosmjernu (*bidirekcionalnu [tzv.* ***full-duplex****]*) komunikaciju. Ovdje se koriste sve četiri parice pa tako imamo veze sa sljedećim paricama:
  - *Pin (nožica)* 1 [*bidirekcionalni* par **A +**] i *Pin (nožica)* 2 [*bidirekcionalni* par **A -**].
  - *Pin (nožica)* 3 [*bidirekcionalni* par **B +**] i *Pin (nožica)* 6 [*bidirekcionalni* par **B -**].
  - *Pin (nožica)* 4 [*bidirekcionalni* par **C +**] i *Pin (nožica)* 5 [*bidirekcionalni* par **C -**].
  - *Pin (nožica)* 7 [*bidirekcionalni* par **D +**] i *Pin (nožica)* 8 [*bidirekcionalni* par **D -**].

Mrežne kabele (i utičnice) dijelimo prema:
- **Kategoriji**: kategorije danas u upotrebi su kategorije kablova: **5**, **6** i **7**, uz razne potkategorije:
  - **Kategorija 5 (Cat 5)**, koja je danas minimalna kategorija u upotrebi i to potkategorija 5 (E), znana kao **CAT 5E** (omogućava brzine do 1 Gbps).
  - **Kategorija 6 (Cat 6)** je u upotrebi za brzine 1Gbps (duljine do 100.m.) i za 10Gbps (duljine do 50.m.).
    - **Cat 6A** – dvostruko boljih karakteristika od **Cat 6** - za brzine 1Gbps (duljine do 100.m.) i za 10Gbps (duljine do 100.m.).
  - **Kategorija 7 (Cat 7)** - za brzine do 10 Gbps (duljine do 100.m.) i više. Postoji i potkategorija:
    - **Cat 7A** – za brzine od 10 Gbps (duljine do 100.m.), 40 Gbps (duljine do 50.m.) i 100 Gbps (duljine do 15.m.).
  - **Kategorija 8 (Cat 8)** – standardno za brzine do 40Gbps, uz dvije potkategorije:
    - **Cat 8.1.** – za brzinu 25Gbps: za duljine kabela od 30.m.
    - **Cat 8.2.** – za brzinu 40Gbps: za duljine kabela od 30.m.

- **Vrsti kabela**:
  - ***UTP*** – neoklopljen, s uvijene četiri (4) parice. Ova vrsta kabela je podložna na razne elektromagnetske smetnje koje mogu uzrokovati probleme u radu mreže.
  - ***FTP*** – oklopljene s folijom (**F** - *Foil*), uvijene četiri (4) parice. Zaštićeno od smetnji zbog zaštitne folije. Pri tome oklop (folija) mora biti uzemljena preko utikača odnosno preko utičnice.
    ***STP*** – oklopljene s opletom žica (**S** - *Shield*), uvijene četiri (4) parice. Zaštićeno od smetnji zbog zaštitnog opleta žica. Pri tome oklop (oplet žica) mora biti uzemljena preko utikača odnosno preko utičnice.
  - ***SFTP*** – oklopljene s opletom žica i s folijom (**S** i **F**), uvijene četiri (4) parice. Zaštićeno od smetnji zbog zaštitne folije i opleta žica. Pri tome oklop mora biti uzemljen preko utikača odnosno preko utičnice.

Sve navedene **S** ili **F** vrste kabela imaju oplet ili foliju oko svih parica zajedno, na vanjskom dijelu kabela, a ispod vanjskog zaštitnog izolacijskog materijala. Dodatno, ovisno o kategoriji moguće su izvedbe u kojima su i pojedine parice zasebno obavijene folijom. Kod svih varijanti **S** ili **F** vrste (koje imaju ili oplet ili foliju) potrebno je napraviti uzemljenje istih, preko utikača (konektora) i utičnice te prespojnog (*patch)* panela.

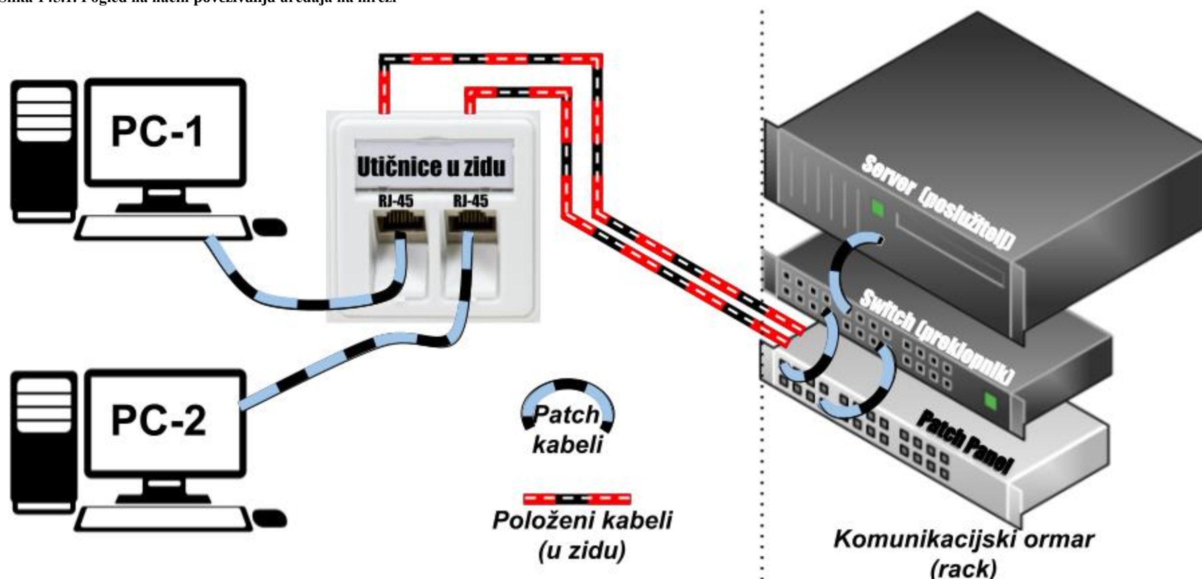


Važno je paziti na to da utikač (**RJ-45**), utičnice i prespojni (*patch) paneli* moraju biti identične i kategorije i vrste, kao i kabel, inače može doći do problema u radu mreže.

Maksimalne duljine kabela se odnose za ukupnu duljinu kabela: od računala do utičnice (kroz prespojni kabel), preko *prespojnog (patch) panela* do prvog aktivnog mrežnog uređaja. Sve duljine se prema tome izražavaju prema principu:

*Uređaj 1 (pr. **PC-1**) → prespojni kabel →* **patch panel** *→ prespojni kabel →* ***aktivni mrežni uređaj*** *(pr.* ***Switch [preklopnik]****)*

**Slika 145.1. Pogled na način povezivanja uređaja na mreži**




Na slici 145.1. vidimo kako se krajnji uređaji na mreži (primjerice računala: **PC-1** i **PC-2**), pomoću prespojih (*patc*h) kabela spajaju na **RJ-45** utičnice u zidu. **RJ-45** utičnice su s položenim kabelima iz zida, spojene na prespojni panel (*patch panel*), koji se obično nalazi u poslužiteljskoj sobi u poslužiteljskom ili (tele)komunikacijskom ormaru (*racku*). Tako da je svaka utičnica u zidu spojena na točno određenu utičnicu na *prespojnom (patch) panelu,* sa stražnje strane *prespojnog panela*.

## Strukturno kabliranje

Umrežavanje preko aktivne mrežne opreme (preklopnik na slici) se radi tako da se točno određena utičnica na *prespojnom panelu* (pr. utičnica broj **1**) spaja na određenu utičnicu na preklopniku, isto kao da smo računalo, poslužitelj ili neki drugi mrežni uređaj direktno spojili na njega (na preklopnik). Poslužitelj je ovdje upravo tako (direktno) spojen na preklopnik, samo s prespojnim (*patch)* kabelom.
Kabliranje možemo podijeliti u tri vrste. To je primarno, sekundarno i tercijarno kabliranje.
Primarno kabliranje predstavlja kabliranje unutar kampusa ili kabliranje unutar stambenih zgrada. Sekundarno kabliranje se naziva i vertikalno kabliranje jer se koristi za provlačenje kabela između katova u zgradi. I konačno, horizontalno odnosno podno kabliranje je tercijarno i ono se odnosi na polaganje kablova na svim etažama to jest katovima stambenih zgrada.



Ovakav način polaganja i spajanja kabela, pasivne i aktivne mrežne opreme se naziva strukturno kabliranje.

U telekomunikacijama, strukturno (strukturirano) kabliranje se odnosi na kabliranje unutar stambenih objekata, zgrada ili kampusa. Ovakva kabelska infrastruktura se sastoji od niza standardiziranih manjih elemenata (dakle strukturiranih) koji se nazivaju podsustavi. Komponente strukturnog kabliranja uključuju upletene parice i/ili optičko kabliranje, prespojne panele (*patch paneli*) i prespojne kabele (*patch kabeli*) te druge elemente.

Strukturno kabliranje se odnosi i na dizajn i instalaciju kabelskog sustava koji će podržati višestruku uporabu hardvera i biti prikladan za sve današnje i buduće potrebe. S ispravno instaliranim sustavom mogu se ispuniti trenutni i budući zahtjevi, a bit će podržan i hardver koji se dodaje u budućnosti.

Naime strukturno kabliranje definira performanse topološke mreže u spoju zvijezda, definira dopuštene udaljenosti povezivanja, određuje vrstu i metode povezivanja, specificira zahtjeve za parametre kvalitete pojedinih elemenata i cijelog mrežnog sustava, neovisno o odabranom mediju kabelskog sustava.
Temeljna načela koja strukturni kabeli trebaju zadovoljiti su da kabeli trebaju biti:

- **Zasićeni** – odnosno da mogu osigurati dovoljan broj priključnih točaka u odnosu na broj zaposlenih.
- **Generički** - odnosno da je postavljanje kabela neovisno o tehnologiji opreme.
- **Fleksibilni** – odnosno da je omogućeno povezivanje sustava putem standardiziranih komponenti.

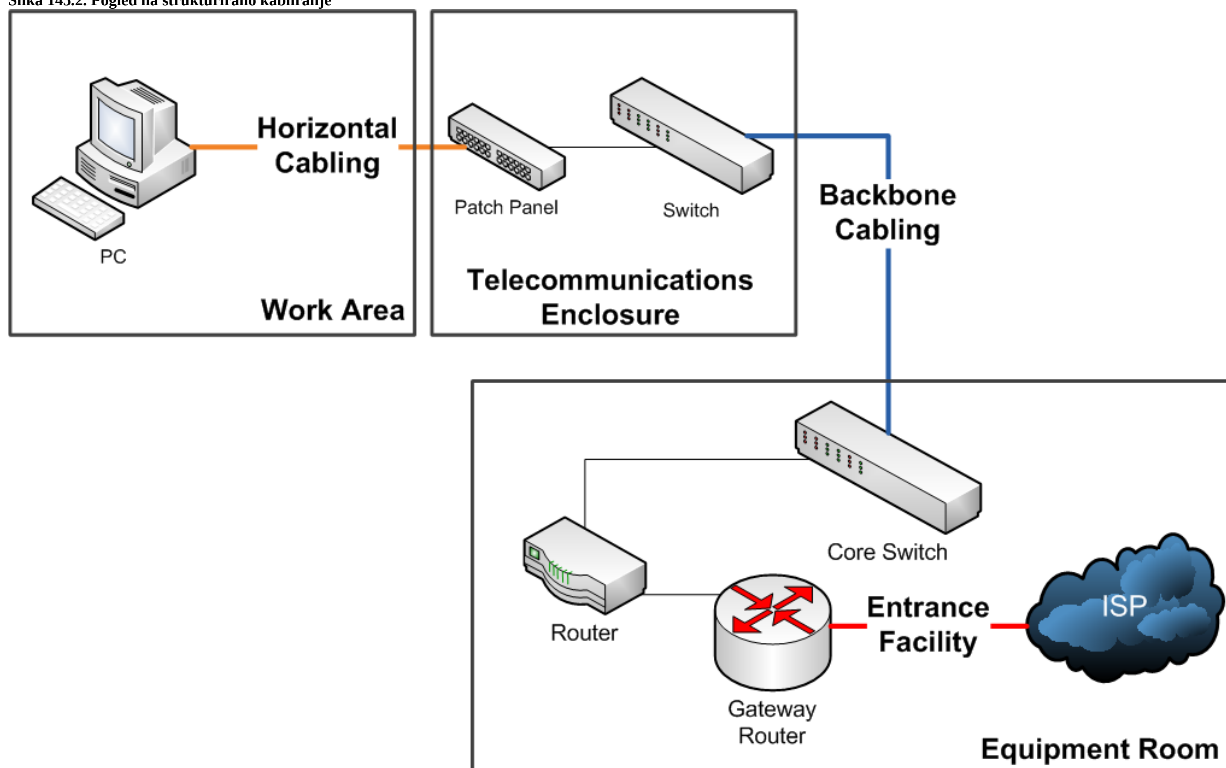
Projektiranje i instalacija strukturnog kabliranja regulirano je skupom standarda koji određuju ožičenja podatkovnih centara, ureda i stambenih zgrada, za podatkovnu ili govornu komunikaciju pomoću različitih vrsta kabela, a najčešće su to:

- Bakreni kabeli kategorije **5e** (*Cat 5e*), za brzine do 2.5Gbps (**2.5GBASE-T** standard).
- Bakreni kabeli kategorije **6** (*Cat 6*), za brzine do 10Gbps (**10GBASE-T** standard), za udaljenost do 55 metara.
- Bakreni kabeli kategorije **6a** (*Cat 6a*), za brzine do 10Gbps (**10GBASE-T** standard), za udaljenost do 100 metara.
- Bakreni kabeli kategorije **7** (*Cat 7*), za brzine do 10Gbps (**10GBASE-T** standard), za udaljenost do 100 metara.
- Bakreni kabeli kategorije **7a** (*Cat 7a*), za brzine do 40Gbps (**40GBASE-T** standard), za udaljenost do 50 metara odnosno za brzine do 100Gbps (**100GBASE-T** standard), za udaljenost do 15 metara.
- Optički kabeli i modularne utičnice (konektori) i pripadajući moduli za optičke kabele.

Samo neki od standarda koji definiraju strukturno (strukturirano) kabliranje su:

- Za generičko kabliranje i instalacije: **EN 50173**, **EN 50174**, **ISO/IEC 11801**, **ANSI/TIA/EIA-568-B, 569, 570 i 606.**
- Za uzemljenje i povezivanje je tu standard: **ANSI/TIA/EIA-607**, …

**Slika 145.2. Pogled na strukturirano kabliranje**




**Autor slike: Lanil Marasinghe (Wikipedia).**

Kako je vidljivo na slici 145.2, položeni kablovi koji povezuju **RJ-45** utičnice na zidu i prespojne (*patch*) panele definiraju se kao **horizontalno** odnosno tercijarno kabliranje. Dok se povezivanje centralnog preklopnika prema **WAN** mreži (obično prema usmjerivaču), naziva okosnicom odnosno centralnom vezom (Engl. *backbone*), a ako se odnosi na vezu između etaža (katova) unutar stambenog objekta, tada se može nazivati i **vertikalnim** kabliranjem odnosno sekundarnim kabliranjem.

Dakle u slučaju veze između više telekomunikacijskih soba i/ili povezivanja više njih na različitim lokacijama ili katovima, govorimo o vertikalnom povezivanju odnosno sekundarnom kabliranju.

Kada govorimo o standardima, sljedeći standardi se odnose na *Ethernet* komunikaciju preko bakrenih uvijenih parica:

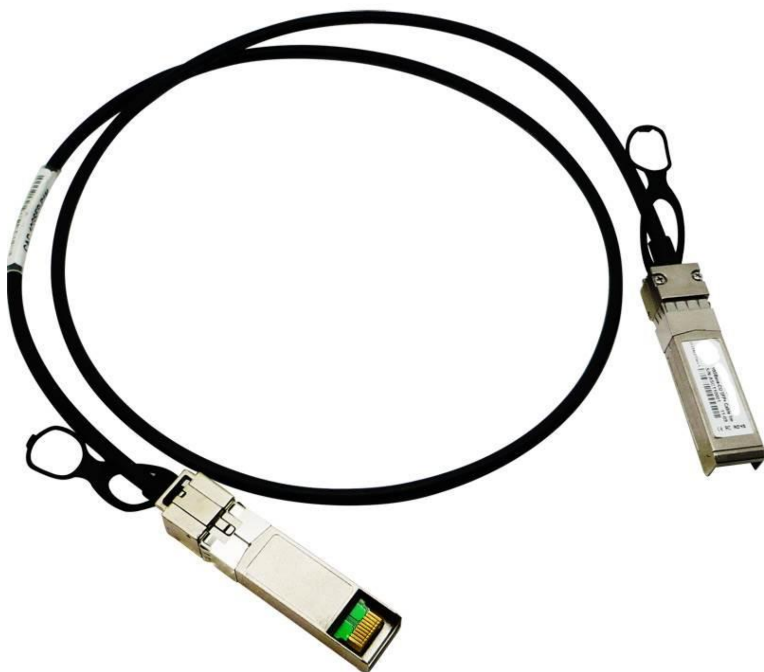
- Za rad na 10 Mbps [**10BASE-T**] je zadužen standard **802.3i.**
- Za rad na 100 Mbps [**100BASE-TX**] je zadužen standard **802.3u.**
- Za rad na 1.000 Mbps (1 Gbps) [**1000BASE-T**] je zadužen standard **802.3ab.**
- Za rad na 2.500 Mbps (2.5 Gbps) [**2.5GBASE-T**] je zadužen standard **802.3bz-2016**.
- Za rad na 5.000 Mbps (5 Gbps) [**5GBASE-T**] je zadužen standard **802.3bz-2016**.
- Za rad na 10.000 Mbps (10 Gbps) [**10GBASE-T**] je zadužen standard **802.3an.**
- Za rad na 25.000 Mbps (25 Gbps) [**25GBASE-T**] je zadužen standard **802.3bq-2016.**
- Za rad na 40.000 Mbps (40 Gbps) [**40GBASE-T**] je zadužen standard **802.3bq-2016.**
- Za rad na 100.000 Mbps (100 Gbps) [**100GBASE-T**] su zaduženi sljedeći standardi:
  - **802.3ba-2010, 802.3bg-2011**
  - **802.3bj-2014, 802.3bm-2015**
  - **802.3cd-2018**

Navedeni standardi se odnose na komunikaciju preko uvijenih parica.
Međutim postoji i cijeli nîz standarda za razne varijante optičkih komunikacija.
Za veće brzine (2021.g.) postoje i 200Gbps i 400Gbps *ethernet*, koji je opisan u **802.3bs**, ali trenutno podržava samo optičku komunikaciju, kako je definirano u: **200GBASE-LR4**, **200GBASE-FR4**, **200GBASE-DR4** i **400GBASE-SR16**.

## Direct Attach Copper (DAC)

Osim bakrenih parica (UTP/STP/FP/SFTP) i pripadajućih utikača RJ-45, kada je potrebno povezati mrežne uređaje (najčešće mrežne preklopnike, usmjerivače i sličnu mrežnu opremu) vrlo velikim brzinama, a osim optike, o kojoj ćemo uskoro govoriti, postoji i još jedan način povezivanja. To je takozvani *Direct Attach Cable* odnosno **DAC**, koji se često naziva i izravni kabel, izravni bakreni kabel ili kabel velike brzine. Ovakva veza velike brzine je vrsta jeftinog rješenja za povezivanje na kratke udaljenosti koje zamjenjuje optičke module. Oba njegova kraja imaju sklopove unutar posebnih SFP/SFP+ ili QSFP modula koji su spojeni samo s fiksnim priključkom. To znači da se **SFP** utikač odnosno glava modula i posebni bakreni kabel ne mogu odvojiti. Bakreni kabeli koji se ovdje koriste su posebni takozvani ***Twinaxial*** kabeli, koji su slični ***koaksijalnim*** kabelima.
U usporedbi s aktivnim optičkim kabelima, **DAC** moduli na kabelima velike brzine ne zahtijevaju skupe optičke lasere i druge elektroničke komponente. Stoga se štedi na cijeni i troškovima potrošnje energije.

**Slika 145.3. SPF+ modul s pripadajućim Twinax kabelom (tzv. DAC kabel).**

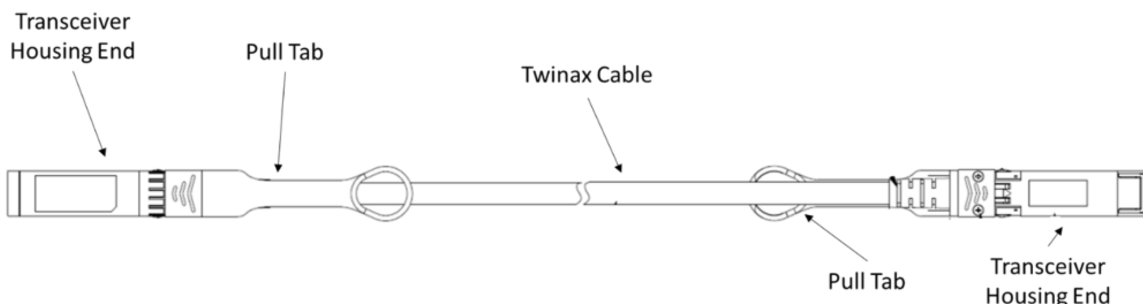


Štoviše, ovakav **SFP + *Twinax*** (**DAC**) kabel ima nekoliko puta nižu latenciju od trenutnih **10GBASE-T** Cat 6/Cat 6A/Cat 7 **SFP+** sustava kabliranja.
To je stoga što **10GBASE-T** ima drugačije kodiranje (i dekodiranje) signala, za koje je potrebno nešto više vremena za obradu.

S druge strane **SFP + Twinax** (**DAC**) ima nešto veću latenciju od optičke varijante to jest od **SFP + Optike**.
Potrošnja energije ***Twinaxa*** sa **SFP+** je oko 0,1 vata, što je također puno bolje od 4–8 vata za **10GBASE-T**.
**SFP+ Twinax** (**DAC)** na slici se također naziva **10GBASE-CR** ili **10GBASE-CR1**.
Nadalje, postoje i pasivne i aktivne varijante ovakvih kabela. Pasivne varijante su u izvedbama za duljine do maksimalno 7.m., dok one aktivne mogu biti za duljine do čak 15.m.

**Izvor slike:** https://en.wikipedia.org/wiki/File:Twinax-CAB-10GSFP-P8M.jpg

**Slika 145.4. SPF DAC s Twinax kabelom**




**Izvor slike:** https://www.addonnetworks.com/faq/dac

Pogledajmo i neke od uobičajenih izvedbi **DAC** kabela i pripadajućih modula (lijeva-desna strana):

| Lijeva strana modula | Desna strana modula | Propusnost |
|---|---|---|
| **SFP+** | **SFP+** | **10 Gbps** |
| **XFP** | **SFP+** | **10 Gbps** |
| **SFP28** | **SFP28** | **25 Gbps** |
| **QSFP+** | **QSFP+** | **40 Gbps** |
| **QSFP+** | **4 x SFP+** | **40 Gbps → 4 x 10 Gbps** |
| **QSFP+** | **4 x XFP** | **40 Gbps → 4 x 10 Gbps** |
| **QSFP28** | **QSFP28** | **100 Gbps** |
| **QSFP28** | **4 x SFP28** | **100 Gbps → 4 x 25 Gbps** |
| **QSFP+** | **QSFP+** | **56 Gbps** |
| **QSFP56** | **QSFP56** | **200 Gbps** |
| **QSFP112 ili QSFP-DD** | **QSFP112 ili QSFP-DD** | **400 Gbps** |

O **SFF** (*Small Form Factor*) odnosno **SFP/SFP+/XFP/QSFP** sučeljima detaljnije u jednom od poglavlja koja slijede.

**Izvori informacija: (2),(3),(4),(44),(983),(984),(1013),(1555),(1556),(1557),(1558),(1559),(K-3),(K-6),(K-9).**

## 19.2.1.2. Optika

Za umrežavanje pomoću optičkih vlakana (tzv. svjetlovoda), potrebne su i posebne mrežne kartice, s optičkim primopredajnicima na kojima se nalazi neka od utičnica (konektora) za spajanje optičkih vlakana, poput:

- **LC** utičnice (najčešće se koriste za računala i mrežnu opremu [preklopnici, usmjerivači, …]).
- **SC** utičnice (najčešće se koriste za **FTTH** [*Fiber To The Home*] i u LAN mrežama).
- **ST** (najčešće se koriste u industrijskim mrežama) ili nekih drugih utičnica.

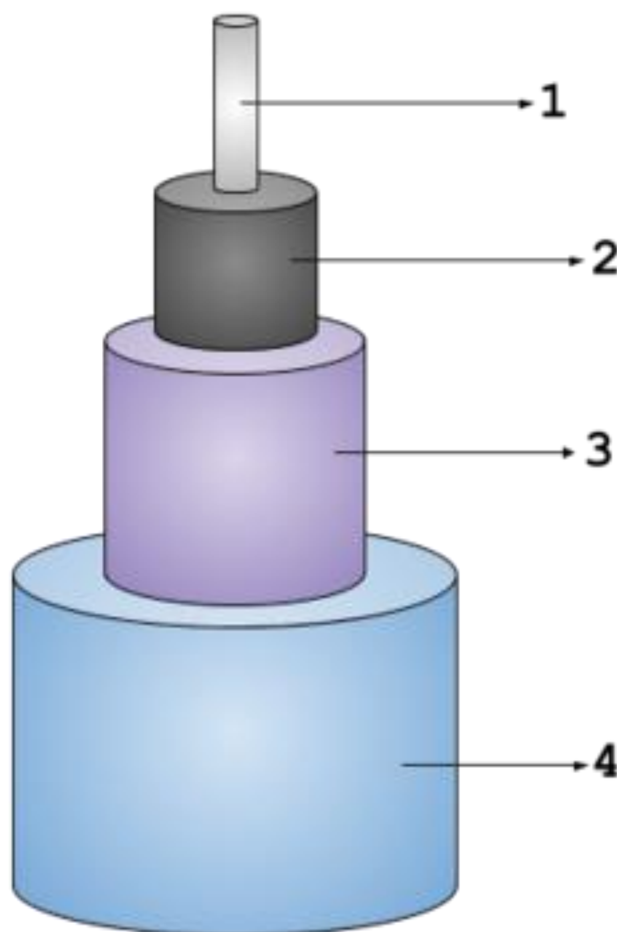
Osim navedenih, postoje i mrežne kartice s takozvanim **SFP, SFP+** ili sličnim utorima. U njih je potrebno spojiti **SFP/SFP+** ili slične module, koji uobičajeno na sebi imaju **LC** utičnice. Međutim osim sâme utičnice, pogotovo za optička vlakna, stvar odabira jest i za kakvu modulaciju i snagu signala te za koju vrstu optičkog vlakna je namijenjen konkretni optički ***SFP*** primopredajnik. Pitanje kod odabira optičkog primopredajnika jest i za koju je on vrstu optičkog vlakna namijenjen, jer ih postoje dvije vrste:

- **Jednomodno vlakno** *(Single mode)* – uobičajeno se koristi za veće udaljenosti i veće brzine prijenosa.
- **Višemodno vlakno** *(Multi mode*) – koristi se za kraće udaljenosti.

**Jednomodno** *(engl. single mode)* optičko vlakno odnosno njegova jezgra je vrlo tanka, obično **9 µm**, dok je obloga jezgre obično debljine **125 µm**. Kroz ovu vrstu optičkog vlakna mogu prolaziti svjetlosni impulsi jednog *modaliteta* (usmjerenosti) pod malim ulaznim kutom, stoga minimalno dolazi do refleksije ili odstupanja od osi vlakna, te svjetlosni impulsi koji su nosioci signala (podataka) vrlo malo degradiraju. Dodatno, oni mogu istovremeno koristiti različite valne duljine što omogućava velike brzine. Zbog ovih karakteristika i valnih duljina na kojima se odašilju svjetlosni impulsi ovakva vlakna i pripadajući optički primopredajnici koriste se za veće ili vrlo velike udaljenosti, čak i više stotina kilometara. Jedini problem jest vrlo visoka cijena ovakvih optičkih vlakana i primopredajnika, ali i opreme za povezivanje odnosno takozvano zavarivanje spojeva i konektora.

**Višemodna** *(engl. multi mode)* optička vlakna znatno su deblja, jezgra je obično **50 µm** ili **62.5 µm**, a obloga jezgre je također debljine **125 µm**. Zbog toga se kroz nju istovremeno može slati više optičkih impulsa različitih modaliteta (usmjerenosti) i s više valnih duljina, ali uz puno više refleksija, a sâmim time i slabljenja signala. Ove karakteristike također omogućavaju velike brzine jer se istovremeno može slati više signala (podataka), ali na manje udaljenosti i to obično do nekoliko stotina metara ili maksimalno nekoliko kilometara.

**Slika 146. Pogled na presjek optičkog kabela (jednog vlakna).**




Pogledajmo presjek jednog optičkog vlakna, na slici **146.** (lijevo).

1. **Jezgra**: 9 µm promjer za *single mode*, te 50µm ili 62.5µm za *multimode*. Jezgra je izrađena od prozirnog materijala, koji ima što bolja optička svojstva za provođenje svjetlosti točno određenih (definiranih) valnih duljina.

2. **Obloga**: 125 µm promjer. Obloga se izrađuje od materijala koji ima znatno manji indeks refrakcije od jezgre pa se stoga svjetlosna zraka od njega odbija nazad prema jezgri, te putuje dalje.

3. **Razdjelnik**: 250 µm promjer. Razdjelnik može biti od različitih materijala, a kod kabela za vanjsku primjenu on je napravljen od izuzetno izdržljivih materijala otpornih na istezanje (vlak).
Ovakvi materijali su iz obitelji *aramida* poput ***kevlara*** i ***twaron*a**.
Uloga ovog sloja je mehanička zaštita optičkog vlakna (ili više njih).
Debljina i promjer ovise o broju optičkih vlakana.

4. **Omot**: 400 µm promjer. On čini vanjski sloj kabela. Debljina i promjer također ovise o broju optičkih vlakna unutar kabela.

---

U svakom slučaju, bez obzira koristila se *single mode* ili *multi mode* optička vlakna, ona prenose svijetlost. Međutim brzina svjetlosti koja prolazi kroz optičko vlakno je manja od brzine svjetlosti koja prolazi kroz vakuum. Pojednostavljeno, što je gušći medij kroz koji prolazi svjetlost, to svjetlost sporije prolazi kroz njega. Brzina svjetlosti kroz optička vlakna varira između 180.000 km/s do 202.000 km/s.

To znači da zbog fizikalnih zakona dolazi do latencije (kašnjenja) oko 5 µs (mikro sekundi) po kilometru optičkog vlakna. Pogledajmo tablicu s latencijom po kilometru, za svjetlost u svjetlovodu odnosno optičkom vlaknu:

| **Latencija** | **250 µs** | **500 µs** | **1 ms (1.000 µs)** | **2,5 ms (2.500 µs)** | **5 ms (5.000 µs)** |
|---|---|---|---|---|---|
| **Udaljenost** | **50 km** | **100 km** | **200 km** | **500 km** | **1.000 km** |

## Karakteristike *multimode* i *single mode* vlakana

**Multimode (MM) vlakna**
Prema međunarodnim standardima **ISO/IEC 11801** i **TIA-492AAAD** definirano je pet kategorija *multimode* vlakana, prema promjeru jezgre vlakna, širini pojasa te performansama na određenim valnim duljinama. Karakteristike su vidljive u tablici:

| Naziv | Boja kabela | Promjer jezgre | Podržane valne duljine | Maks. udalj. 1 Gbps | Maks. udalj. 10 Gbps | Maks. udalj. 25 Gbps | Maks. udalj. 40 Gbps | Maks. udalj. 100 Gbps | Maks. udalj. 200 Gbps | Maks. udalj. 400 Gbps |
|---|---|---|---|---|---|---|---|---|---|---|
| **OM1** | Narančasta | 62.5 µm | 850 nm, 1300 nm | 275 m | 30 m | - | - | - | - | - |
| **OM2** | Narančasta | 50 µm | 850 nm, 1300 nm | 550 m | 80 m | - | - | - | - | - |
| **OM3** | Aqua | 50 µm | 850 nm, 1300 nm | 1000 m | 300 m | 70 m | 100 m | 70 m | - | - |
| **OM4** | Pink | 50 µm | 850 nm, 1300 nm | 1100 m | 550 m | 100 m | 150 m | 100 m | 70 m | - |
| **OM5** | Limunsko zelena | 50 µm | 850 nm, 880 nm, 910 nm, 940 nm | 1100 m | 550 m | 100 m | 150 m | 150 m | 150 m | 100 m |

***Navedene maksimalne udaljenosti variraju, ovisno o vlaknu i primopredajniku.***

Oznaka **OM** označava "*Optical Multimode*". **OM1** i **OM2** vlakna nisu prikladna za brzine prijenosa iznad 10 Gbps zbog njihovih ograničenih širina pojasa i veće disperzije svjetla. Za brzine iznad **100 Gbps**, obično se koriste **OM4** ili **OM5** za kraće udaljenosti. **OM5** vlakna su radi maksimalne učinkovitosti optimizirana za kraće valne duljine: od 850nm do 940nm i za **Shortwave WDM**.

**Single mode (SM) vlakna**
*Single-mode* vlakna (**OS1, OS2**) zbog njihovih superiornih karakteristika kao što su niža disperzija i veća širina pojasa, bolje su prilagođena za vrlo visoke brzine prijenosa podataka na većim ili izrazito velikim udaljenostima (pr. do 200km). Oznaka **OS** označava "*Optical Single Mode*". **OS1** kabeli su namijenjeni za unutarnje polaganje dok su **OS2** kabeli uobičajeno posebno zaštićeni jer su namijenjeni za vanjsko polaganje odnosno polaganje: u (vanjske) cijevi, pod zemlju, u vodu (pr. kao prekomorski kabeli) ili zrakom – na telefonskim ili električnim stupovima ili na raznim nosačima između zgrada i slično.
Pogledajmo njihovu kategorizaciju i osnovne karakteristike u tablici:

| Naziv | Boja | Promjer jezgre | Maks. udalj. za 1 Gbps | Maks. udalj. za 10 Gbps | Maks. udalj. za 25 Gbps | Maks. udalj. za 40 Gbps | Maks. udalj. za 100 Gbps | Maks. udalj. za 200 Gbps | Maks. udalj. za 400 Gbps | Maks. udalj. za 800 Gbps |
|---|---|---|---|---|---|---|---|---|---|---|
| **OS1** | *Žuta | 9 µm | 10 km | 10 km | 10 km | 10 km | 10 km | 10 km | 10 km | 10 km |
| **OS2** | *Žuta | 9 µm | 200 km | 200 km | 200 km | 200 km | 200 km | 200 km | 200 km | 200 km |

***Maksimalne udaljenosti primarno ovise o konkretnom optičkom vlaknu/kabelu, ali i pripadajućem primopredajniku!***
**** SM Prespojni/patch kabeli su uobičajeno OS2 i žute su boje, dok vanjski OS2 mogu biti i drugih boja!***

## Modulacija i kodiranje te multipleksiranje signala

*Modulacija* signala u optičkom vlaknu predstavlja proces promjene karakteristika svjetlosnog signala kako bi se pomoću njega prenijele informacije. Pogledajmo neke od mehanizama moduliranja svjetlosnog signala (o njima detaljnije nešto kasnije):

- **NRZ** (*Non-Return-to-Zero*) - standardna modulacija za niže brzine prijenosa.
- **PAM-4** (*Pulse Amplitude Modulation*) - koristi četiri razine amplitude za prijenos više podataka po simbolu, što je efikasnije za veće brzine prijenosa.
- **DP-16QAM** (*Dual Polarization 16-Quadrature Amplitude Modulation*) - kombinira amplitude i faze za još veću efikasnost iskorištenja spektra.
- **DP-64QAM** i **DP-256QAM** – napredne modulacije koje omogućuju još veće brzine prijenosa koristeći veći broj amplituda i faznih razina.

**Kodiranje** je proces pretvorbe digitalnih podataka u oblik prikladan za prijenos kroz optičko vlakno. U procesu kodiranja dodatno je moguće koristiti i mehanizme za otkrivanje i ispravljanje grešaka u prijenosu (tzv. **FEC**). Uobičajeno se koriste:

- **8b/10b** – kodiranje koje osigurava dovoljan broj prijelaza signala za sinkronizaciju.
- **64b/66b** – efikasnije kodiranje s manje zaglavlja od 8b/10b, koristi se za brže prijenose.
- **256b/257b** ili **1024b/1027b** – visoko efikasno kodiranje za vrlo visoke brzine prijenosa.

***Multipleksiranje*** omogućuje prijenos više signala istovremeno kroz jedno optičko vlakno, čime se značajno povećava kapacitet komunikacijskog sustava. Najčešće korištena tehnika *multipleksiranja* se naziva *Wavelength Division Multiplexing* – (**WDM**). Ona omogućuje prijenos više kanala podataka istovremeno korištenjem različitih valnih duljina („*boja*") svjetlosti u istom optičkom vlaknu. Njene pod varijante su:

- ***CWDM*** – koji koristi šire razmake između valnih duljina, a koriste se: 1310nm i 1550nm ili opseg: 1270-1610nm..
- ***DWDM*** – koji koristi uske razmake između valnih duljina, omogućujući prijenos više kanala kroz jedno vlakno, za veće udaljenosti i kapacitete. Koriste se unutar opsega valnih duljina: 1525–1565 nm (**C-band**), ili 1570–1610 nm (**L band**).
- ***SWDM*** (*Short Wavelength Division Multiplexing - za Multimode*) – koristi višestruke valne duljine unutar kratkog valnog spektra (846 nm - 953 nm) za prijenos podataka preko *multimode* vlakana.

Nadalje, kod brzina preko ***100Gbps***, za povećanje kapaciteta prijenosa podataka često se koristi i više prostornih kanala unutar jednog optičkog kabela pomoću tzv. *Space Division Multiplexing* (**SDM**) tehnike *multipleksiranja*. Najjednostavniji oblik **SDM**-a koristi više pojedinačnih optičkih vlakana unutar jednog optičkog kabela. Dok u naprednijim ***SDM*** sustavima, jedno optičko vlakno može imati više jezgara (tzv. *Multi-core Fiber* - **MCF**). Svaka jezgra tada funkcionira kao neovisni kanal za prijenos podataka. ***SDM*** se koristi za povećanje kapaciteta gradskih (**MAN**) mreža te mreža velikog dometa (tzv. **WAN** mreža), kao i za podmorske optičke kabele – za povećanje kapaciteta međunarodnih telekomunikacijskih veza.
Ovisno o potrebnoj udaljenosti, koristi se i različita oprema unutar primopredajnika svjetlosnih signala. Tako imamo:

- **LED** (*Light Emitting Diode*) – koristi se u starijim ili manje zahtjevnim aplikacijama, obično za niže brzine i kraće udaljenosti.
- **VCSEL** (*Vertical-Cavity Surface-Emitting Laser*) – efikasan i troškovno učinkovit za *multimode* vlakna, pogodan za kratke do srednje udaljenosti.
- **DML** (*Directly Modulated Laser*) – koristi se za kratke do srednje udaljenosti, obično u *single-mode* vlaknima.
- **EML** (*Externally Modulated Laser*) – koristi se za visoke brzine prijenosa i duge udaljenosti, pruža bolju kontrolu nad signalom.
- **DFB** (*Distributed Feedback Laser*) - koristi se za *single-mode* aplikacije, pruža stabilan i čist signal za velike udaljenosti.
- ***Coherent Detection*** - napredna tehnologija koja koristi fazu i amplitudu za prijenos, omogućava vrlo velike brzine prijenosa na dugim udaljenostima.

Pogledajmo zbirnu tablicu (samo nekih od mnoštva varijanti), do sada naučenih komponenti uključenih u rad svjetlovoda:

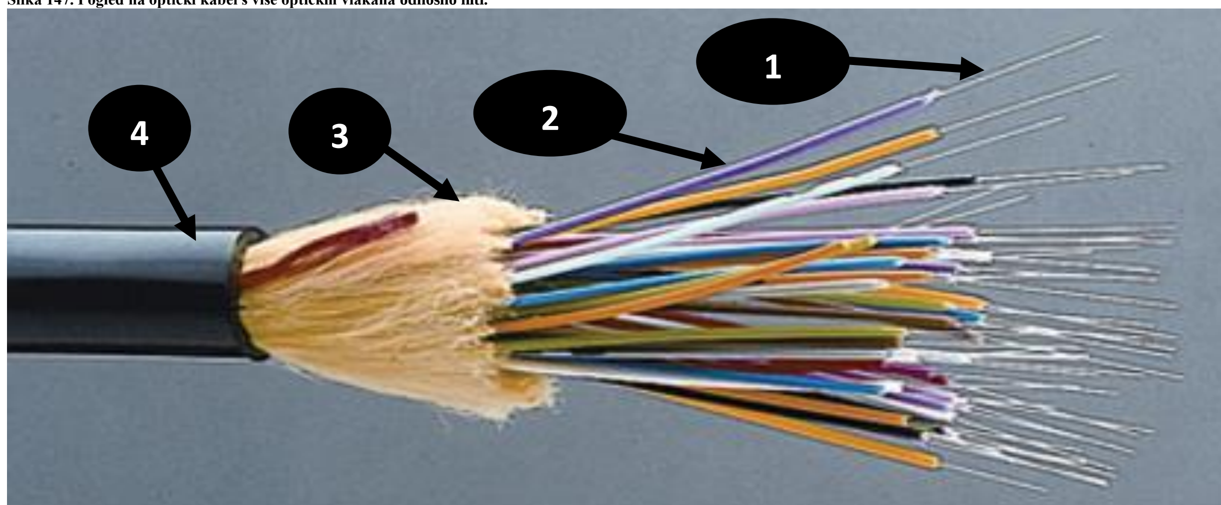
| **Brzina prijenosa** | **Vrsta optike** | **Modulacija** | **Kodiranje** | **Tehnologija moduliranja svjetlosnog signala** | **Vrsta WDM multipleksiranja** | **Maks. udaljenost (približno) Ili min. → max.** |
|---|---|---|---|---|---|---|
| 1 Gbps | **MM** (OM1-OM4) | NRZ | 8b/10b | LED | CWDM | 275 m (OM1),<br>550 m (OM2),<br>1000 m (OM3),<br>1100 m (OM4) |
| 1 Gbps | **SM** (OS1/OS2) | NRZ | 8b/10b | DFB laser | CWDM | 10 km (OS1),<br>20 km (OS2) |
| 10 Gbps | **MM** (OM3/OM4) | NRZ | 64b/66b | VCSEL | CWDM | 300 m (OM3),<br>550 m (OM4) |
| 10 Gbps | **SM** (OS1/OS2) | NRZ | 64b/66b | DFB laser | CWDM/DWDM | 10 km (OS1),<br>40 km (OS2) |
| 25 Gbps | **MM** (OM3-OM4) | NRZ,<br>PAM-4 | 64b/66b | VCSEL,<br>DML | CWDM | 70 m (OM3),<br>100 m (OM4) |
| 25 Gbps | **SM** (OS1/OS2) | NRZ,<br>PAM-4 | 64b/66b | EML,<br>DFB laser | CWDM/DWDM | 10-40 km (OS2) |
| 40 Gbps | **MM** (OM3/OM4) | NRZ | 64b/66b | VCSEL | CWDM | 100 m (OM3),<br>150 m (OM4) |
| 40 Gbps | **SM** (OS1/OS2) | NRZ,<br>PAM-4 | 64b/66b | DML,<br>DFB laser | DWDM | 10-40 km (OS2) |
| 100 Gbps | **MM** (OM3/OM4) | PAM-4 | 256b/257b | VCSEL | CWDM | 100 m (OM3),<br>150 m (OM4) |
| 100 Gbps | **SM** (OS1/OS2) | NRZ,<br>DP-QPSK | 256b/257b | DFB laser | DWDM | 80-200 km (OS2) |
| 200 Gbps | **MM** (OM4/OM5) | PAM-4 | 256b/257b | VCSEL | CWDM | 70m (OM4)<br>1500m (OM5) |
| 200 Gbps | **MM (**OM5) | PAM-4 | 256b/257b | VCSEL | SWDM | 150m (OM5) |
| 200 Gbps | **SM** (OS2) | DP-16QAM | 256b/257b,<br>1024b/1027b | EML,<br>DFB laser | DWDM | 10,80-200 km (OS2) |
| 400 Gbps | **MM** (OM5) | PAM-4 | 256b/257b | VCSEL | SWDM | 100m (OM5) |
| 400 Gbps | **SM** (OS2) | DP-16QAM,<br>DP-64QAM | 256b/257b,<br>1024b/1027b | EML,<br>DFB laser | DWDM | 10, 20, 40km ili<br>80-200 km (OS2) |
| 800 Gbps | **SM** (OS2) | DP-64QAM,<br>DP-256QAM | 256b/257b,<br>1024b/1027b | EML,<br>DFB laser | DWDM | 10, 20, 40km ili<br>80-200 km (OS2) |



Ako primjerice pokušavate spojiti mrežne uređaje koristeći **SM OS2** vlakno s primopredajnikom koji koristi **EML** ili **DFB** lasere velike snage, na udaljenosti manjoj od 80 km, najvjerojatniji problem koji će se dogoditi jest preopterećenje (ili uništenje) prijemnika zbog prejakog signala. Stoga pripazite pri odabiru pravilnog primopredajnika (optičkih modula manje snage) ili koristite optičke ***atenuatore*** (oslabljivače signala) ili prilagodite izlaznu snagu lasera.

Standardni optički kabeli u praksi imaju više optičkih vlakana, od uobičajeno osam, pa sve do nekoliko stotina. Pogledajmo sliku 147 jednog optičkog kabela koji se sastoji od nekoliko desetaka optičkih vlakana (s oznakama dijelova/slojeva kabela [1 - 4]).

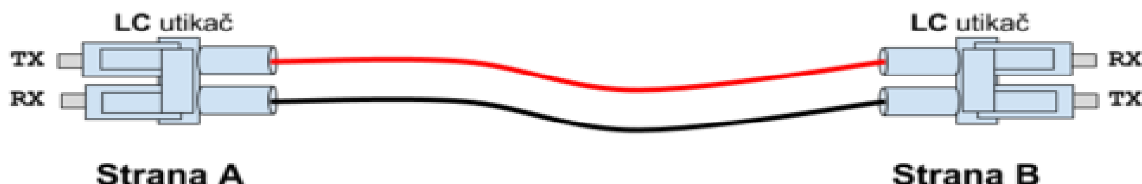
**Slika 147. Pogled na optički kabel s više optičkih vlakana odnosno niti.**




Izvor slike: https://fiberopticsof.wordpress.com/2015/05/22/fiber-optic-cables-structure-termination-and-application/

Dodatno je važno znati kako je za komunikaciju uobičajeno potreban pâr optičkih vlakana (dva komada*) jer se jedno optičko vlakno koristi za slanje, a drugo za primanje signala. Dakle kada strana **A** komunicira sa stranom **B** to se odvija u dva smjera:
Strana **A** koristi jedno optičko vlakno za slanje signala (***TX***) dok je na drugoj strani to vlakno spojeno na prijemni dio (***RX***).
S druge strane je također ***TX*** strana spojena na odašiljački dio, koji na drugoj strani završava na prijemnom djelu, kao na slici **148**.

**Slika 148. Pogled na LC utikač i vezu (LC → LC).**




Na slici **148** su vidljivi **LC** utikači (konektori) na obje strane optičkog kabela odnosno vlakna (svjetlovoda).

---

* *Postoje i tehnologije poput **PON (za FTTH)** koje koriste samo jedno optičko vlakno za dvosmjernu komunikaciju!*

**Sažetak:** optička vlakna su ključna komponenta modernih računalnih mreža zbog svojih izvanrednih karakteristika:

- **Visoka propusnost** – optička vlakna mogu prenijeti ogromne količine podataka, što ih čini idealnim za mreže koje povezuju različite (ključne) dijelove mrežne infrastrukture. U 2023. godini rekordna testna brzina prijenosa na *single mode* optičkom vlaknu je iznosila **22,9 Petabita** u sekundi (24.012.390 Gbps) (**1693**).
- **Brzina** – svjetlosni signali omogućuju vrlo brzu komunikaciju s minimalnim kašnjenjem, što je kritično za aplikacije koje zahtijevaju obradu podataka u stvarnom vremenu.
- **Dug domet** – optička vlakna omogućuju prijenos podataka na vrlo velike udaljenosti bez potrebe za pojačalima, za razliku od bakrenih kabela koji zahtijevaju česte pojačivače signala. Za velike udaljenosti se koriste ***Sigle mode*** vlakna (*Single mode Fiber* [**SMF**]), a kao izvori svjetla se tada koriste posebni laseri. Pogledajmo skraćenu tablicu s udaljenostima koje se mogu postići sa *Single mode Fiber* (**SMF)** kabelima i pripadajućim optičkim primopredajnicima:

| Vrsta kabela | Standard | Maksimalna duljina (bez pojačala) |
|---|---|---|
| **OS1** | **ITU-T G.652A/B/C/D** | **Do 10 km** |
| **OS2** | **ITU-T G.652C/D<br>ITU-T G.657.A1/A2,…** | **Do 200 km** |

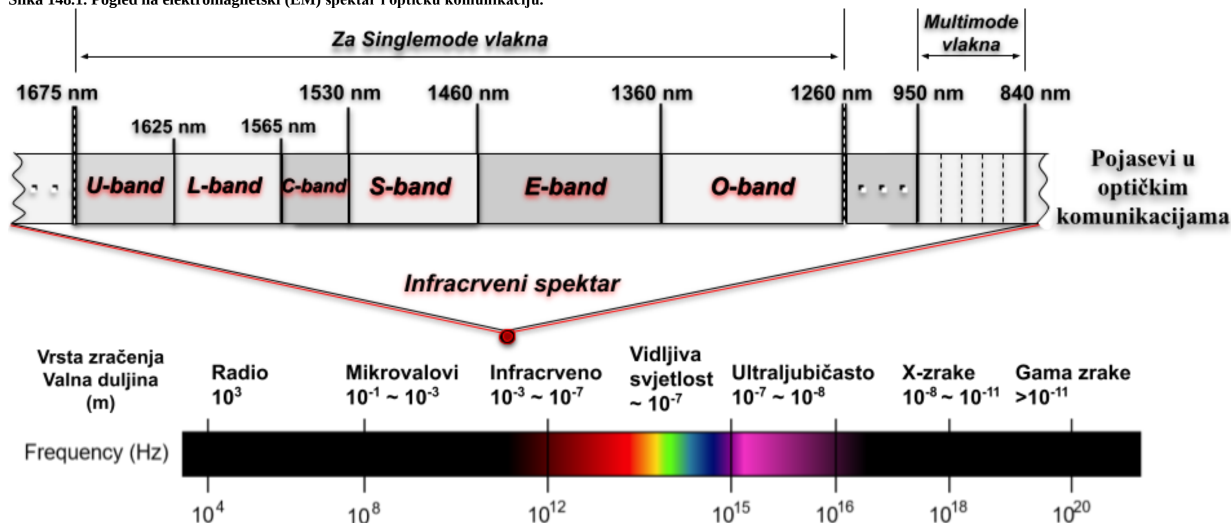
Prema **ISO/IEC 11801**: **OS1** su kabeli za unutarnju upotrebu, dok su **OS2** uobičajeno za vanjsku upotrebu, iako u praksi možemo pronaći čak i prespojne (*patch*) optičke kabele koji su **OS2**.

- **Manja Interferencija** – optička vlakna nisu podložna elektromagnetskim smetnjama, što ih čini pouzdanijima za upotrebu u okruženjima s puno elektromagnetske buke odnosno smetnji.
- **Sigurnost** – teže je neovlašteno presresti podatke prenesene optičkim vlaknima u usporedbi s električnim signalima, što povećava sigurnost podataka.

**Optička komunikacija detaljnije**

Optička komunikacija temelji se na pretvorbi električnih signala u svjetlosne impulse koji se potom prenose kroz optička vlakna. Valne duljine tih svjetlosnih impulsa su unutar infracrvenog dijela elektromagnetskog (**EM)** spektra, kako je vidljivo na slijedećoj slici. Za *singlemode* vlakna se konkretno radi o rasponu od 1260 nm do 1675 nm koji je idealan zbog niskih gubitaka i disperzije u optičkim vlaknima, kao i učinkovitih pojačala koja su razvijena za te valne duljine. Dodatno je i taj raspon valnih duljina podijeljen u nekoliko segmenata jer svaki segment ima specifične prednosti i nedostatke u pogledu gubitaka i disperzije svjetla. Za *multimode* vlakna se koristi drugi dio spektra (840nm – 950nm) koji je pogodan za velike brine, ali vrlo male udaljenosti.
U konačnici ovi impulsi svjetla predstavljaju digitalne podatke (bitove) koje je potrebno prenijeti od izvora do odredišta.

**Slika 148.1. Pogled na elektromagnetski (EM) spektar i optičku komunikaciju.**




**Detaljnije o pojasevima u optičkim komunikacijama**

U optičkim komunikacijama za računalne mreže koristi se više "*bandova*" odnosno pojaseva unutar dijela infracrvenog spektra koji se koristi za prijenos podataka. Ti pojasevi su definirani na temelju valnih duljina svjetlosti, a svaki od njih ima specifična svojstva, uključujući gubitke prilikom prijenosa te dostupnu propusnost. Osnovni definirani pojasevi su:

- **O-band** (*Original band*) – valne duljine: 1260 - 1360 nm. Namjena: koristi se za prijenos signala na kratkim udaljenostima zbog niskog disperzijskog gubitka na tim valnim duljinama. Ovo je povijesno prvi pojas koji se koristio u optičkim mrežama.
- **E-band** (*Extended band*) – valne duljine: 1360 - 1460 nm. Namjena: proširenje *O-banda* za dodatne kanale, koristi se manje zbog povećanih gubitaka u ovom pojasu.
- **S-band** (*Short band*) – valne duljine: 1460 - 1530 nm. Namjena: rjeđe se koristi, ali može biti koristan u specifičnim primjenama koje zahtijevaju dodatne kanale (izvan *C-pojasa*).
- **C-band** (*Conventional band*) - valne duljine: 1530 - 1565 nm. Namjena: najčešće korišten pojas u optičkim komunikacijama jer nudi niske gubitke i visoku propusnost. Uglavnom se koristi za **DWDM** sustave.
- **L-band** (*Long band*) – valne duljine: 1565 - 1625 nm. Namjena: proširenje *C-banda* kako bi se povećao broj kanala u **DWDM** sustavima.
- **U-band** (*Ultralong band*) – valne duljine: 1625 - 1675 nm. Namjena: Još uvijek eksperimentalni pojas s potencijalom za buduće proširenje kapaciteta.

Različiti pojasevi imaju različite karakteristike u pogledu disperzije i gubitaka pa je odabirom odgovarajućeg pojasa moguće optimizirati performanse mreže. Nadalje, korištenjem više pojaseva istovremeno, moguće je značajno povećati propusnost mreže upotrebom tehnike poput **WDM** (*Wavelength Division Multiplexing*). Osim toga neki pojasevi su bolje prilagođeni za specifične primjene, npr. kraće udaljenosti, veće brzine ili manju latenciju (kašnjenje) i slično.

**Modulacija svjetlosnog signala**

Modulacija svjetlosnog signala ključna je komponenta u optičkoj komunikaciji jer omogućuje prijenos informacija putem svjetlosnog vala. Modulacija u ovom kontekstu odnosi se na promjenu svojstava svjetlosnog vala kako bi se kodirali podaci. Postoji nekoliko metoda modulacije koje se koriste u optičkoj komunikaciji, a svaka ima svoje prednosti i primjene.
Naime, u optičkoj komunikaciji podaci se prenose promjenom određenih svojstava svjetlosnog vala, kao što su: **amplituda** odnosno snaga svjetlosti, **frekvencija** to jest valna duljina svjetlosti i **faza** to jest relativna pozicija vala u vremenu.
Postoji više vrsta modulacija koje se koriste u optičkoj komunikaciji, a mi ćemo navesti samo neke:

- ***On-Off Keying*** (**OOK**) – najjednostavniji oblik modulacije u kojem se svjetlosni izvor uključuje i isključuje (tj. svijetli ili ne svijetli) kako bi se prikazali podaci u binarnom obliku (nula i jedan). **NRZ** je jedna od varijanti **OOK**.
- ***Amplitude Shift Keying*** (**ASK**) – mijenja amplitudu svjetlosnog signala kako bi se kodirale različite razine podataka. Na primjer, svjetlost s različitim razinama intenziteta može predstavljati različite binarne vrijednosti. **PAM 4** je jedna od varijanti **ASK**, kod koje svjetlosni signal može poprimiti četiri različite amplitude (razine), čime omogućuje prijenos dva bita po simbolu.
- ***Frequency Shift Keying*** (**FSK**) – mijenja frekvenciju svjetlosnog vala za kodiranje informacija. Svaka frekvencija odgovara određenoj vrijednosti podataka.
- ***Phase Shift Keying*** (**PSK**) – mijenja fazu svjetlosnog signala za prijenos podataka. U optičkoj komunikaciji, PSK može biti vrlo učinkovit jer omogućuje prijenos više podataka unutar istog spektra. Ovdje postoje i podvrste: ***Binary PSK*** (**BPSK**) – koristi dvije faze, te *Quadrature PSK* (**QPSK**) - koji koristi četiri faze te *Differential Phase Shift Keying* (**DPSK**) koji koristi razlike u fazi između uzastopnih signala za kodiranje podataka.
- ***Quadrature Amplitude Modulation*** (**QAM**) – *QAM* je kombinacija amplitudne i fazne modulacije. Koristeći više razina amplituda i faza, QAM omogućuje prijenos najviše bitova podataka po simbolu. Svaki simbol predstavlja određenu kombinaciju amplitude i faze. Kombinacija tih vrijednosti kodira određeni broj bitova informacija.

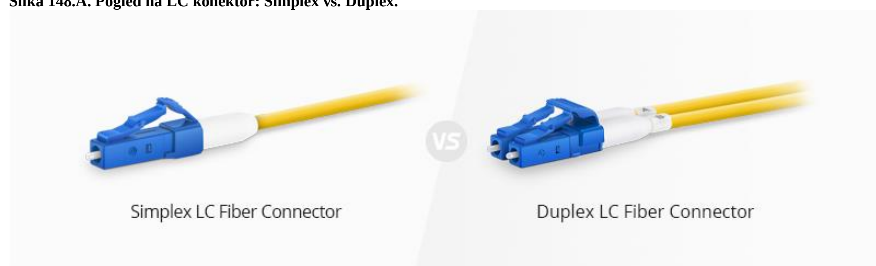
**Izvori informacija: (1064),(1065),(1682),(1683),(1684),(1685),(1686),(1687),(1688),(1689),(1690),(1691),(1692),(1693),(1694),(1695),(1741),(1742),(1744), ISO/IEC 11801, ITU-T G.652A/B/C/D/, ITU-T G.654, ITU-T G.656, ITU-T G.657.**

### 19.2.1.2.1. Utikači i utičnice za optička vlakna

Utikači i utičnice odnosno tzv. konektor optičkih vlakana koriste se za povezivanje optičkih vlakana, olakšavajući učinkovit prijenos svjetlosnih signala. Upotrebom konektora (utikača i utičnica) optičkih vlakana omogućuje se njihovo brže spajanje i odspajanje. Oni dolaze u različitim vrstama kao što su **SC**, **LC**, **ST** ili **MTP**, a svaki od njih je dizajniran za posebne primjene. Na tržištu je dostupno sveukupno oko 100 različitih vrsta konektora za optička vlakna, iako se od svih njih najčešće koristi samo nekolicina.

**Simplex vs. Duplex konektori**
*Simplex* veza znači da se signali šalju u jednom smjeru; signal se prenosi kroz dva *simplex* konektora i *simplex* optički kabel od uređaja A do uređaja B. Dakle za prijenos signala se koristi jedno optičko vlakno za slanje, a jedno za primanje signala, bez obzira koristi li se *Singlemode* ili *Multimode* vlakno. Tako je i s konektorima, kako je vidljivo na donjoj slici.

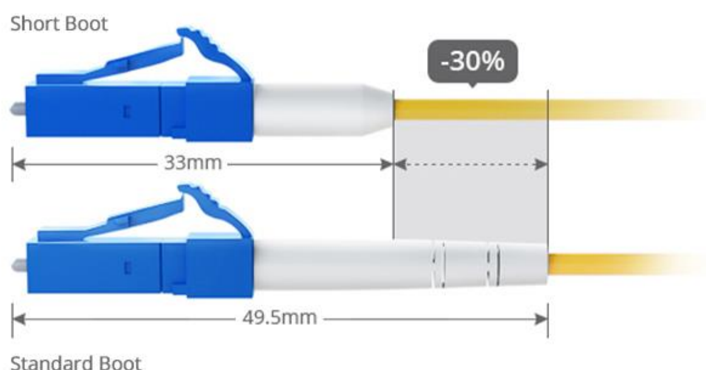
**Slika 148.A. Pogled na LC konektor: Simplex vs. Duplex.**




**Slika je preuzeta sa: https://community.fs.com**

**Duljina stražnje strane konektora**
Što se tiče duljine stražnje strane konektora odnosno svojevrsne uvodnice ili nosača konektora, postoje standardna izvedba i kratka izvedba. Standardni nosač može zaštititi kabel i konektor od oštećenja, izbacivanja niti iz tijela konektora itd.
Dok kratki nastavak ima istu funkciju, a razlikuje se po kraćoj strukturi nosača.

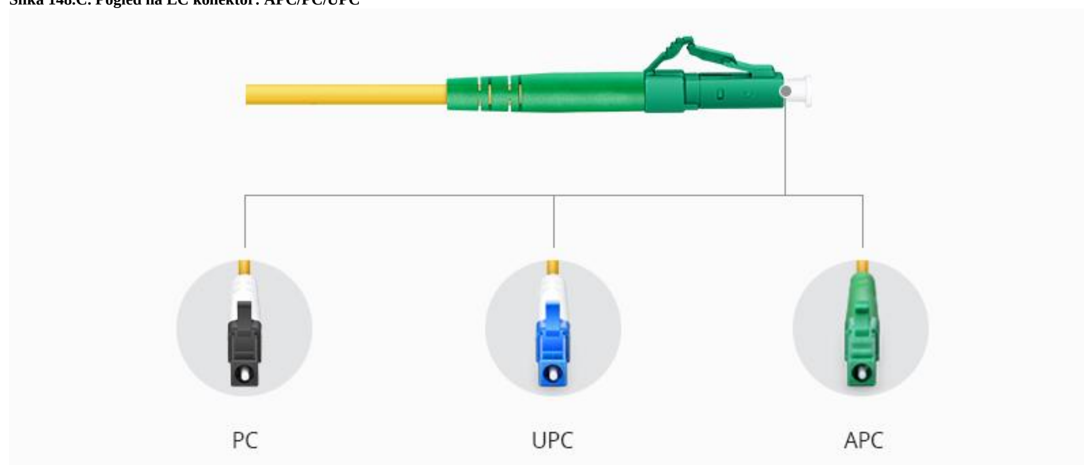
**Slika 148.B. Pogled na LC konektor: pogled na uvodnicu/nosač.**




**Slika je preuzeta sa: https://community.fs.com**

**Način poliranja: APC/PC/UPC konektori za optička vlakna**
Prema vrsti završnog poliranja, konektori optičkih kabela mogu se podijeliti u tri vrste: **PC**, **UPC** i **APC** konektori. Šifra boja pruža prikladnu metodu za identifikaciju ove tri vrste konektora: šifra boje **PC**-a je crna, šifra boje za **APC** konektor je zelena, a za **UPC** konektor je plava.

**Slika 148.C. Pogled na LC konektor: APC/PC/UPC**




**Slika je preuzeta sa: https://community.fs.com**

Jednostavnije rečeno, **PC**, **UPC** i **APC** odnose se na stilove poliranja umetka unutar konektora za optička vlakna, kao što pokazuje sljedeća slika.
Čahura je kućišta za izloženi kraj vlakna, dizajnirana za spajanje na drugo vlakno ili u odašiljač ili prijemnik.

**Slika 148.D. Pogled na vrh konektora i način poliranja (PC).**



Physical Contact Connector

**Slika je preuzeta sa: [https://community.fs.com](https://community.fs.com)**

---

**Slika 148.E. Pogled na vrh konektora i način poliranja (UPC).**



Ultra Physical Contact Connector

**Slika je preuzeta sa: [https://community.fs.com](https://community.fs.com)**

**UPC** konektori izrađeni su posebnom tehnikom poliranja, mijenjajući staklo unutar konektora kako bi laserski signali lakše prolazili. Iako to povećava brzinu signala, također olakšava oštećenje stakla. Ako se **UPC** optički kabel često uključuje/isključuje, on će se brže istrošiti od standardnog ili **APC** ekvivalenta.

**Slika 148.F. Pogled na vrh konektora i način poliranja (APC).**



Angled Physical Contact Connector

**Slika je preuzeta sa: [https://community.fs.com](https://community.fs.com)**

**APC** konektori eliminiraju UPC-ov problem trošenja i habanja tako što staklo u konektorima zakreću pod kutom od 8° uz posebnu tehniku poliranja. Ovaj kut sprječava nastanak dodatne štete, a istovremeno omogućuje čvršće i sigurnije spojeve. Jedina mana je to što kut zahtijeva da se konektor priključi desnom stranom prema gore; za razliku od standardnih i **UPC** konektora, moguće je imati **APC** konektor naopako.

## Vrste utikača (konektora)

Na tržištu postoji oko 100 vrsta optičkih konektora, od kojih ćemo spomenuti samo njih nekoliko najkorištenijih:

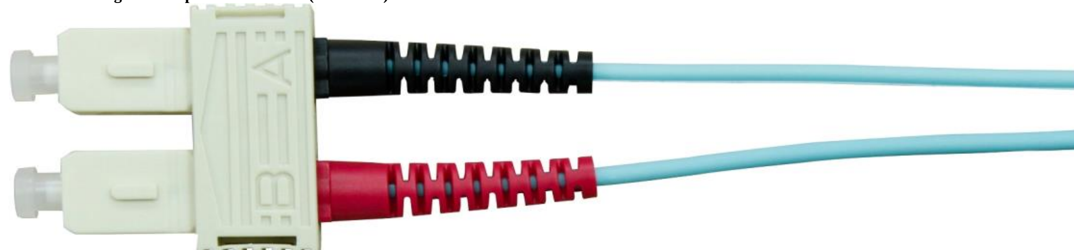
- **LC** konektor - *Lucent konektor* (LC), kategoriziran kao konektor malog oblika (**SFF**), ima čahuru od 1,25 mm. Njegov kompaktni dizajn doveo je ove konektore za optička vlakna do široke popularnosti u podatkovnim komunikacijama. On je očit izbor za visokoučinkovito kabliranje zbog svojih malih dimenzija, Trenutačno se **LC** konektor za optička vlakna ističe kao konektor koji se najviše koristi u raznim primjenama.

**Slika 148.G. Pogled na duplex LC konektor (MM OM3).**



- **SC** konektor - SC konektor bio je početni izbor za standard **TIA 568** i prepoznat je kao uskočni konektor koji se sigurno zabravljuje jednostavnim pokretom guranja i povlačenja. Oznaka "**SC**" potječe od dizajna "kvadratnog (*Square*) konektora", kojeg karakterizira kućište konektora četvrtastog oblika. S 2,5 mm čahurom. Dvostrukom veličinom od prethodnog LC konektora — SC optički konektor nalazi optimalnu upotrebu u podatkovnim komunikacijama i telekomunikacijskim primjenama, uključujući *point-to-point* i pasivno optičko umrežavanje. Zahvaljujući svojim izvanrednim performansama, SC konektor za optička vlakna ostaje drugi najčešći izbor.

**Slika 148.H. Pogled na duplex SC konektor (MM OM3).**



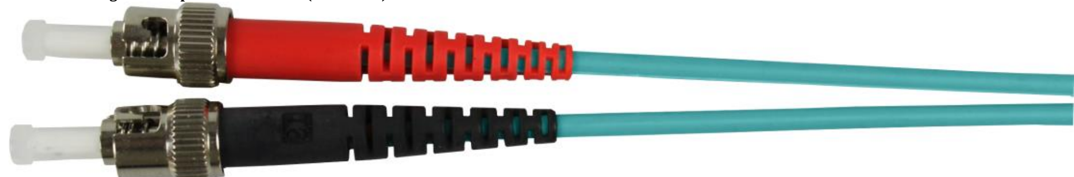
- **FC** konektor - "FC" je kratica za spojnicu s čahurom (*Ferrule Connector*). **FC** konektor za optička vlakna ističe se kao prvi konektor za optička vlakna koji uključuje keramičku čahuru. Za razliku od **SC** i **LC** konektora s plastičnim kućištem, FC konektor ima okrugli vijak izrađen od poniklanog ili nehrđajućeg čelika. Kako bi se osiguralo ispravno umetanje, FC konektor za optička vlakna koristi ključ za poravnanje, koji se zatim učvršćuje u adapter/utičnicu pomoću stezne čahure s navojem. Unatoč dodatnoj složenosti u proizvodnji i instalaciji, FC konektori i dalje imaju prednost u primjeni kod preciznih instrumenata kao što su OTDR-ovi te ostaju važan izbor za primjene s jednomodnim vlaknima.

**Slika 148.I. Pogled na duplex FC konektor (MM OM2) .**



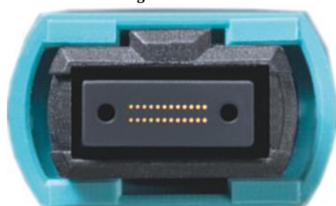
- **ST** konektor – ST (*Straight Tip*) optički konektor, koji je razvio i licencirao AT&T nedugo nakon uvođenja tipa **FC**, sadrži keramičku, opružnu čahuru od 2,5 mm pričvršćenu na mjesto bajunetnim nosačem. Ovaj konektor se obično koristi u primjenama na kraćim udaljenostima, kao što su instalacije višemodnih vlakana u kampusu i zgradama, korporativna mrežna okruženja i vojne aplikacije.

**Slika 148.J. Pogled na duplex ST konektor (MM OM3).**



- **MTP/MPO** konektor – za razliku od prethodnih konektora za optička vlakna, MTP/MPO konektor za optička vlakna ističe se kao konektor za više vlakana istovremeno, nadmašujući druge po veličini. Ovaj konektor spaja više vlakana, u rasponu od 12 do 24, unutar jedne pravokutne čahure. Primarno korišteni u 40G i 100G optičkim paralelnim vezama velike propusnosti, MTP/MPO konektori su prilično složeni, ali često u primjeni za veće brzine (100+Gbps).

**Slika 148.K. Pogled na MPO konektor.**



**Slika 148.L. Pogled na razne vrste optičkih konektora i kabela.**




**Slika je preuzeta s: https://community.fs.com.**

Na slici vidimo različite vrste konektora za optička vlakna s pripadajućim (spojenim) optičkim kabelima različitih kategorija:

- (A) – ***FC konektor simplex*** [zeleni=***APC***], kabel [žuti=***single mode***].
- (B) – ***ST konektor simplex*** [crni=***UPC***], kabel [aqua-plavi=***multi mode – OM3***].
- (C) – ***SC konektor simplex*** [sivi=***UPC***], kabel [aqua-plavi=***multi mode – OM3***].
- (D) – ***LC konektor simplex*** [sivi=***UPC***], kabel [aqua-plavi=***multi mode – OM3***].
- (E) – ***SC konektor simplex*** [zeleni=***APC***], kabel [žuti=***single mode***].
- (F) – ***SC konektor duplex*** [sivi=***UPC***], kabel [aqua-plavi=***multi mode – OM3***].
- (G) – ***LC konektor duplex*** [zeleni=***APC***], kabel [aqua-plavi=***multi mode – OM3***].
- (H) – ***MTP konektor***, kabel [pink=***multi mode – OM4***].
- (I) – ***FC konektor duplex*** [crni=***UPC***], kabel [aqua-plavi=***multi mode – OM3***].
- (J) – ***LC konektor simplex*** [zeleni=***APC***], kabel [žuta=***single mode***].
- (K) – ***SC konektor simplex*** [sivi=***UPC***], kabel [aqua-plavi=***multi mode – OM3***].
- (L) – ***ST konektor simplex*** [zeleni=***APC***], kabel [žuta=***single mode***].
- (M) – ***LC konektor simplex*** [plavi=***UPC***], kabel [žuta=***single mode***].

Prema udruženju *Fiber Optic Association* (**FOA**) definirani su i certifikati za znanja i vještine potrebne za polaganje i spajanje optičkih kabela i konektora te optičkih komponenti (prespojnice, patch/prespojni paneli i slično).
Naime *Fiber Optic Association* (**FOA**) međunarodno je profesionalno društvo za optička vlakna. *FOA* je 1995. godine osnovala skupina trenera iz industrije, vlade i obrazovanja koji su željeli stvoriti industrijske standarde za obuku i certificiranje tehničara za optička vlakna. FOA je neprofitna organizacija sa sjedištem u Kaliforniji, SAD, koja ima više od 200 pridruženih organizacija za obuku u više od 40 zemalja. FOA odobrava organizacije za obuku za rad s optičkim vlaknima i certificira njihove instruktore koji obučavaju projektante, instalatere i operatere svih vrsta mreža s optičkim vlaknima.
Neki od **FOA** certifikata su sljedeći:

- Osnovni:
  - **CFOT** - *Certified Fiber Optic Technician*: pokriva osnove optičkih vlakana za sve aplikacije
  - **CPCT** - *Certified Premises Cabling Technician*: pokriva optičke, bakrene i bežične instalacije.
- Specijalistički:
  - **CFOS/S** - *Certified Fiber Optic Specialist in Splicing.*
  - **CFOS/C** - *Certified Fiber Optic Specialist in Connectors.*
  - **CFOS/T** - *Certified Fiber Optic Specialist in Testing.*
  - **CFOS/O** - *Certified Fiber Optic Specialist in Outside Plant Installation.*
  - **CFOS/D** - *Certified Fiber Optic Specialist in Fiber Optic Network Design.*

**Izvori informacija: (1720),(1721),(1722),(1723).**

### 19.2.1.2.2. Fiber to the Home (FTTH)

*Fiber to the Home* (FTTH) je tehnologija kod koje se optičko vlakno koristi za pristup internetu, tako da se ono instalira i spaja izravno na kućanstvo ili poslovni prostor. Ova tehnologija pruža brze internetske usluge i veću propusnost, a i dalje je superiorna u usporedbi s vezama koje koriste koaksijalne kabele (tzv. *kablovski internet*) ili **(A)DSL** (*Digital Subscriber Line*).
Važno je znati da je vršni pojam *Fiber to the Home* (*vlakna do doma*) jedna od iteracija pod krovnom kategorijom "**FTT**x", koja identificira bilo koji dizajn širokopojasne mreže koji povezuje optička vlakna izravno s određenom priključnom točkom od mreže telekoma (ISP-a) prema kućanstvu (kućama, stanovima ili uredima).
Ostali pojmovi odnosno segmenti **FTTx** vezanih za sâmo kabliranje uključuju (pogledajte sliku **148.M**):

- *Fiber to the Curb* (**FTTC**) – odnosi se na optičko vlakno dovedeno do rubova ulice (ili naselja) to jest blizu korisničkog područja, ali ne direktno do odredišne zgrade ili kuće.
- *Fiber to the Building* (**FTTB**) – optičko vlakno se dovodi do zgrade (npr. stambene zgrade, poslovne zgrade) i završava u poslovnom prostoru, pristupnoj kutiji ili TK ormariću unutar (ili ispred) zgrade.
- *Fiber to the Premises* (**FTTP**) – označava da se optičko vlakno dovodi direktno sve do krajnjeg korisnika (*premises*) odnosno kuće, stana ili poslovnog prostora.

U svakom slučaju **FTTH** je atraktivno rješenje pri izgradnji mreža u budućnosti jer je tehnologija temeljena na optičkima vlaknima dokazana i proširiva, što znači da će moći podržati širokopojasne zahtjeve u doglednoj budućnosti. Osim toga optički (svjetlovodni) kabeli imaju gotovo neograničen kapacitet. *FTTH* tehnologije koje se danas koriste mogu prenositi informacije desetke ili stotine puta dalje i stotine puta brže od bakrenih parica. U usporedbi s bakrom, optičko vlakno je mnogo sigurnije, pouzdanije i izdržljivije, a čak i na vrlo velikim udaljenostima ima vrlo male gubitke signala.
Budući da optički kabeli prenose informacije putem svjetlosnih impulsa, otporni su na električne smetnje i naponske udare. Osim toga, svjetlovodni kabeli dizajnirani su da traju 30-50 godina,
Za ***FTTH*** se koristi takozvana *pasivna optička mreža* tj. „*Passive Optical Network*“ (**PON**). To je telekomunikacijska mreža načinjena od optičkih vlakana koja za prijenos signala od krajnje točke telekoma do kućanstva koristi samo uređaje bez napajanja, za razliku od većine ostale (aktivne) elektroničke opreme. U praksi se PON-ovi koriste za tzv. posljednju milju između pružatelja internetskih usluga (ISP) i njihovih korisnika. U ovakvoj primjeni, **PON** ima topologiju *point-to-multipoint* u kojoj ISP (telekom) koristi jedan krajnji uređaj (sa svoje strane mreže) za opsluživanje mnogih uređaja krajnjih korisnika odnosno kućanstava, koristeći tehnologiju kao što je pr. **GPON**.
U ovakvoj topologiji jedno optičko vlakno sa strane telekoma opslužuje mnoga kućanstva tako što se grana u više vlakana putem pasivnog optičkog razdjelnika (*splittera).* Zatim se svako od tih pojedinačnih vlakana koristi za samo jednog krajnjeg korisnika odnosno kućanstvo. Dakle svjetlosni signal od ISP-a (telekoma) dijeli se kroz razdjelnike (*splittere*) kako bi došao do svih krajnjih korisnika (kućanstava). U drugom smjeru se svjetlosni signali iz kućanstava kombiniraju (*multipleksiraju*) i šalju u jedno optičko vlakno prema telekomu (ISP-u), kako je vidljivo na slici **148.M**.

## *Gigabit Passive Optical Network* (GPON) [ITU-T G.984]

Tehnologija koja se najčešće koristi za ***FTTH*** jest *Gigabit Passive Optical Network* (**GPON**) tehnologija (**ITU-T G.984**).
Prva inačica *GPON-a* ratificirana je 2003. godine, a od tada je nekoliko puta proširivana i revidirana.

**Optička vlakna i konektori**
Kod ove *PON* tehnologije za *FTTH* mreže koristi se samo jedno optičko vlakno za dvosmjernu komunikaciju (*upload/download*) a samim time i jedan konektor na kućnom usmjerivaču (*routeru*). Dakle kroz samo jedno optičko vlakno prolaze podaci u oba smjera (tzv. *full duplex* komunikacija). *GPON* koristi *single-mode* optičko vlakno (SMF, **ITU-T G.652**). Ono podržava dug domet i minimalne gubitke signala, što ga čini idealnim za pružanje pouzdanih i stabilnih internet usluga velikom broju korisnika. Što se utikača (konektora) na krajnjoj točki u kućanstvu tiče (kućni *router*), najčešće se koristi **SC/APC** konektor. **APC** kategorija **SC** konektora ima ukoso polirani kraj pod kutom od 8 stupnjeva, što smanjuje refleksiju svijetlosti unazad, čime se poboljšava kvaliteta signala.

U radu GPON-a koriste se *Time Division Multiple Access* (**TDMA**) i *Wavelength Division Multiplexing* **(WDM)** tehnike.
Pri tome ***TDMA*** omogućava dijeljenje jednog optičkog vlakna između više korisnika, dok ***WDM*** omogućuje različitim signalima (*uplink* i *downlink*) korištenje različitih valnih duljina: uobičajeno *1310 nm* za *uplink* i *1490 nm* za *downlink,* što znači da se pomoću *WDM-a* odvija dvosmjerna komunikacija kroz jedno *single-mode* optičko vlakno.
***GPON*** mreža započinje u centralnom čvoru pružatelja usluga, zvanom *Optical Line Terminal* (**OLT**).
*OLT* je odgovoran za upravljanje cijelom *GPON* mrežom te povezivanje s internetom i drugim uslugama.
Od *OLT*-a se proteže po jedno optičko vlakno prema korisnicima. Međutim, ovo vlakno prvo dolazi do pasivnog optičkog razdjelnika (***splitera***) koji se nalazi u blizini korisnika; primjerice unutar **TK** ormarića u ulici ili naselju.
Pasivni optički razdjelnik u navedenom **TK** ormariću dijeli signal s jednog optičkog vlakna od ISP-a (telekoma) na više izlaznih vlakana prema kućanstvima. Na primjer, jedan optički razdjelnik može podijeliti signal na **32, 64** ili čak **128** vlakana, koja se zatim protežu prema različitim kućanstvima, prema princip: jedno vlakno → jedno kućanstvo.
Što se samih optičkih razdjelnika tiče, osim već navedenih koji dijele optički signal na 32, 64 ili čak 128 izlaza odnosno u konačnici optičkih vlakana, postoje i oni koji ih dijele na **4** (1:4) ili **8** (1:8), a dodatno je moguće i njihovo kaskadno povezivanje (jednog na drugi), sve do granice od najčešće 32. Kaskadno spajanje optičkih razdjelnika (*splittera*) se rijetko koristi u urbanim sredinama, a češće u ruralnim, jer smanjuje troškove te ubrzava vrijeme instalacije sustava. Pri tome je važno znati da projektanti optičkog sustava moraju paziti na ograničenja ovakvog (i bilo kojeg drugog) spoja jer svaki optički razdjelnik oslabljuje signal (oznaka u **dB**) što utječe i na maksimalnu duljinu svjetlovoda do kućanstva. Dakle svjetlosni signal gubi snagu kako prolazi kroz vlakno (gubici po kilometru) i kroz razdjelnike. Dodatno viši omjeri podjele u razdjelnicima odnosno dijeljenja signala (poput 1:64) znače da svaki korisnik dobiva slabiji signal.

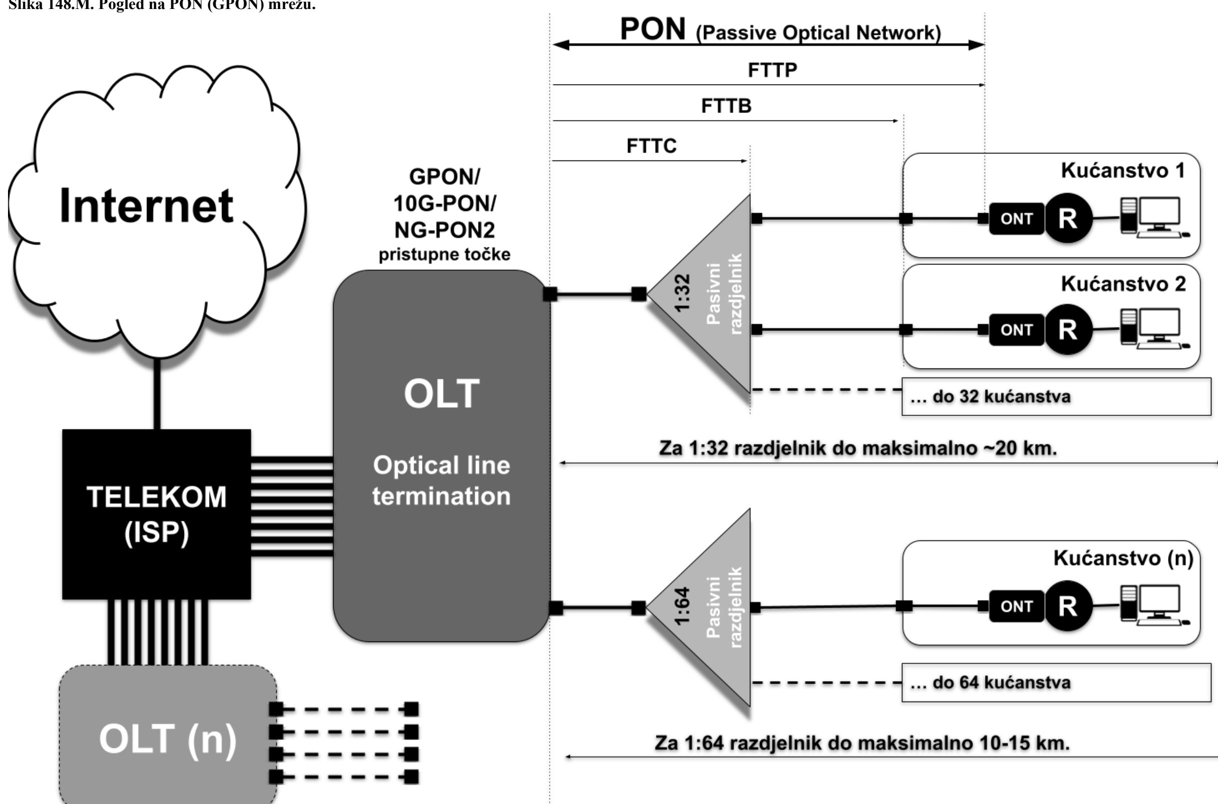
Ako gledamo s druge strane, svako pojedino kućanstvo ima svoje optičko vlakno (svjetlovod) koje dolazi do navedenog optičkog razdjelnika. Na strani korisnika (kućanstva), to se optičko vlakno povezuje s uređajem zvanim *Optical Network Terminal* (**ONT**) ili *Optical Network Unit* (**ONU**). Ovaj uređaj pretvara optički signal u električni signal koji može koristiti kućna oprema (preklopnici, računala, TV uređaji itd.). U novije vrijeme (2020.g.) ova (**ONT/ONU**) značajka nalazi se u kućnom usmjerivaču (*routeru*), koji dobivate od telekoma (ISP-a) od kojeg i koristite ***FTTH*** uslugu. Mada je moguća i varijanta u kojoj se optičko vlakno u kućanstvu prvo spaja u tzv. *Fiber media converter* odnosno uređaj koji konvertira optički signal u električni (uobičajeno *Ethernet*), koji se potom spaja na kućni usmjerivač (*router*).

Pogledajmo kako se ***multipleksira*** signal u *GPON* mreži:

- ***Downstream*** (od **OLT**-a prema korisnicima) – u smjeru prema korisnicima *OLT* šalje podatke kroz jedno vlakno prema optičkom razdjelniku (*splitteru*). Svi korisnici (kućanstva) koji su spojeni na taj razdjelnik primaju iste podatke, ali samo ***ONT*** uređaj (konkretno usmjerivač unutar kućanstva) kojem su podaci namijenjeni ih dekodira i koristi.
  - *Wavelength Division Multiplexing* (**WDM**) se koristi za dvosmjerni prijenos signala preko jednog vlakna, pri čemu *downstream* (*download*) signal uobičajeno koristi valnu duljinu od *1490 nm*.
- **Upstream** (od korisnika prema **OLT**-u [*i internetu*]) – u smjeru prema *OLT*-u, svi *ONT* uređaji šalju podatke koristeći *Time Division Multiple Access* (**TDMA**). To znači da svaki ONT uređaj ima svoj vremenski interval (*time slot*) kada može slati podatke, čime se izbjegava kolizija signala.
  - *Upstream* (*upload*) signal koristi drugu valnu duljinu, obično *1310 nm*, što omogućava da se *upstream* i *downstream* komunikacija odvijaju istovremeno.

Pogledajmo logičku shemu **PON** mreže na donjoj slici.

**Slika 148.M. Pogled na PON (GPON) mrežu.**




**Nove tehnologije: 10G-PON (XG-PON ili G.987)**

**10G-PON** (znan i kao XG-PON ili **G.987**) je nešto noviji od GPON standarda, a definiran je 2010. godine. Sposoban je isporučiti brzine pristupa internetu do 10 Gbps (gigabita u sekundi) preko postojeće optičke ***PON*** mreže i optičkih vlakana. I ovdje se koriste već navedene tehnike *multipleksiranja* kako bi se spriječilo međusobno ometanje signala i ubrzala komunikacija. Međutim postoje dvije varijacije ovog vršnog standarda:

- **Asimetrični 10G-PON** (tzv. **XG-PON1**) – on je propusnost od 10 Gbps je u dolaznom smjeru (*download*) dok je odlazna (*upload*) uobičajeno 2,5 Gbps. Za dolazni promet koristi valne duljine: *1575 – 1580nm,* a za odlazni: *1260-1280nm.*
- **Simetrični 10G-PON (XG-PON2)** – on je simetrične propusnosti od 10Gbps u oba smjera, ali zahtjeva lasere veće snage na strani **ONT** opreme (*upload*). Valne duljine su slične **XG-PON1**, ali je malo drugačiji način modulacije signala.

---

**Još novije tehnologije: NG-PON2**

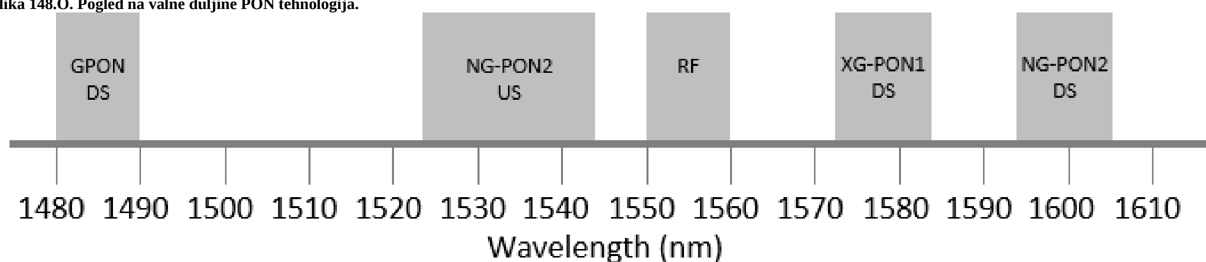
*NG-PON2* također poznat kao ***TWDM-PON*** je standard za *PON* optičke mreže sljedeće generacije iz 2015. godine. Standard je razvio **ITU** i detaljno opisuje arhitekturu sposobnu za ukupnu propusnost mreže do 40 Gbps. Međutim, on najčešće zahtijeva zamjenu i **OLT** i zamjenu **ONT** opreme, kroz postojeća optička vlakna.
Za dolazni promet koristi valne duljine: *1524 – 1544nm,* a za odlazni: *1596-1603nm.*

Od 2019. do 2021. godine objavljen je niz novih preporuka pod naslovom **Higher Speed PON** (serija standarda **G.9804 [HSP]**) namijenjenih nasljednicima **NG-PON2** za propusnosti do **50Gbps (50G-PON).**

**Kako radi TWDM PON**

*Time and Wavelength Division Multiplexing Passive Optical Network* je napredna tehnologija koja kombinira *Time Division Multiplexing* (**TDM**) i *Wavelength Division Multiplexing* (**WDM**) kako bi povećala kapacitet i fleksibilnost **PON** sustava.
Pri tome ***WDM*** omogućuje korištenje višestrukih valnih duljina za dolazni i odlazni promet, a unutar svake valne duljine koristi ***TDM*** kako bi se promet podijelio između različitih korisnika (kućanstava).
Pogledajmo i valne duljine navedenih **PON** tehnologija na slici **148.O.**:

**Slika 148.O. Pogled na valne duljine PON tehnologija.**

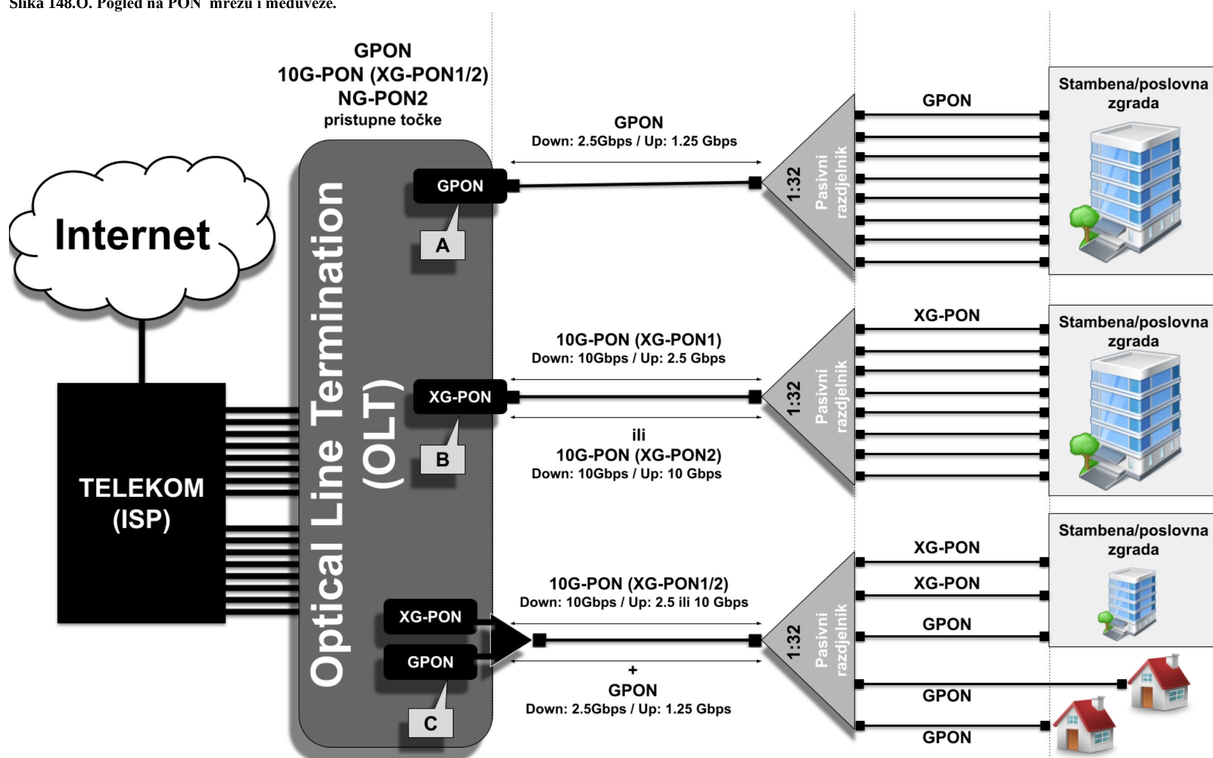



**Slika je preuzeta s Wikipedije: https://en.wikipedia.org/wiki/File:Optical_Network_Spectrum_including_NG-PON2_wavelengths.png**

Kako je vidljivo sa grafikona to jest slike **148.O.**, svaka od navedenih *PON* tehnologija koristi drugačiju valnu duljinu svjetlosti, ali i vrstu modulacije signala za komunikaciju. Upravo zahvaljujući različitim valnim duljinama i vrstama modulacije signala te tehnologijama poput *WDM*-a, različite **PON** tehnologije mogu koegzistirati i prolaziti kroz isto optičko vlakno. To omogućuje veću fleksibilnost i učinkovitost u upravljanju mrežnim resursima.
Kako je primjerice vidljivo na donjoj slici, na **OLT** uređajima je moguća upotreba:

- (A) – **GPON** modula na koji se standardno spaja jedno optičko vlakno koje vodi do TK ormarića blizu korisnika (pr. ulica ili naselje), u kojem se nalazi optički razdjelnik (*splitter*), koji optički signal dijeli (u ovom slučaju 1:32) na maksimalno 32 kućanstva. Pri tome svako kućanstvo dobiva GPON mrežu i teoretsku maksimalnu propusnost [*Down/UP*]: 2.5/1.25 Gbps. Ne zaboravimo da svi oni i dalje dijele istu maksimalnu propusnost do **OLT** uređaja.
- (B) – **XG-GPON** modula koji se spaja na isti način kao i GPON, ali koji podržava znatno veću ukupnu propusnost ([*Down/UP*]: 10/2.5 Gbps za **XG-PON1** ili 10/10 Gbps za **XG-PON2**). I ovdje svi krajnji korisnici i dalje dijele istu maksimalnu propusnost do **OLT** uređaja.
- (C) – kombinacija **XG-GPON/1/2** i primjerice **GPON** modula koji se spajaju na isti način kao i prethodno navedeni, ali je ovdje razlika u tome što je, zbog upotrebe različitih valnih duljina, moguće kombinirati GPON i XG-PON1/2 istovremeno. Naravno i ovdje svi krajnji korisnici i dalje dijele istu maksimalnu propusnost do **OLT** uređaja.

**Slika 148.O. Pogled na PON mrežu i međuveze.**




*Ono što je važno razumjeti jest da se navedene maksimalne propusnosti ("brzine") između **OLT** i razdjelnika (**splittera**) dijele među svim korisnicima (kućanstvima) koji su povezani na isti optički razdjelnik!*

Sve navedeno u konačnici znači da stvarno dostupne *brzine* za svakog pojedinog korisnika mogu varirati ovisno o ukupnom broju korisnika na optičkom razdjelniku i trenutnom opterećenju veze prema **OLT** uređaju. Naravno, i sama propusnost od **OLT** uređaja prema mreži telekoma (ISP-a) je također druga varijabla koja utječe na konačnu propusnost.

## QoS

Kvaliteta usluge (**QoS**) u **FTTH** mrežama je ključna za osiguranje pravilne distribucije mrežnih resursa, posebno u situacijama u kojima je potrebno prioritizirati određene vrste mrežnog prometa, kao što su glas, video i podaci. **QoS** se implementira prema različitim standardima i metodologijama koje su usklađene sa specifikacijama određene *FTTH* tehnologije.



Za detalje o klasičnom (*Ethernet*) ***QoS*** pogledajte poglavlje:
**23.1.1. Osiguranje kvalitete (*QoS*) i klasifikacija prometa (*ToS i DSCP*).**

### QoS i GPON

Primjerice za ***GPON***, **QoS** se najčešće temelji na ITU-T standardu **G.984**. Ovaj standard specificira različite tipove usluga i njihove prioritete, kao i mehanizme za upravljanje propusnošću i kašnjenjem. Konkretno je **ITU-T G.984.3** ključni standard koji definira mehanizme za *QoS* u GPON mrežama. On specificira kako se implementira takozvani **T-CONT** (*Transmission Container*), koji je osnovna jedinica za upravljanje propusnim opsegom i ***QoS*** u GPON mrežama. Dakle **T-CONT** se koristi za agregaciju (povezivanje) različitih tokova podataka iz različitih servisa (glas, video, internet) te omogućava dodjelu resursa prema njihovim ***QoS*** zahtjevima. Konkretno, GPON koristi sljedeće metode za upravljanje ***QoS****-om*:

- **OAM** (*Operations, Administration, and Maintenance*) – OAM značajke omogućuju nadzor i upravljanje kvalitetom usluge u mreži.
- **Dodjeljivanje propusnosti** (*Bandwidth Allocation*) – GPON koristi različite metode za dodjelu propusnosti korisnicima, uključujući dinamičku alokaciju propusnosti, gdje se resursi dodjeljuju na temelju trenutne potražnje i potreba korisnika.
- **Prioriteti prometa** (*Traffic Prioritization*) – različite vrste prometa (npr. video, glas, podaci) mogu se označiti s različitim prioritetima kako bi se osigurala optimalna izvedba i minimizirali utjecaji zagušenja. To konkretno znači da svaki *T-CONT* može biti konfiguriran sa različitim razinama prioriteta. Na primjer, glasovne usluge obično dobivaju viši prioritet u odnosu na podatkovne usluge kako bi se osigurala mala latencija (kašnjenje).
- *Dynamic Bandwidth Allocation* (**DBA**) – ovo je mehanizam u GPON-u koji omogućava dinamičku dodjelu propusnosti prema potrebama korisnika i trenutnim uvjetima u mreži. Naime **GPON OLT** (*Optical Line Terminal*) i **ONT** (*Optical Network Terminal*) odnosno vaš usmjerivač (*router*) mogu primijeniti mehanizme za oblikovanje mrežnog prometa kako bi se osiguralo da nijedan korisnik ne prekorači dodijeljeni opseg propusnosti.

*Novije generacije PON mreža koriste naprednije varijante DBA mehanizma:* ***NG-PON2 → ITU-T G.989****,*
***XG-PON → ITU-T G.987****,* ***XGS-PON → ITU-T G.9807.1****.*

### Kako radi T-CONT?

*T-CONT* možemo zamisliti kao virtualni prijenosni kontejner koji se koristi za prijenos podataka između *Optical Network Terminala* (**ONT**) kod korisnika i *Optical Line Terminala* (**OLT**) u centrali telekoma. Svaki *T-CONT* može prenositi podatke različitih servisa, kao što su klasični Internet promet, glas, video i druge usluge, a ima definirane **QoS** parametre.

### Kategorije T-CONT-a

Postoji pet različitih kategorija *T-CONT-a*, od kojih svaka pruža različitu razinu ***QoS-a***:

- **T-CONT tip 1** (*Fixed Bandwidth*) – ovaj *T-CONT* ima fiksnu količinu rezervirane propusnosti i koristi se za servise koji zahtijevaju stroge uvjete *QoS-a*, kao što su glasovni pozivi (*VoIP*).
- **T-CONT tip 2** (*Assured Bandwidth*) – ovaj *T-CONT* osigurava određenu količinu propusnosti, uz mogućnost korištenja dodatne propusnosti ako je dostupna. Često se koristi za video usluge.
- **T-CONT tip 3** (*Best Effort*) – ovaj *T-CONT* ne garantira propusnost, već koristi preostali kapacitet mreže. Primjenjuje se za servise gdje *QoS* nije kritičan, poput običnog surfanja internetom.
- **T-CONT tip 4** (*Non-Assured Bandwidth*) – on omogućuje prijenos podataka samo kad je kapacitet dostupan, bez ikakvih garancija za *QoS*. Može se koristiti za prijenos pozadinskih podataka koji nisu vremenski osjetljivi.
- **T-CONT tip 5** (*Hybrid*) – on je kombinacija više prethodnih *T-CONT-ova* , koji može podržavati više vrsta servisa unutar jednog *T-Conta*, ovisno o potrebama korisnika i mrežnim uvjetima.

### Kako radi QoS u GPON sustavu?

**1. Alokacija resursa** - kada korisnik pošalje zahtjev za prijenos podataka, **ONT** koristi odgovarajući *T-CONT* na temelju vrste servisa koji se koristi. *OLT* zatim alocira propusnost prema *QoS* zahtjevima definiranima za taj *T-CONT*. Naime kućni usmjerivač koji korisnik ima povezan na GPON mrežu obično upravlja s različitim vrstama prometa (npr. video streaming, VoIP, surfanje internetom). Ovaj promet se može povezati na različite *T-CONT* kategorije u GPON mreži, ovisno o njegovim *QoS* zahtjevima. Na primjer, VoIP (glasovni promet) će biti povezan na *T-CONT* s visokim prioritetom (npr. **T-CONT tip 1**), dok će surfanje internetom biti vezano na *T-CONT* s nižim prioritetom (npr. **T-CONT tip 3**).

**2. Upravljanje propusnošću** – s druge strane veze **OLT** koristi algoritme poput *Dynamic Bandwidth Allocation* (**DBA**) kako bi dinamički prilagodio propusnost svakom *T-CONTu* ovisno o trenutnoj opterećenosti mreže i potrebama korisnika. *DBA* je algoritam na *OLT-u* koji dinamički dodjeljuje propusnost korisnicima u skladu s trenutnim potrebama i dostupnim resursima mreže. DBA algoritam uzima u obzir vrstu prometa (kroz *T-CONT*) i osigurava da prioritetni promet dobije potrebnu propusnost. Na primjer, tijekom video konferencije, koja ima veće zahtjeve za latenciju i stabilnost, *DBA* će osigurati da taj promet dobije potrebnu širinu pojasa prije nego što se ona dodijeli manje kritičnim servisima. Kada primjerice korisnici imaju različite ugovorene brzine (npr. 1Gbps/500Mbps ili 500Mbps/500Mbps), **T-CONT** i **DBA** pomažu u upravljanju tim brzinama. Dakle *T-CONT* se koristi za grupiranje korisnika prema njihovim zahtjevima, dok *DBA* dinamički dodjeljuje propusnost unutar tih *T-CONT* kategorija prema trenutnim potrebama (i mogućnostima mreže).

**3. Prioritizacija** - *T-CONT* kategorije s većim *QoS* zahtjevima (npr. **T-CONT tip 1**) imaju prioritet nad *T-CONT* kategorijama koje koriste *Best Effort* pristup. Na taj se način osigurava minimalna latencija i potrebna propusnost kritičnih servisa.

## Druge slične tehnologije

## 10G-EPON

*10-Gigabit **Ethernet** Passive Optical Network* (**10G-EPON**) i *10-Gigabit Passive Optical Network* (**10G-PON**) su dvije tehnologije koje se koriste za brzi prijenos podataka preko optičkih mreža.
U rujnu 2009. *IEEE 802* je ratificirao amandman na 802.3 za objavljivanje dodatka **802.3av** kao standard **IEEE Std 802.3av-2009.** Nakon toga objavljen je **802.3bk** kao dopuna i odobrena je u kolovozu 2013. Ubrzo nakon toga objavljena je i kao standard **IEEE Std 802.3bk-2013**. U lipnju 2020. IEEE je odobrio IEEE **802.3ca**, koji omogućuje simetričan ili asimetričan rad s brzinama od 25 Gbps ili 50 Gbps (za *downlink*) i brzinama od 10 Gbps, 25 Gbps ili 50 Gbps (za *uplink*) preko iste optičke infrastrukture.
Kao i sve **E**PON mreže, **10G-EPON** prenosi podatke u paketima promjenjive duljine do 1518 bajta, kako je navedeno u standardu *IEEE 802.3* (***Ethernet***). Ovi paketi promjenjive duljine bolje su prilagođeni IP prometu nego paketi fiksne duljine od 53 bajta koje koriste druge pasivne optičke mreže, kao što je GPON. To može značajno smanjiti troškove *10G-EPON-a* u usporedbi s drugim sustavima. Tipični gubitak (*podaci-zaglavlja* paketa) u *10G-EPON* mrežama je otprilike 7,42%. Dok je isti tipični gubitak *podaci-zaglavlja* paketa (tzv. ***overhead***) za GPON mreže čak 13,22%. Ovaj visoki omjer gubitaka *podaci-zaglavlja* u konačnici omogućuje visoku iskoristivost sustava koji može koristiti jeftine (dostupne) optičke komponente.

**10G-EPON** i **10G-PON** se razlikuju u nekoliko ključnih aspekata:

- **10G-EPON** je standardiziran od strane IEEE (*Institute of Electrical and Electronics Engineers*) kao dio IEEE standarda **802.3av**.
- **10G-PON** je standardiziran od strane ITU-T (*International Telecommunication Union Telecommunication Standardization Sector)* kao dio **G.987** serije preporuka.

**Protokoli i arhitektura**
**10G-EPON** koristi ***Ethernet*** kao svoj osnovni protokol za prijenos podataka. Stoga je njegova arhitektura bazirana na ***Ethernet*** mrežnim okvirima, što omogućava jednostavnu integraciju s postojećim ***Ethernet*** mrežama.


Za detalje o ***Ethernet*** mrežama nastavite s proučavanjem ovog cijelog poglavlja o mrežama !

S druge strane ***10G-PON*** koristi takozvanu *GPON Encapsulation Method* (**GEM**) za ugnježđivanje (*enkapsulaciju*) podataka, što je specifično za *PON* tehnologije u *ITU-T* standardima. Ova arhitektura je bliža tradicionalnim telekomunikacijskim mrežama.

**Kapacitet i propusnost (*brzina*)**
**10G-EPON** nudi najčešće simetričnu brzinu prijenosa od 10 Gbps u oba smjera (*uplink* i *downlink*), mada može biti i asimetričan. **10G-PON** može biti simetričan (10/10 Gbps) ili asimetričan (10/2.5 Gbps), gdje je *downlink* brzina 10 Gbps, a *uplink* 2.5 Gbps.

**Kvaliteta usluge (QoS)**
**10G-PON** ima ugrađene **QoS** mehanizme unutar **GEM** protokola, omogućavajući detaljniju kontrolu i upravljanje prometom u mrežama. S druge strane **10G-EPON** koristi ***Ethernet*** **QoS** mehanizme, što omogućava naprednu kontrolu nad klasifikacijom i prioritetom prometa, ali može zahtijevati dodatne konfiguracije za optimizaciju u telekomunikacijskim mrežama.


Za detalje o klasičnom (*Ethernet*) ***QoS*** pogledajte poglavlje:
**23.1.1. Osiguranje kvalitete (*QoS*) i klasifikacija prometa (*ToS i DSCP*).**

**Optički spektar**

- **10G-EPON** koristi različite valne duljine za *uplink* (1270 nm) i *downlink* (1577 nm).
- **10G-PON** također koristi različite valne duljine, ali se razlikuju u modulaciji, dok su valne duljine obično 1577 nm za *downlink* i 1270 nm za *uplink*.

**Kompatibilnost i primjena**

- **10G-EPON** može biti kompatibilan s ranijim inačicama EPON-a (**1G-EPON**), omogućavajući koegzistenciju u istom vlaknu. ***10G-EPON*** je češće korišten u poslovnim mrežama, velikim podatkovnim centrima te od strane kabelskih operatera.
- **10G-PON** je također kompatibilan s GPON (2.5G/1.25G), omogućavajući postepeni prelazak na brže brzine. ***10G-PON*** je češće implementiran u telekomunikacijskim mrežama za pružanje širokopojasnog pristupa korisnicima.

**Izvori informacija (1726),(1727),(1728),(1729),(1730),(1731),(1732),(1733),(1737),(1738),(1739),(1740),(1743), ITU-T G.984, ITU-T G.987, ITU-T G.9804, ITU – T G.989.1, 802.3av.**

## 19.2.2. Mrežne kartice

U lokalnim mrežama za umrežavanje računala koristimo (*Ethernet*) mrežne kartice. Dakle mrežna kartica se koristi za mrežnu komunikaciju. Mrežna kartica poznata je i pod engleskim nazivima *LAN card* ili *Network interface controller (**NIC**)*. Ona se ugrađuje u računalo kao zasebna kartica preko *PCI* ili *PCI Express* sabirnice ili se već nalazi ugrađena na matičnu ploču računala, što je u današnje vrijeme standard.

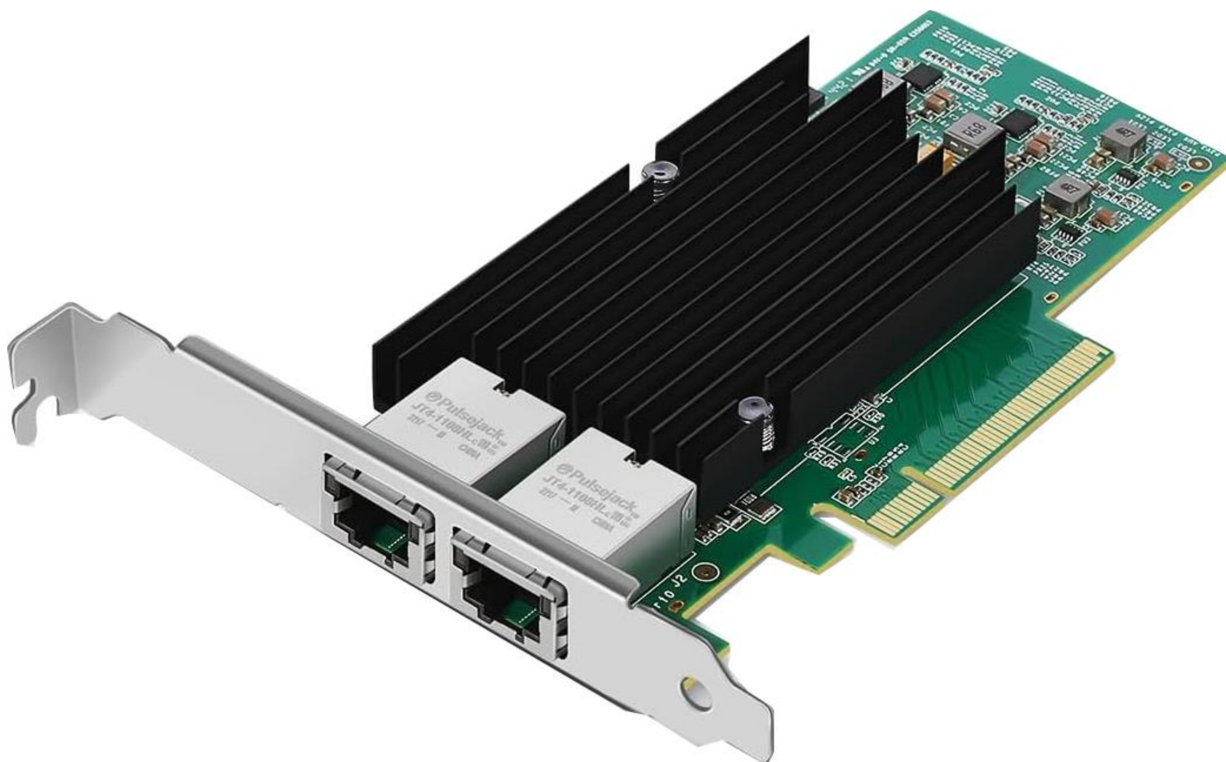
U grubo postoje dvije glavne kategorije mrežnih kartica: klasične i one za poslužitelje.

Slika 149. Pogled na mrežne kartice tvrtke Intel



Od običnih za klasična stolna računala, poput ove 1Gbps ***Intel 82574*** s PCI konektorom, na slici lijevo.

Do posebne kategorije mrežnih kartica koje možemo nazvati *poslužiteljskim*, poput ove 2x10Gbps **Intel X540** s *PCI express* (**PCIe**) **x8** konektorom, s dvije RJ45 utičnice, na slici dolje.





Kada govorimo o mrežnoj kartici mislim na hardversku karticu (sklopovlje), dok mrežno sučelje općenito, može označavati ili hardversku karticu ili virtualno (softversko) mrežno sučelje.

Važno je znati da niti *poslužiteljske* mreže kartice nisu sve iste, kao niti njihovi upravljački programi. Neke od njih možda nisu loše, ali imaju prilično loše upravljačke programe. Druge odrađuju sâmo standardne stvari dok neke od snažnijih podržavaju cijeli nîz dodatnih funkcionalnosti kojima obično rasterećuju centralni procesor (CPU). Neke od snažnijih *poslužiteljskih* mrežnih kartica danas standardno sâme odrađuju neke od sljedećih funkcionalnosti (o ovome detaljnije nešto kasnije):

- ***TCP*** *Offload* (*Checksum/Large send*) i **UDP**: za izračune provjernog zbroja (*checksum*) unutar TCP ili UDP paketa.
- **802.1Q** : za virtualne mreže (***VLAN***ove).
- **802.1p** (***QoS***): za protokole koji osiguravaju kvalitetu usluge.
- **802.3ad**, *Fast Ether Channel* i *Gigabit EC:* za *agregaciju* tj. udruživanje više mrežnih kartica u jednu (logičku).
- **802.3z**, **802.3ab**, **802.3u** i **802.3x** - *flow control:* za kontrolu protoka.
- Kriptiranje i dekriptiranje podataka.
- I/O Virtualizaciju u kombinaciji sa **SR-IOV** tehnologijom, …

Dakle moderne mrežne kartice imaju napredne mogućnosti kojima rasterećuju CPU odrađivanjem sve više zadataka bez njegovog posredovanja. Ako gledamo na komunikaciju mrežne kartice s ostatkom računala, ona u slučajevima kada je potrebno ili nešto slati na mrežu ili primiti s mreže, može funkcionirati na dva načina (ovisno o mogućnostima sâme kartice i njenog upravljačkog programa) odnosno kombinacijom obje metode, što ovisi o trenutnom stanju rada:

- Pomoću ***polling*** mehanizma, kod kojega CPU provjerava status mrežne kartice: provjerava je li nešto zaprimila.
- Pomoću signala prekida odnosno ***interrupt*** (IRQ), kada mrežna kartica za svaku i najmanju aktivnost (pr. svaki primljeni mrežni paket ili nekoliko njih) generira signal prekida, s kojim javlja procesoru (*CPUu)* da prekine sa svojim aktivnostima i posveti pažnju mrežnoj kartici. Nakon toga se događa tzv. *context switching*.

***Pooling i IRQ***

Mrežne kartice kao i drugi uređaji (pr. disk kontroleri) kada im je potrebna pažnja *CPUa* odnosno kada od *CPUa* traže procesorsko vrijeme za obradu njihovih zahtjeva, generiraju hardverski signal prekida (*Engl interrupt/IRQ*). U slučaju mrežne kartice, kod svakog novog mrežnog paketa koji je zaprimljen, mrežna kartica kreira signal prekida (*IRQ*). CPU tada zaustavlja trenutni proces (program) s obradom, odnosno pauzira ga, privremeno pohranjuje međurezultate, prazni memoriju i registre te preuzima na obradu zadatak od mrežne kartice. Točnije preuzima na obradu taj konkretni mrežni paket. Za gigabitne brzine (1Gbps) to znači 1.48 milijuna paketa u sekundi, koje mrežna kartica može primiti ili poslati za 64 bajtne pakete. Jasno je kako se broj signala prekida (*IRQ*) kod ovakvog načina rada također mora popeti na isti broj: 1 paket = 1 signal prekida (*interrupt*).

Mehanizmi implementirani u *Linux*, *FreeBSD* i druge moderne operativne sustave u slučajevima kada broj mrežnih paketa i sâmim time signala prekida (**IRQ**) postaje prevelik da bi zagušio cijeli operativni sustav i sve programe, prelazi u drugačiji način rada. Ovaj novi način rada ovisi o tome podržava li ga uopće upravljački program mrežne kartice kao i sâm hardver kartice.

Dakle u tom trenutku se prelazi na rad u kojem se više ne generiraju signali prekida za svaki pojedini mrežni paket, već se sustav počinje brinuti o tome kako bi svako malo s posebnim mehanizmima sâm dohvaćao pakete od mrežne kartice (*Engl. Polling mehanizam*). Na ovaj način sâm sustav se počinje brinuti i o tome kako će opsluživati ostale programe (procese) na najnižoj razini. Kod boljih implementacija ovdje se pazi na tzv. ***context switching*** na najbolji mogući način i time se dodatno dobiva na performansama jer se CPU (pomoću *task/proces schedulera*) prebacuje između procesa na način koji je optimalan i kada je to stvarno nužno. U konačnici:

- Izbjeglo se zagušenje cijelog sustava zbog generiranja prevelikog broja signala prekida (***IRQ***).
- Paketi koji se povlače (*poll*) s mrežne kartice na obradu, mogu se dohvaćati u nîzu (više njih odjednom).
- Kod najgorih slučajeva, kada se tolika količina paketa ne može obraditi niti ovom metodom, sustav ih odmah odbacuje na sâmoj mrežnoj kartici, bez nepotrebnog dodatnog procesiranja od strane *mrežnog stôga* odnosno dijela kernela operativnog sustava zaduženog za mrežu.

Osim toga, sâm prijenos podataka od mrežne kartice prema CPU može se odraditi na dva načina:

- Pomoću „*programmed input/output*“ metode u kojoj se CPU brine za svaki prijenos podataka od i prema mrežnoj kartici. Ovo naravno nije najbolja metoda jer se opterećuje CPU za svaku aktivnost mrežne kartice.
- Pomoću Direct memory access (**DMA**) metode u kojoj mrežna kartica može preko **DMA** kanala pristupati direktno RAM memoriji bez posredovanja *CPUa*. Ovo je danas standardna metoda komunikacije, ali zahtjeva dodatnu logiku odnosno hardversku podršku unutar mrežne kartice.

*Svaka mrežna kartica, od strane proizvođača ima zapisanu (svoju) jedinstvenu identifikacijsku adresu, takozvanu **MAC** adresu. **MAC** adresa mrežne kartice koristi se u mrežnoj komunikaciji, kao jedan od identifikatora veze.*



Vezano za **MAC** adrese pogledajte poglavlje:
**19.3.1. Što su MAC adrese.**

### 19.2.2.1. Logička shema mrežne kartice

Prvo pogledajmo logičku shemu dijelova i načina rada mrežne kartice (fizičkog mrežnog sučelja), kako je vidljivo na slici 150.

**Slika 150. Logička shema rada mrežne kartice.**



Gledano od dna sheme na slici 150, vidljivo je kako je ***PHY*** dio zadužen za pretvaranje svega onoga što je došlo s mreže; električni signali ili svjetlosni impulsi, pretvaraju se u mrežne okvire i obrnuto.

***PHY*** radi na OSI sloju jedan (OSI 1). *PHY* funkcionalnost se često naziva i ***MDI*** (*Media Dependent Interface*) jer njegov rad ovisi o vrsti medija za koji je napravljen:

- *Za bakar* su to električni signali i načini modulacije te slanja i primanja signala.
- *Za optiku*: ovisno za koje vlakno (*single*/*multi* mode) te koje valne duljine i slično.

Sljedeća komponenta u komunikaciji unutar mrežne kartice je **MII** *čip*, koji je zadužen za komunikaciju između *PHY* i *MAC* dijela. Dakle ***MII*** (*Media Independent Interface*) mora znati komunicirati s ***PHY*** od kojega mora moći zatražiti postavljanje parametara rada, ali i moći ih pročitati odnosno provjeriti njihov trenutni status. Preko ***MII*** se odrađuje konfiguracija mrežnog sučelja (pr. brzina i *duplex*) kao i eventualne postavke poput *auto negotiation* protokola i njegovog rada te provjerava status istih parametara. Iz razine operativnog sustava najniže što se spuštamo kod korištenja i upotrebe mrežne kartice je upravo do razine **MII** sklopa, što je i logično jer nas niti ne treba zanimati kako će se točno kreirati i slati ili primati signali na mrežu odnosno mrežni medij. Tako u operativnom sustavu *Linux* postoje programi `mii-tool` kao i `ethtool` s kojima je moguće direktno pristupiti ***MII** sklopu* u svrhu promjene ili čitanja parametara rada mrežne kartice (OSI slojevi **1** i **2**). Naravno i druge softverske komponente mu mogu pristupiti. Nadalje se ***MII*** spaja na ***MAC sklop*** *(čip)*, koji odrađuje sve što se i prema logici događa na OSI sloju dva (OSI 2) kojemu on i pripada. Dodatno ***MAC** sklop* često dodaje i **SFD** (*Start frame delimiter*) i ***IFG*** (*Inter frame Gap*) polja koja prema logici pripadaju OSI sloju jedan (OSI 1). Dakle ovdje se radi s mrežnim okvirima i njegovim logičkim cjelinama i dijelovima: od provjere ispravnosti mrežnih okvira, preračunavanjem **FCS** provjernog zbroja (engl. *checksum*) kao i umetanjem preambule (***SFD***) i ***IFG*** polja.


Za detalje pogledajte poglavlje: **20.1. Oblik mrežnih okvira na OSI slojevima 2 i 1.**

Komponenta koja slijedi je ***Network Layer*** odnosno mrežni sloj koja odrađuje sve što je potrebno na OSI sloju tri (*IP* sloj prema *TCP/IP* protokolu). Sve ostale akcelerirane odnosno ubrzane funkcije na ovom sloju, ako ih sâma kartica podržava, odrađuju se ovdje. Nadalje se sve prosljeđuje na komponentu koja radi na OSI sloju četiri (OSI 4), koja je zadužena za *TCP* ili *UDP* protokol te također, ako je nešto od toga hardverski ubrzano, odrađuje se ovdje.
Potom sve ide prema vršnim komponentama kojih može biti i manje, pa se sve prepušta dalje na softversku obradu. Odnosno moguće je da ovdje postoji i više komponenti unutra naprednih mrežnih kartica. Potom se sve šalje na sabirnicu (*PCI* ili *PCI Express*) te prema procesoru (CPU) na obradu. I konačno na kraju sve dolazi do aplikacija koje su krajnje točke u komunikaciji između računala koja komuniciraju. Vrlo često su sve gore navedene komponente integrirane u jedan jedini sklop (*čip)*, barem na većini mrežnih kartica.



***SFP*** *moduli sadrže* ***PHY*** *sklop, dok se kod klasičnih mrežnih kartica on nalazi integriran na sâmoj mrežnoj kartici.*

**I u konačnici u čemu su sve razlike između mrežnih kartica:**

- U cijeni i brzini.
- Podržanim funkcionalnostima.
- Upravljačkim programima (*driverima*) te njihovoj stabilnosti i brzini (loš *upravljački program* = loša mreža).
- Te podršci i dokumentaciji.

Sada nam postaje jasno kako uopće nije nevažno koju mrežnu karticu treba odabrati, ako želimo siguran, pouzdan i brz rad računala ili poslužitelja na mreži. Isto vrijedi i kod odabira proizvođača koji izrađuju preklopnike. Razlike u praksi znaju biti drastične. Ako govorimo o mrežnim karticama za računala, postoje razlike u cijeni od nekoliko desetaka ili stotina kuna, a nekada i tisuća KN, te razlike u stabilnosti odnosno nestabilnosti te do varijacija u brzini rada. Vrlo se često događa da ste kupili sve mrežne komponente, kao i pasivni dio mreže (kablovi, utičnice, *patch paneli*) koje prema standardima podržavaju brzinu deklariranu kao primjerice **1Gbps**, ali u praksi vam je brzina jedva nešto veća od **500Mbps**.

## 19.2.2.2. Small Form-factor (SFF) i Small Form-factor Pluggable (SFP)

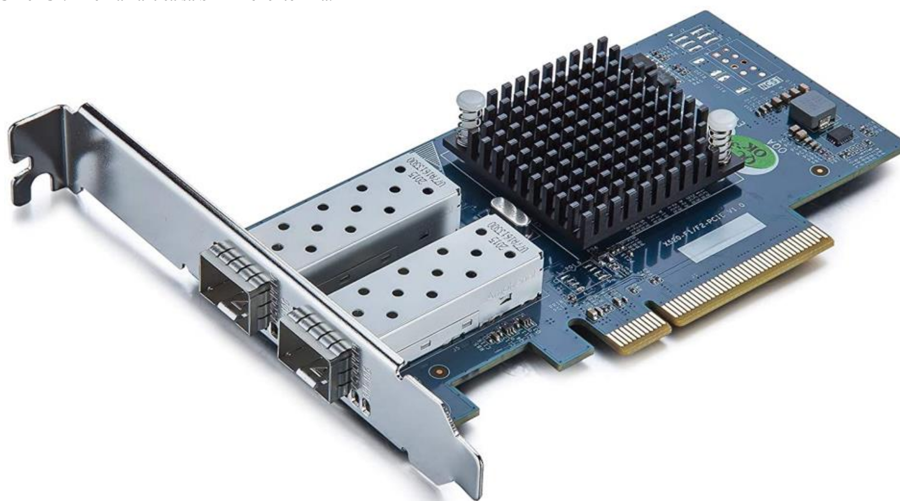
Mrežne kartice obično na sebi imaju utičnicu **8P8C**, koju nazivamo **RJ-45** i koja za umrežavanje koristi bakrene parice. Mana bakrenih parica je njihova maksimalna propusnost; pr. najveća **Cat 8** kategorija kabela ima maksimalnu propusnost **40Gbps**. Međutim osim ovog standardnog **RJ-45** konektora, izrađuju se i mrežne kartice s modularnim utorima poput: **SFP** i **SFP+ (**poput kartice tvrtke ***Qlogic***, na slici 151.**)** te **XFP**, kao i **SFP28**, **QSFP**, **QSFP+**, **QSFP14**, **QSFP28** te **QSFP56** utora.

**SFP** utori se koriste i na preklopnicima, usmjerivačima, vatrozidima i drugoj mrežnoj opremi, koja ih intenzivno koristi, kao i RJ-45 utore, a pogotovo za veće brzine (**10+Gbps**), za koje su neke od varijanti **SFP** utora jedini mogući izbor.

Naime sve naveden varijante ***SFP*** sučelja su zapravo utičnice u koje se mogu spojiti moduli raznih brzina, ali i različitih vrsta vanjskih utičnica. Razne ***SFP*** varijante modula se koriste redom, ovisno o brzini sâmog sučelja: **SFP** (do 1Gbps), **SFP+** (do 10Gbps), **SFP28** (do 25Gbps), **QSFP+** (do 40Gbps), **QSFP14** (do 50Gbps), **QSFP28/SFP-DD/SFP112** (do 100Gbps) te **QSFP56/SFP-DD112** (do 200Gbps), **QSFP112/QSFP-DD** (do 400 Gbps), ….

Općenito, ***SFP*** (*Small Form-factor Pluggable*) moduli se nazivaju primopredajnicima odnosno *transceiverima* jer se u njima nalazi sva potrebna elektronika za pretvaranje signala između mrežne kartice i vanjskog utora *SFP* modula.

**Slika 151. Mrežna kartica sa SFP+ konektorima.**



Dakle ***SFP*** primopredajnici s vanjske strane imaju utičnicu (konektor) s kojom ga spajamo na mrežu, a ona može biti:

- ***RJ-45*** (bakar).
- ***SFP*** + ***Twinax (DAC)***
- ***SFP*** optika s ***LC*** utičnicom.
- ili neki drugi (**SFP)** utor.

Ali osim sâmog konektora, pogotovo za konektore za optička vlakna, ovdje je stvar odabira za kakvu modulaciju i snagu signala, te za koju vrstu optičkog vlakna je namijenjen konkretni **SFP** primopredajnik.

Pitanje odabira **SFP** primopredajnika najčešće se odnosi na vrstu optičkog vlakna na koje ćemo ga spojiti, a koje može biti:

- *Jednomodno (engl. single mode).*
- *Višemodno (engl. multi mode*).

***Osim optičkih SPF modula, koristi se i TWinax DAC*** (*Direct Attach Cable*)***, ali za vrlo male udaljenosti (do 15.m.)***

Alat odnosno naredba `ethtool` na Linuxu omogućuje pregledavanje detaljnih informacija o mrežnim sučeljima, uključujući informacije o **SFP** (*Small Form-factor Pluggable*) modulima. Da biste vidjeli koji SFP modul se koristi na željenom mrežnom sučelju, možete koristiti naredbu `ethtool -m` ili `ethtool --module-info`.
Pogledajmo primjer za eth0 mrežno sučelje na kojem imamo 25Gbps **SFP28** modul (**25GBASE-SR** [*za multimode optiku*]):

```
ethtool -m eth0

Identifier                                  : 0x03 (SFP)
Extended identifier                         : 0x0d (SFP+ / SFP28)
Connector                                   : 0x07 (LC)
Transceiver codes                           : 0x40 0x00 0x00 0x00 0x00
Transceiver type                            : 25G Ethernet: 25GBASE-SR
Encoding                                    : 0x06 (64B/66B)
BR, Nominal                                 : 25781MBd
Rate identifier                             : 0x00
Length (SMF,km)                             : 0km
Length (SMF)                                : 0m
Length (50um)                               : 70m
Length (62.5um)                             : 0m
Length (Copper)                             : 0m
Length (OM3)                                : 100m
Laser wavelength                            : 850nm
. . .
```

Korisne informacije nam može dati i ispis naredbe `dmidecode`:

```
dmidecode -t connector
```

**Vratimo se na malo starije SFP** i **SFP+ module koji rade na brzinama do 10Gbps**
***SFP*** moduli su namijenjeni za brzine do 1.000 Mbps (1Gbps), a **SFP+** moduli su za brzine do 10.000 Mbps (10 Gbps).
Pogledajmo i dva: **SFP** i **SFP+** modula. Slika 152 prikazuje optički **SFP+** modul (s **LC** optičkim konektorom) tvrtke **HP** (model J9150A). Dok slika 153 prikazuje **Cisco 1000Base-T** 1Gbps s **RJ-45** konektorom.

**Slika 152. SFP+ optički modul .**




**Slika 153. SFP modul: SFP – RJ-45.**



Bez obzira koristili mrežne kartice s **RJ-45** ili raznim varijantama *Small Form-factor Pluggable* (***SFP)*** modula, propusnosti ovih modula koji su danas u upotrebi su od 100Mbps do 800Gbps.
Pogledajmo tablicu sa *Small Form-factor Pluggable* ***(SFP)*** vrstama modula te njihovim maksimalnim pripadajućim propusnostima i načinima njihovog međusobnog povezivanja:

| Modul | Maksimalna propusnost | Vrsta veze | | Modul | Maksimalna propusnost | Vrsta veze |
|---|---|---|---|---|---|---|
| **SFP** | **100 Mbps – 1 Gbps** | **Parica ili Optičko vlakno** | | **QSFP+** | **40 Gbps** | **Optičko vlakno ili DAC*** |
| **SFP+** | **10 Gbps** | **Parica ili Optičko vlakno ili DAC*** | | **QSFP28** | **50 Gbps** | **Optičko vlakno ili DAC*** |
| **SFP28** | **25 Gbps** | **Optičko vlakno ili DAC*** | | **QSFP28** | **100 Gbps** | **Optičko vlakno ili DAC*** |
| **SFP56** | **50 Gbps** | **Optičko vlakno ili DAC*** | | **QSFP56** | **200 Gbps** | **Optičko vlakno ili DAC*** |
| **SFP-DD SFP112** | **100 Gbps** | **Optičko vlakno ili DAC*** | | **QSFP112** | **400 Gbps** | **Optičko vlakno ili DAC*** |
| **SFP-DD 112** | **200 Gbps** | **Optičko vlakno ili DAC*** | | **QSFP-DD** | **400 Gbps** | **Optičko vlakno ili DAC*** |

U novije vrijeme pojavio se takozvani ***small form-factor pluggable double density*** (**SFP-DD**) standard, koji je udvostručio propusnosti, pa smo tako, osim navedenih **SFP-DD**, **SFP-DD112** i **QSFP-DD**, dobili i **QSFP-DD-800**, za **800 Gbps**.

***DAC-** ***Direct attach cable*** je izvedba **SFP** modula s tvornički ugrađenim ***Twinax*** kabelom.

## Oznake SFP modula

Svaki *Small Form-factor Pluggable* (**SFP**) modul osim brzine rada može biti namijenjen za drugi medij: dok neki mogu biti za bakar (**RJ-45** ili **DAC**), drugi mogu biti za spajanje na optička vlakna (*single mode* ili *multimode*).
Osim toga, što se tiče modula za optiku, **SFP** moduli odnosno njihovi primopredajnici mogu biti različite snage: neki koriste led diode, a neki drugi razne vrste lasera različitih izlaznih snaga, sve ovisno o tome za koju udaljenost i vrstu optičkog vlakna su namijenjeni.

*SFP* moduli se uobičajeno označavaju na slijedeći način:

## 1000BASE-T

Pri tome:

- **1000** – označava gigabitni *Ethernet* (1000Mbps = 1Gbps).
- **BASE** – označava da se radi o baznom (*baseband*) prijenosu podataka. *Baseband* prijenos koristi cijeli kapacitet medija za prijenos jednog signala. Suprotno od toga je *broadband*, gdje se više signala može prenijeti simultano na različitim frekvencijama (BASE se koristi u većini primjena).
- **T** – označava (*Twisted Pair*) odnosno upotrebu bakrenih upletenih parica. npr., 10GBASE-T koristi bakrene kabele (CAT6a) s konektorima RJ-45.

Pogledajmo i sve druge oznake odnosno zadnje slovo u oznaci:

- **CR**: (*Copper*) - Označava upotrebu bakrenih ***Direct Attach kabela*** (**DAC**). Tipično se koristi za vrlo kratke domete (npr. 1, 2, 3, 4, 5 ili 7 metara).
- **T**: (*Twisted Pair*) – označava upotrebu bakrenih upletenih parica. npr., 10GBASE-T koristi bakrene kabele (CAT6a).
- **SR**: (*Short Range*) – označava kratki domet prijenosa preko *multimode* optičkih kabela. Tipično se koristi za domete do 100 metara.
- **LR**: (*Long Range*) – označava dugi domet prijenosa preko *singlemode* optičkih kabela. Tipično se koristi za domete do 10 kilometara. Primjerice to su SFP moduli poput: **10GBASE-LR**, **25GBASE-LR**, …
- **ER**: (*Extended Range*) – označava produženi domet prijenosa preko *singlemode* optičkih kabela. Tipično se koristi za domete do 40 kilometara.
- **ER4**: (*Extended Range with 4 Lanes*) – označava produženi domet prijenosa s 4 paralelne trake (lijeva i desna). Tipično se koristi za *singlemode* optičke kabele.
- **LX**: (*Long Wavelength*) – označava dugi domet prijenosa preko *singlemode* optičkih kabela, često koristi valnu duljinu od 1310 nm.
- **SX**: (*Short Wavelength*) – označava kratki domet prijenosa preko *multimode* optičkih kabela, često koristi valnu duljinu od 850 nm.
- **DR**: (*Data Rate*) – označava prijenose s velikom propusnošću podataka na srednjim dometima (npr. do 500 metara preko *singlemode* optičkih kabela).
- **FR**: (*Forward Error Correction*) - Označava prijenose s korekcijom pogrešaka naprijed, obično za domete do 2 kilometra preko singlemode optičkih kabela.
- **SR4**: (*Short Range with 4 Lanes*) – označava kratki domet prijenosa s 4 paralelne trake (lijeva i desna). Tipično se koristi za *multimode* optičke kabele.
- **LR4**: (*Long Range with 4 Lanes*) – označava dugi domet prijenosa s 4 paralelne trake (lijeva i desna). Tipično se koristi za *singlemode* optičke kabele.
- **LR8**: (*Long Range with 8 Lanes*) – označava dugi domet prijenosa s 8 paralelnih traka (lijeva i desna). Tipično se koristi za *singlemode* optičke kabele.
- **SR16**: (*Short Range with 16 Lanes*) – označava kratki domet prijenosa sa 16 paralelnih traka (lijeva i desna). Tipično se koristi za *multimode* optičke kabele.

Postoje i još neki specifični standardi za takozvani ***Backplane*** *Ethernet*. *Backplane Ethernet* se odnosi na električnu ploču koja povezuje različite module ili komponente unutar mrežnog uređaja ili poslužitelja. On pruža putove za komunikaciju između različitih modula bez potrebe za vanjskim kablovima, a koristi se primjerice unutar tzv. ***Blade poslužitelja***.

- **KX**: (*Copper*) – pr. **1000BASE-KX** – standard za 1 Gbps *Ethernet* preko *backplane veze* , koristi se za povezivanje unutar mrežnih uređaja ili poslužitelja.
- **KR:** (*Copper*) – pr. **10000GBASE-KR** – standard za 10 Gbps *Ethernet* preko *backplane veze* , koristi se za povezivanje unutar mrežnih uređaja ili poslužitelja.
- **KR4:** (*Copper*) – pr. **40GBASE-KR4** – standard za 40 Gbps *Ethernet* preko *backplane veze ili* **100GBASE-KR4** (100 Gbps), koristi se za povezivanje unutar mrežnih uređaja ili poslužitelja.
- **KP4:** (*Copper*) – pr. **100GBASE-KP4** – standard za 100 Gbps *Ethernet* preko *backplane veze* , koristi se za povezivanje unutar mrežnih uređaja ili poslužitelja.



Vezano za optiku, modulaciju signala i *modulatore* (izvore svjetlosnih signala), pogledajte poglavlja:
**19.2.1.2. Optika.**
**19.2.1.2.2. Fiber to the Home (FTTH).**

Pogledajmo skraćenu tablicu s raznim vrstama **SFP** modula i njihovim značajkama:

| Naziv Modula | Brzina | Medij | Maksimalna Udaljenost | Tip Modulatora (izvor svjetlosnog signala) |
|---|---|---|---|---|
| 1000BASE-T | 1 Gbps | *Twisted Pair (UTP)* | 100 m | *- Bakar -* |
| 1000BASE-LX | 1 Gbps | *Single-mode fiber (SMF)* | 10 km | *Fabry-Perot Laser* |
| 10GBASE-T | 10 Gbps | *Twisted Pair (UTP)* | 100 m | *- Bakar -* |
| 10GBASE-SR | 10 Gbps | *Multimode fiber (MMF)* | 300 m | VCSEL |
| 10GBASE-LR | 10 Gbps | *Single-mode fiber (SMF)* | 10 km | DFB Laser |
| 10GBASE-ER | 10 Gbps | *Single-mode fiber (SMF)* | 40 km | DFB Laser |
| 10GBASE-ZR | 10 Gbps | *Single-mode fiber (SMF)* | 80 km | DFB Laser |
| 25GBASE-SR | 25 Gbps | *Multimode fiber (MMF)* | 100 m | VCSEL |
| 25GBASE-LR | 25 Gbps | *Single-mode fiber (SMF)* | 10 km | DFB Laser |
| 40GBASE-SR4 | 40 Gbps | *Multimode fiber (MMF)* | 100-150 m | VCSEL (4x) |
| 40GBASE-LR4 | 40 Gbps | *Single-mode fiber (SMF)* | 10 km | DFB Laser (4x) |
| 100GBASE-SR4 | 100 Gbps | *Multimode fiber (MMF)* | 70-100 m | VCSEL (4x) |
| 100GBASE-LR4 | 100 Gbps | *Single-mode fiber (SMF)* | 10 km | DFB Laser (4x) |
| 100GBASE-ER4 | 100 Gbps | *Single-mode fiber (SMF)* | 40 km | DFB Laser (4x) |
| 100GBASE-ZR4 | 100 Gbps | *Single-mode fiber (SMF)* | 80 km | DFB Laser (4x) |
| 200GBASE-SR4 | 200 Gbps | *Multimode fiber (MMF)* | 70-100 m | VCSEL (4x) |
| 200GBASE-LR4 | 200 Gbps | *Single-mode fiber (SMF)* | 10 km | DFB Laser (4x) |
| 400GBASE-SR16 | 400 Gbps | *Multimode fiber (MMF)* | 100 m | VCSEL (16x) |
| 400GBASE-LR8 | 400 Gbps | *Single-mode fiber (SMF)* | 10 km | DFB Laser (8x) |
| 400GBASE-ER8 | 400 Gbps | *Single-mode fiber (SMF)* | 40 km | DFB Laser (8x) |
| DWDM SFP+ | 10 Gbps | *Single-mode fiber (SMF)* | 100-200 km | DFB Laser |
| DWDM QSFP28 | 100 Gbps | *Single-mode fiber (SMF)* | 80-200 km | DFB Laser |



Za *Small Form-factor Pluggable* (**SFP**) odnosno za *Small Form Factor* (**SFF**) zadužene su razne organizacije, među kojima su: ***SNIA SFF Technology Affiliate***, ***MSA** (Multi-Source Agreement) Groups* i ***IEEE** (Institute of Electrical and Electronics Engineers).*

**Izvori informacija: (44),(1013),(K-3),(K-6),(K-9),(K-17), `man ethtool`,`man dmidecode`.**

### 19.2.2.3. MDI i MDI-X

**MDI** (*Medium Dependent Interface*) opisuje specifično mrežno sučelje (bakar ili optika) na fizičkom sloju komunikacije.
Isto tako standardima je definirana i mogućnost sučelja zvanog **MDI-X** (*Medium Dependent interface Crossover*).
Ova mogućnost odnosno sposobnost određenog mrežnog sučelja se odnosi na unutarnju shemu spajanja odnosno rasporeda nožica/vodova. Tako su mrežna sučelja (*interfacei/portovi)* na primjerice preklopniku (*switchu)* označeni s *MDI-X* zapravo spojeni kao da imaju „*crossover*" vezu unutar sebe tj. zamijenjene su im *nožice* za primanje (RX) i slanje (TX).
Tako da bi spajanjem normalnog odnosno prespojnog (*patch)* kabela na njima zapravo dobili *ukrižani* (*engl. crossover)* spoj; kao da smo koristili *crossover* kabel. U pravilu mrežne kartice na računalima i usmjerivačima (*routerima*) interno koriste **MDI** (dakle normalan raspored vodova), dok su mrežne kartice/portovi na koncentratorima (*HUBovima)* i preklopnicima (*switchevima*) izvedene u **MDI-X** varijanti. Možemo to gledati i na način kako se spajaju uređaji:

- **Računalo** (*MDI*) → običan kabel → **Preklopnik** (*MDI-X*).
- **Računalo** 1 (*MDI*) → *crossover* (*ukrižani*) kabel → **Računalo** 2 (*MDI*).
- ***Preklopnik*** *1* (*MDI-X*) → *crossover* (*ukrižani*) kabel → ***Preklopnik*** *2* (*MDI-X*).

**Auto MDI-X**
Na svim novijim mrežnim uređajima ozbiljnijih proizvođača koristi se i noviji standard koji se zove **Auto MDI-X**. Ovdje se radi o sposobnosti svakog mrežnog sučelja (*porta)* na uređaju da prvo prepozna način spoja druge strane te se automatski prilagodi odnosno promjeni svoje interno kabliranje. Drugim riječima ovakvi uređaji nakon (ispravne) detekcije druge strane, sâmi sebe prebacuju iz ***MDI*** u ***MDI-X*** način unutarnjeg kabliranja.
Ova detekcija je prilično brza (oko 500 ms), ali je u algoritam prepoznavanja dodan dodatni sigurnosni brojač, tako da se vrijeme detekcije i prebacivanja može povećati do 1.5 sekunde. Naravno i ovu funkcionalnost baš nisu svi proizvođači implementirali najbolje pa ona stoga kod većine njih radi kako treba, ali kod nekih baš i ne. Stoga je potreban oprez tijekom (pravilnog) odabira vrste mrežnih kablova za povezivanje.

**Izvor informacija: (1014).**

### 19.2.3. Brzina mreže

Kod onoga što nazivamo *brzina* mreže, potrebno je razlikovati širinu komunikacijskog pojasa odnosno takozvani *bandwidth* od propusnosti mreže odnosno takozvanog *throughput*-a.

#### 19.2.3.1. Širina komunikacijskog pojasa (*bandwidth*)

Širina komunikacijskog pojasa odnosno ***bandwidth*** se definira kao količina informacija koje se mogu prenijeti određenim mrežnim medijem odnosno u jednom komunikacijskom kanalu. Pojam širina pojasa definira vršnu brzinu prijenosa informacija, odnosno korisnu brzinu prijenosa na fizičkom sloju gledano kroz mrežni medij, kapacitet kanala ili maksimalnu *propusnost* logičke ili fizičke komunikacijske veze u digitalnom komunikacijskom sustavu. To znači da što je veća širina komunikacijskog pojasa, odnosno kanala, više informacija se može njome prenijeti. Pri tome mrežni medij može biti:

- Žičani kabeli, obično parični [*FTP*, *STP*, *SFTP* ili *UTP*] ili koaksijalni kabeli [RG*] .
- Optički kabeli: *single* ili *multi mode*.
- Ili za bežične mreže (eter): zrak, vakuum..

Za svaki medij postoje ograničenja komunikacijskog pojasa. Odnosno uvijek postoje razlike između širine komunikacijskog pojasa (*bandwidtha)* i efektivne propusnosti (*throughputa)* i to zbog raznih gubitaka: što u sâmom mediju jer dolazi do slabljenja ili miješanja signala, a što zbog dodataka na "korisne" podatke, koje dodaju sâmi mrežni protokoli.
Jedinica za širinu komunikacijskog pojasa je ***bps*** (*bits per second*) odnosno bitova u sekundi, kako je vidljivo u tablici:

| **Jedinica** | **Oznaka** | **Vrijednost** |
|---|---|---|
| Bitova u sekundi | ***bps*** | 1bps = Osnovna jedinica |
| Kilobita u sekundi | ***kbps*** | 1kbps=1.000 bps |
| Megabita u sekundi | ***Mbps*** | 1Mbps=1.000.000 bps |
| Gigabita u sekundi | ***Gbps*** | 1Gbps=1.000.000.000 bps |
| Terabita u sekundi | ***Tbps*** | 1Tbps=1.000.000.000.000 bps |

**Izvor informacija: (757),(K-10).**

#### 19.2.3.2. *Propusnost* (*throughput*)

Mrežna propusnost (engl. *throughput*) odnosi se na brzinu isporuke poruka preko komunikacijskog kanala, kao što su *Ethernet* ili *paketni radio*, u komunikacijskoj mreži. Podaci koje te poruke sadrže mogu se isporučivati preko fizičkih ili logičkih veza ili mrežnih čvorova. Propusnost se obično mjeri u bitovima u sekundi (**bit/s** ili **bps**), a ponekad i u paketima podataka po sekundi (**p/s** ili **pps**). Dakle propusnost *odnosno* ***throughput*** označava uspješno prenesenu količinu informacija u jedinici vremena.
Radi se o tome da u prijenosu podatak kroz mrežu dolazi do gubitaka i usporavanja, uvjetovanih mrežnom opremom ili kvalitetom veza i drugim čimbenicima. **Maksimalnu teoretsku propusnost** mreže izraženu u **`Bps`** možemo izračunati tako da maksimalnu teoretsku propusnost podijelimo s osam (8), jer bitove u sekundi želimo pretvoriti u *bajte* u sekundi (**`Bps`**).

**Za Ethernet mreže možemo općenito koristiti sljedeću tablicu:**

| **Širina komun. pojasa (*Bandwidth*) [Mbps]** | **Maksimalna teoretska propusnost [MBps]** |
|---|---|
| 1 Mbps | 128 kBps |
| 10 Mbps | 1,28 MBps |
| 100 Mbps | 12,8 MBps |
| 1.000 Mbps (1Gbps) | 128 MBps |
| 10.000 Mbps (10 Gbps) | 1.280 MBps |

Postoji i još jedan dodatan pojam, a to je stvarna odnosno **efektivna propusnost** mreže, koja varira ovisno o mnogim čimbenicima poput:

- Kvalitete veza i mrežnoj opremi.
- Izvorišnoj i odredišnoj točki komunikacije.
- Kontrolnim i drugim podacima (tzv. zaglavlja), koja unose mrežni protokoli, a koji se nadodaju na *korisne* podatke.

***Pojam propusnosti nije ograničen samo na mreže, tako da i druga sučelja također imaju svoje propusnosti.***



Vezano za propusnosti raznih sučelja (pr. računalne sabirnice, mobilne mreže i drugo) pogledajte poglavlje:
**29.6. Propusnosti sučelja.**

Vratimo se na računalne mreže. Klasični primjer može biti naša lokalna mreža. Zamislimo lokalnu (***LAN***) mrežu u kojoj imamo računala s običnim mrežnim karticama, koje imaju propusnost od 1Gbps te mrežni preklopnik koji također na svim mrežnim sučeljima odnosno utičnicama (portovima) ima propusnost od 1Gbps. U teoriji bi komunikacija između dva računala na ovoj mreži, spojena preko ovog preklopnika, imala također propusnost od 1Gbps. Dakle maksimalnu teoretsku brzinu prijenosa podataka od 128 MB/s (mega**bajta** u sekundi). Međutim kod slabijih preklopnika koji su deklarirani kao preklopnici koji rade na **Gbps** brzinama, to u praksi nije nužno tako.

Ako smo odabrali prosječan preklopnik i ako računala koja komuniciraju preko njega uopće mogu podnijeti te brzine, dobit ćemo prosječnu efektivnu propusnost od 200Mbps do 500Mbps (od 25MB/s do 63 MB/s), ali se nećemo niti približiti punoj propusnosti gigabitne veze to jest propusnosti od 1Gbps to jest 128 MB/s.



Često se pojam brzine krivo zamjenjuje s propusnošću mreže.
Tako primjerice ***1Gbps*** ne označava brzinu već propusnost od konkretno jednog gigabita u sekundi.

Dodatno postoji i još jedan mali detalj; ako sve promatramo s razine aplikacije, recimo našeg programa preko kojega primjerice kopiramo podatke s jednog na drugo računalo, a to je efikasnost mrežnih protokola. Ako govorimo o TCP/IP protokolu, svaki sloj (ovog) mrežnog protokola dodaje i svoja zaglavlja pa se tako smanjuje efektivna propusnost (korisnih) podataka.

Naime upotrebom TCP/IP protokola u mrežnoj komunikaciji, podaci koji se šalju ili primaju s mreže razlamaju se u manje cjeline koje nazivamo mrežni paketi. Pri tome svaki mrežni paket osim samih podataka koje prenosi, mora sadržavati identifikatore veze, poput: IP adresa pošiljatelja i primatelja, MAC adresa pošiljatelja i primatelja, TCP/UDP portova (izvorišnih i odredišnih) i tako dalje. Jedan TCP/IP paket na mreži izgleda okvirno ovako: (prikazali smo OSI slojeve paketa uz samo neke osnovne detalje):

| ***OSI 1*** | ***OSI 1*** | ***OSI 2*** | ***OSI 2*** | ***OSI 2*** | ***OSI 3*** | **OSI 5 - 7 Podaci [DATA]** ←46- 1460 bajta → | ***OSI 2*** | ***OSI 1*** |
|---|---|---|---|---|---|---|---|---|
| *Preambula* | *SFD* | *Dest* ***MAC*** | *Src* ***MAC*** | *EtherType (za OSI 3)* | ***(IP)*** *Src IP, Dst IP, Protokol (za OSI 4), TTL, Flags, ToS, ...* | ***OSI 4*** ***(TCP ili UDP)*** *Src **port**, Dst **port**, ... (UDP ili TCP)* | *FCS (CRC)* | *Interframe gap* |
| | | | | | ***20 bajta*** | **20 - 60 bajta TCP ili 8 bajta UDP + 46 - 1460 bajta podaci** | | |
| | | ***6 bajta*** | ***6 bajta*** | ***2bajta*** | ← MTU veličina do 1500 bajta → | | ***4 Bajta*** | |
| ***7 bajta*** | ***1 bajt*** | *Ethernet sloj* ← ***veličina do 1518 bajta*** → | | | | | | ***12 bajta*** |
| ← ***Ukupno svih podataka „na mreži“ do prosječno 1538 bajta*** → | | | | | | | | |

Iz primjera paketa vidimo koliko u slučaju upotrebe TCP/IP protokola, koji je praktično osnova modernog interneta, unutar svakog mrežnog paketa postoje zaglavlja na raznim **OSI** odnosno **TCP/IP** slojevima. U ovom slučaju možemo reći da je primjerice, ako gledamo MTU veličinu mrežnog paketa na OSI sloju 3 (IP sloj prema TCP/IP protokolu) od 1500 bajta podataka, 1460 bajta aplikacijskih podataka koje prenosimo i koje možemo nazvati *korisnim* podacima.



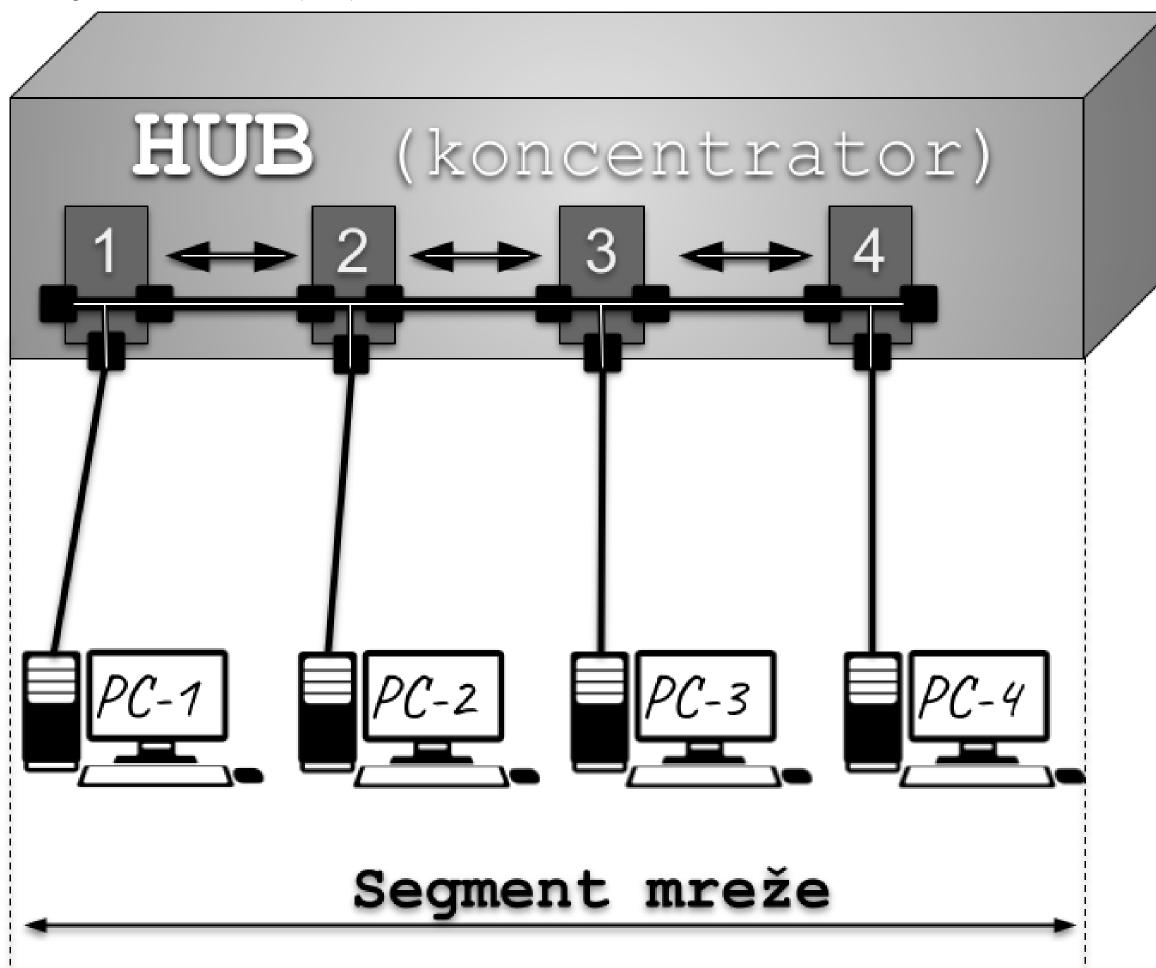
Na efektivnu propusnost utječu i mnogi konfigurabilni parametri, poput onih koje smo opisali u poglavljima:
**24.2.8. Kontrola protoka (*TCP Window* i *Window scaling*).**
**24.2.9. Nadzor zagušenja (*Congestion control*).**
**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**
**25.2. Dodatne mogućnosti linuxa.**
**25.5. Hardverski ubrzane mrežne funkcionalnosti.**

**Izvori informacija: (42),(44),(756),(757),(758),(K-10).**

# 19.3. CSMA/CD i nasljeđe prošlosti

**CSMA/CD (*Carrier Sense Multiple Access With Collision Detection*)([759]) i kolizijska domena.**
Prije vremena preklopnika (*switcheva*) koristili su se uređaji zvani koncentratori odnosno ***HUB***-ovi. Računala i druga mrežna oprema se na njih spajala na isti način kao na preklopnike prema topologiji zvijezda [*star*]: sva računala i oprema, svaki na svoje mrežno sučelje na koncentratoru (***HUB***-u), prema logičkoj shemi vidljivoj na slici 154. ***Hub*** znan i kao *ethernet čvorište*, *mrežno čvorište*, *repetitorsko čvorište*, *višeportni repetitor* ili jednostavno *koncentrator*, bio je mrežni hardverski uređaj za povezivanje više (*Ethernet*) mrežnih uređaja zajedno u jedan segment mreže. Ovakav uređaj imao je više mrežnih sučelja (portova), u kojima se signal doveden na ulaz bilo kojeg sučelja pojačavao te slao na izlaz svakog drugog sučelja, osim dolaznog.

**Slika 154. Logička shema rada koncentratora (HUB-a).**




Pošto se na *koncentratorima* (***HUB***-ovima) dijelila zajednička veza jer je on praktično bio sâmo pojačalo električnih signala, veza je prema tome bila jednosmjerna to jest ***Half Duplex***. Što znači da je omogućavala sâmo primanje ili slanje podataka (mrežnih paketa) na mrežu, u bilo kojem trenutku. Stoga se u praksi stalno događalo da su dva ili više računala u jednom djeliću vremena krenula u slanje podataka te bi dolazilo do tzv. ***kolizije***.

Tu je uskakao ***CSMA/CD*** protokol koji je tada detektirao ovaj slučaj koji se zove ***kolizija***. Dakle aktivirao se mehanizam detekcije kolizije (Engl. *Collision Detection*) koji inicijalno radi tako da nakon detekcije kolizije svaki od pošiljatelja mora sačekati određeni nasumični broj od nekoliko milisekundi te probati slati podatke ponovno, i tako iz početka. Svaki puta kada se detektira kolizija tj. istovremeno slanje podataka na mrežu od strane dva ili više računala u mreži. Osim toga *koncentrator* (***HUB***) je također mogao otkriti koliziju, pri kojoj bi poslao poseban signal zastoja (tzv. **jam** signal) na svim mrežnim sučeljima (portovima), kako je također definirano prema **CSMA/CD** protokolu.

Upotrebom preklopnika (*switcheva*) više nema kolizije, ako se koristi dvosmjerni: ***Full Duplex*** način rada mrežnih sučelja. Sâmim time i kolizijska domena više ne postoji odnosno u najgorem slučaju je moguća na razini jednog mrežnog sučelja (*porta*) na preklopniku i to sâmo, ako je ono konfigurirano u ***Half Duplex*** načinu rada. Važno je znati kako je zbog kompatibilnosti unatrag i dalje ostao ***Half Duplex*** način rada svakog sučelja (*porta*), na koji treba paziti. Problem se može pojaviti jer se svako mrežno sučelje (*port*) na preklopniku i krajnje računalo ili mrežna oprema mogu krivo "dogovoriti" te uspostaviti ***Half Duplex*** umjesto ***Full Duplex*** načina rada; najčešće u slučaju upotrebe ***Auto Negotiation*** protokola.
Više o *duplex* načinu komunikacije u jednoj od sljedećih cjelina.

**Izvori informacija: (759),(K-10).**

## 19.3.1. Što su MAC adrese

**MAC** (*Media Access Control*) **adresa** je jedinstvena adresa ugrađena u svaku mrežnu karticu, od strane proizvođača. Isto tako je i svako mrežno sučelje (*port*) na preklopniku ili bilo kojem mrežnom uređaju također mrežna kartica koja mora imati svoju jedinstvenu *MAC* adresu. *MAC* adresa je 48 bitna, označava se sa 12 heksadecimalnih brojeva grupiranih po dva i obično odvojenih sa znakom **:** ili **–**. Prvih šest heksadecimalnih brojeva označava proizvođača, a ostalih šest su često redni brojevi, iako proizvođač može imati i neku drugu logiku označavanja. U tablici pogledajmo izgled **MAC** adresa:

| **Identifikator tvrtke proizvođača (ili sustava)** | | | **Oznaka (najčešće redni broj proizvođača)** | | |
|---|---|---|---|---|---|
| 8 bitova (oktet) | 8 bitova (oktet) | 8 bitova (oktet) | 8 bitova (oktet) | 8 bitova (oktet) | 8 bitova (oktet) |
| *Organizationally unique identifier (**OUI**)* | | | Najčešće redni broj | | |
| **← 48 bita →** | | | | | |

Dakle, na osnovi ***MAC*** adrese možemo prepoznati proizvođača sâme mrežne kartice: ***Intel***, ***Broadcom***, ***Marvell***, ***Mellanox***, … Tako primjerice MAC adresa: `5C-31-92-37-48-92` pripada opsegu adresa rezerviranom od strane tvrtke **Cisco**.

### Universally administered addresses (UAA) vs. locally administered addresses (LAA)

Važno je znati da **MAC** adrese mogu biti iz dvije kategorije adresa:

- ***Universally administered addresses (UAA)*** odnosno univerzalno administrirana adresa jedinstveno se dodjeljuje uređaju od strane njegovog proizvođača. Prva tri okteta identificiraju organizaciju koja je izdala identifikacijsku oznaku i poznati su kao organizacijski identifikatori [*Organizationally unique identifier*] (**OUI**). Ostatak adrese: zadnja tri okteta za **MAC-48** i **EUI-48** (što je standardno označavanje MAC adresa za: ***Ethernet***, ***802.11***, **Bluetooth**, ***Token Ring i ATM***, kao i: ***FC***, ***FDDI***, ***SAS*** i druge ) ili pet za **EUI-64** (što je prošireno označavanje za: ***IPv6***, ***Infiniband*** i druge) dodjeljuje ista organizacija na gotovo bilo koji način koji želi, kako smo već objasnili.
- ***Locally administered addresses (LAA)*** odnosno lokalno administrirane adrese razlikuju se od univerzalno administriranih adresa postavljanjem odnosno dodjeljivanjem vrijednosti 1, drugom najmanje značajnom bitu prvog okteta adrese. Ovaj bit se također naziva **U/L** bit, skraćeno od *Universal/Local*, koji identificira kako se adresa administrira (i koristi).

**Primjena u virtualizaciji (univerzalne adrese koje se administriraju lokalno)**

U virtualizaciji, *hipervizori* imaju vlastite **OUI** [*Organizationally unique identifier*] oznake, poput sljedećih *hipervizora*::

- ***QEMU (KVM)*** - koji koristi sljedeći opseg MAC adresa: **`52-54-00`**`-xx-xx-xx`.
- ***VMware*** - koji koristi sljedeći opseg MAC adresa: **`00-50-56`**`-xx-xx-xx`.
- **XEN** - koji koristi sljedeći opseg MAC adresa: **`00-CA-FE`**`-xx-xx-xx`.
- **Microsoft** - koji koristi sljedeći opseg MAC adresa: **`00-15-5D`**`-xx-xx-xx`.
- **Proxmox VE** – koristi sljedeći opseg MAC adresa: **`BC-24-11`**`-xx-xx-xx`.

Važno je razumjeti da se svakom novom virtualnom računalu odnosno njegovoj virtualnoj mrežnoj kartici dodjeljuje **MAC** adresa iz nekog od gornjih opsega adresa (prva tri bajta [*3x8 bitova*]), a posljednja tri bajta da budu jedinstvena na lokalnoj mreži.

Vidjeli smo da fizičke mrežne kartice i mrežne kartice virtualnih računala moraju imati svoje MAC adrese, koje ih identificiraju na OSI sloju dva. Važno je razumjeti da svoje jedinstvene MAC adrese imaju i logička mrežna sučelja poput: **Bridge**, **Bond**, **VETH** i **TAP** jer su to sučelja koja rade na OSI sloju dva [*Ethernet*]. Međutim, to nije slučaj sa primjerice **VLAN** sučeljima jer ona standardno koriste MAC adresu primarnog mrežnog sučelja (iako je i to moguće promijeniti).

**Unicast naspram multicasta (I/G bit)**

Najmanji bit prvog okteta adrese naziva se **I/G**, ili ***Individualni/Grupni bit***. Kada je ovaj bit 0 (nula), namjena ovakvog mrežnog okvira je slanje do samo jednog primatelja. Ovakva vrsta prijenosa naziva se ***unicast***. Važno je znati da se *unicast* okviri prenose do svih čvorova unutar kolizijske domene. Međutim ako je najmanji bit prvog okteta postavljen na 1 (tj. druga heksadecimalna znamenka je neparna), mrežni okvir označava *multicast* MAC adresu, što znači da se ovakve MAC adrese koriste za *multicast* komunikaciju. Pogledajmo tablicu s rezerviranim opsezima MAC adresa za posebne namjene:

| I/G ili U/L | Univerzalno administrirana | Lokalno administrirana |
|---|---|---|
| **Unicast** | x**0**-xx-xx-xx-xx-xx<br>x**4**-xx-xx-xx-xx-xx<br>x**8**-xx-xx-xx-xx-xx<br>x**C**-xx-xx-xx-xx-xx | x**2**-xx-xx-xx-xx-xx<br>x**6**-xx-xx-xx-xx-xx<br>x**A**-xx-xx-xx-xx-xx<br>x**E**-xx-xx-xx-xx-xx |
| **Multicast** | x**1**-xx-xx-xx-xx-xx<br>x**5**-xx-xx-xx-xx-xx<br>x**9**-xx-xx-xx-xx-xx<br>x**D**-xx-xx-xx-xx-xx | x**3**-xx-xx-xx-xx-xx<br>x**7**-xx-xx-xx-xx-xx<br>x**B**-xx-xx-xx-xx-xx<br>x**F**-xx-xx-xx-xx-xx |



Pogledajte i sljedeća poglavlja:
**22.3.1. Oblik *multicast* poruke (paketa).**
**19.4. Mrežni most i preklopnik (*Bridge* i *switch*).**
**27.1. Virtualizacija.**

**Izvori informacija: (1442),(1443).**

# 19.4. Mrežni most i preklopnik (*Bridge* i *switch*)

Kako bi se problem s kolizijskom domenom odnosno s kolizijom ublažio, mreža na kojoj su računala spojena pomoću *koncentratora* (**HUB-ova**), morala se razdijeliti na segmente u kojima je svaki od segmenata bio u svojoj kolizijskoj domeni.
Da bi to bilo moguće, napravljeni su uređaji koji su imali sâmo nekoliko mrežnih sučelja na koja su se spajali *koncentratori* te se mreža s njima tako segmentirala. Ovi uređaj se zovu ***premosnici*** ili ***mrežni mostovi*** odnosno ***Bridgevi***.

Međutim, prvo moramo razumjeti da podaci mrežom putuju u mrežnim paketima, a pri tome svaki pojedini mrežni paket minimalno mora sadržavati **MAC** adresu pošiljatelja i primatelja. Premosnici u radu svaki mrežni paket na svakom svôm mrežnom sučelju promatraju na OSI sloju dva (OSI 2) odnosno gledaju mu **MAC** adresu s koje je paket došao odnosno gledaju mu takozvanu izvorišnu (Engl. *Source*) **MAC** adresu. Oni potom provjeravaju na koju odredišnu **MAC** (Engl. *Destination*) adresu paket mora biti prebačen odnosno isporučen.

Ovakvi uređaji su se zvali *mrežni mostovi* ili *premosnici* odnosno *Bridge-evi* jer su kao mostovi povezivali više segmenata iste mreže. Iste mreže, ako gledamo na razini IP adresnog opsega, gledano na OSI sloju tri (OSI 3).
To znači kako se lokalna mreža koja može biti unutar istog opsega IP adresa, segmentirala (razdvojila) na više nezavisnih dijelova, što se tiče kolizije. Ovi uređaji su preteča današnjih preklopnika (*switcheva*), ali su u to vrijeme imali sâmo nekoliko mrežnih sučelja, poput današnjih usmjerivača. Kako ne bi bilo zabune, ovi uređaji su prema svojoj funkcionalnosti preklopnici s malim brojem mrežnih sučelja na kojima se u radu, kao i na preklopnicima (*switchevima*) gradila tablica preklapanja i sâmo preklapanje, a koje je odrađivao centralni procesor (*CPU*) ovakvog uređaja.
Dakle ovi uređaji su razdvajali dva ili više segmenata iste mreže, unutar koje postoji kolizijska domena, preklapanjem (prebacivanjem) samo onih mrežnih paketa, između segmenata mreže, između kojih se odvija komunikacija. Dakle mrežni paketi koji ne moraju komunicirati s drugim segmentima mreže se zadržavaju unutar svog segmenta mreže odnosno svoje kolizijske domene i ne prosljeđuju se drugdje (i ne zagušuju ostatak mreže nepotrebno).



Osim navedene primjene ***bridge*** uređaji se koriste i za druge namjene, poput njihove primjene za povezivanje različitih mrežnih sučelja; primjerice fizičkih mrežnih kartica i virtualnih mrežnih sučelja (**TUN**, **TAP**, **VETH**,...), što se najčešće koristi u virtualizaciji. Stoga pogledajte i sljedeća poglavlja:

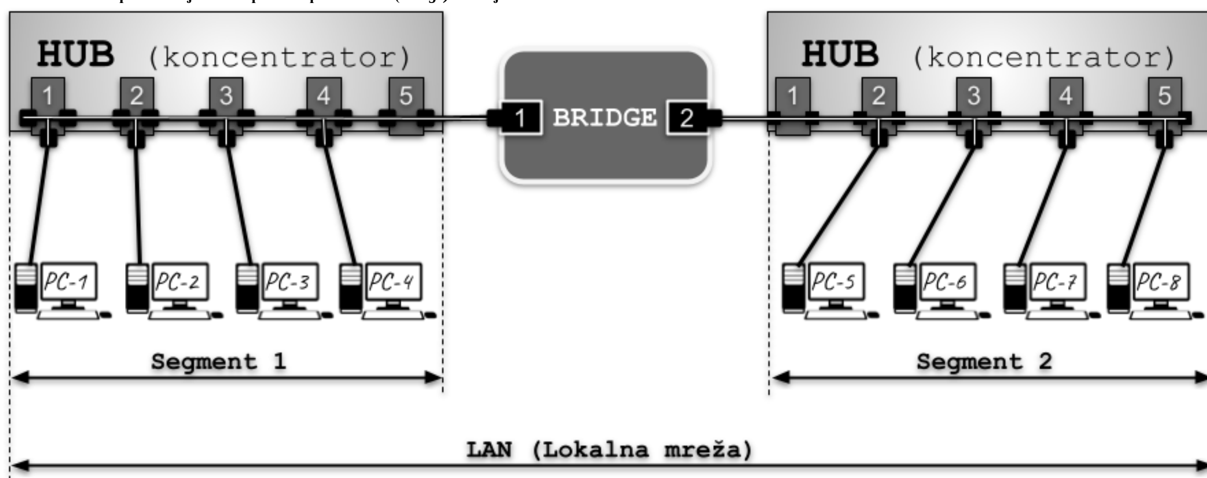

**20. Kako radi mreža na OSI sloju 2.**
**20.1. Oblik mrežnih okvira na OSI slojevima 2 i 1.**
**20.6.1. Mrežni most (bridge) odnosno premosnik.**
**20.6.3. TUN i TAP - posebna mrežna sučelja.**
**20.6.4. VETH - posebno mrežno sučelje.**
**27.1. Virtualizacija.**




Standard koji definira rad mrežnog mosta (*premosnika*) tj. ***bridge*** uređaja, ali i ***preklopnika*** je: 802.1D.

Pogledajmo i način spajanja **HUB** i **bridge** uređaja te segmentaciju lokalne mreže pomoću njih, kako je vidljivo na slici 155.
I danas se pojam *Bridge ili mrežni most* (premosnik) koristi unutar svih operativnih sustava koji nam nude mogućnost povezivanja dvaju (ili više) mrežnih sučelja u mrežni most (*bridge)* odnosno u način rada u kojemu se mrežni paketi prebacuju iz jednog mrežnog sučelja na drugo, na osnovi tablice preklapanja (OSI 2 sloj) odnosno ovisno o MAC adresama (izvor/odredište).

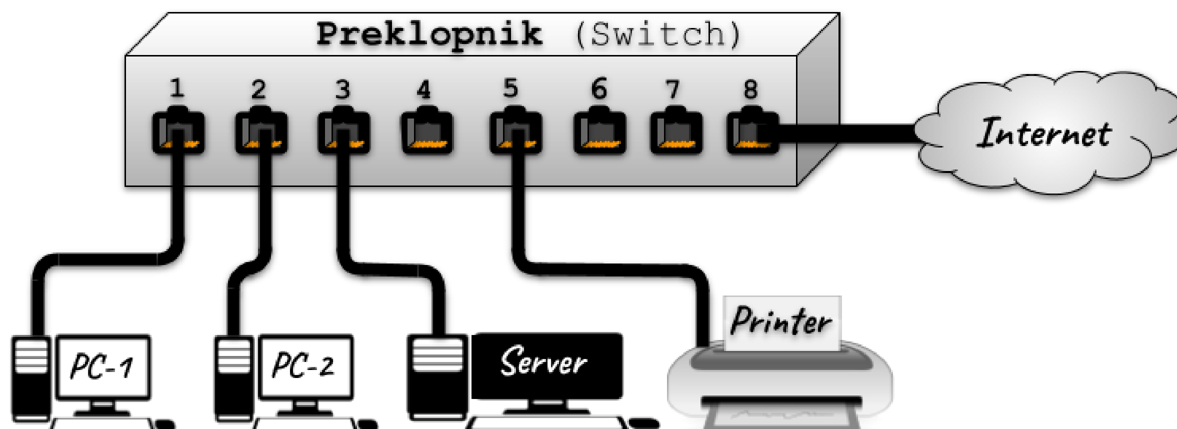
**Slika 155. Shema povezivanja mreža pomoću premosnika (*Bridge)* uređaja.**




U ovakvom načinu rada, vaše računalo (na slici vidljivo kao **BRIDGE**) postaje mali preklopnik odnosno takozvani ***bridge***.

Kasnijim razvojem i ubrzavanjem hardvera razvijali su se i preklopnici, koje možemo promatrati kao *bridge* uređaje s više mrežnih sučelja (mrežnih kartica/*portova*). Pogledajmo i način spajanja računala i drugih mrežnih uređaja na preklopnik (slika 156). Kod ovakvog načina spajanja (logički spoj *zvijezda*); sva računala, poslužitelji i druga mrežna oprema, spajaju se direktno na preklopnik, i to svaki mrežni uređaj na svoju mrežnu utičnicu na preklopniku, slično kao kod spajanja s mrežnim koncentratorom. Međutim preklopnik (engl. *switch*) prema načinu rada nije samo pojačalo signala kao što je bio koncentrator, već on u potpunosti radi kao mrežni most (*bridge*), što znači da za svaki mrežni paket provjerava MAC adrese pošiljatelja i primatelja te na osnovu njih odlučuje gdje (na koje svoje mrežno sučelje) proslijediti svaki pojedini mrežni paket.

**Slika 156. Način spajanja računala i drugih mrežnih uređaja na preklopnik (engl. *switch*).**




Važnost ovakvog povezivanja i rada je u činjenici kako se kolizijska domena smanjila praktično na razinu svakog pojedinog mrežnog sučelja, u slučajevima, ako se koristi dvosmjerni način komunikacije. Ovakav dvosmjerni način komunikacije svakog pojedinog mrežnog sučelja poznat je kao ***Full Duplex*** način rada.

Treba znati i da je zbog kompatibilnosti unatrag i dalje ostao moguć i jednosmjerni odnosno ***Half Duplex*** način rada svakog mrežnog sučelja, na koji treba paziti. Problem se može pojaviti jer se svako mrežno sučelje na preklopniku i krajnje računalo ili mrežna oprema mogu krivo "dogovoriti" te uspostaviti ***Half Duplex*** umjesto ***Full Duplex*** načina rada. Ovo se događa u slučaju upotrebe ***Auto Negotiation*** protokola za koji uglavnom niste niti svjesni kako je već u upotrebi na vašem preklopniku. Ali o njemu nešto kasnije.

**Izvori informacija: (1015),(1016),(1442),(K-6).**

## 19.4.1. Metode preklapanja paketa: *cut-through* i *store and forward*

Kako smo počeli govoriti o preklapanju i preklopnicima, moramo se upoznati i s načinima na koje preklopnici rade.
Iako preklopnici gledaju sâmo OSI sloj dva (OSI 2) svakog mrežnog paketa i na osnovu pročitane odredišne MAC adrese svakog pojedinog mrežnog paketa odlučuju na koje mrežno sučelje (Engl. *port* ili *interface)* će proslijediti svaki pojedini mrežni paket, postoje dva načina na osnovu kojih se to može odraditi.

Prvi način je ***cut-through***[(45)] pri kojem preklopnik gleda sâmo odredišnu MAC adresu (svakog) mrežnog paketa te odmah kreće s prosljeđivanjem (preklapanjem) tog paketa na odredište. Ova metoda je brza i ne zahtijeva puno resursa sa strane preklopnika. Dakle jeftinija je što se tiče hardvera preklopnika. Mana joj je u tome što se ne provjerava *FCS* (*frame check sequence*) dio paketa koji sadrži provjerni zbroj (izračunat pomoću ***CRC*** algoritma). Zbog toga se događa da se neispravni paketi kojima se u ovoj metodi ne provjerava integritet, uredno prosljeđuju (preklapaju) prema odredištu. U tom slučaju se mrežom potencijalno šire neispravni paketi. Njih može generirati neispravna mrežna kartica ili uzrok mogu biti neke druge greške s izvorišnog računala. Ova metoda je nastala u trenutku razvoja *ethernet* preklopnika, u vremenima kada su mreže radile na 10Mbps brzinama, a kasnije i 100Mbps, te je većina mrežnih uređaja radila u ***Half Duplex*** načinu rada u kojemu je dolazilo i do kolizije. Stoga je u teoriji kada dođe do kolizije paketa na mreži moguće da su neki paketi oštećeni te da su manji od 64 *bajta*. Ovakvi paketi se nazivaju ***runt*** paketi odnosno ***runt*** mrežni okviri. Dakle jedino što ovakvi preklopnici dodatno provjeravaju je minimalna veličina mrežnog okvira (paketa) na OSI sloju dva, točnije svi mrežni okviri koji su manji od 64 *bajta* se odmah odbacuju. I danas neki proizvođači jeftinije mrežne opreme proizvode uređaje koji rade na ovakav način.

Drugi način rada je ***store and forward***[(46)] kod kojega se za svaki primljeni mrežni paket (okvir) ponovno preračunava *FCS* provjerni zbroj pomoću istog *CRC* algoritma. Potom se on uspoređuje s onim koji se nalazi u mrežnom paketu. Ako preračunati *CRC* i onaj koji se nalazi u paketu nisu isti, paket se odbacuje. Jasno je kako je za ovo potrebna dodatna snaga i brzina odnosno snažniji hardver preklopnika. Prednost je i u tome što se neispravni mrežni paketi ne prosljeđuju dalje te ne zagušuju mrežu. Mana je dodatno vrijeme obrade odnosno kašnjenje, koje uvelike ovisi o brzini hardvera ovakvih preklopnika.

**Izvori informacija: (45),(46).**

# 19.5. Duplex

*Duplex* komunikacija je uz brzinu mreže (10Mbps/100Mbps/1.000Mbps/…) drugi važan parametar vezan uz sâmu brzinu. Ovo možda zvuči pomalo čudno. O čemu se radi?.

## 19.5.1. Half Duplex

U starim mrežama koje su radile na brzinama od 10 Mbps ili 100 Mbps, a koje su dijelile jedan medij, dakle iz vremena korištenja *koncentratora* odnosno takozvanih mrežnih ***HUB***-ova, u svakom trenutku je bilo moguće ili primati ili slati podatke preko mreže. Naime sjetimo se kako se komunikacija odvijala preko jednog dijeljenog medija [kabel].

Zbog problema, ako je netko u takvoj mreži pokušao slati podatke istovremeno s nekim drugim, je i uveden ***CSMA/CD*** protokol koji je to riješio tako da je natjerao sve koji su spojeni na mrežu, da u jednom djeliću sekunde sâmo jedan od njih može slati podatke, a ostali sâmo slušati. Ovakav način komunikacija nazivamo ***Half Duplex***. Važno je razumjeti da je ***Half Duplex*** način komunikacije u sâmo jednom smjeru: ili se šalje ili se prima podatke, nikako istovremeno.
Dakle *half duplex* način komunikacije označava jednosmjernu komunikaciju.

## 19.5.2. Full Duplex

Razvojem mreža i prelaskom s mrežnih *koncentratora* (***HUB***-ova) na preklopnike više nije bilo potrebe za ***Half Duplex*** komunikacijom, iako neki uređaji i danas traže takav način rada. Upotrebom preklopnika komunikacija više ne teče kroz jedan dijeljeni medij između svih koji su spojeni na mrežu već je praktično svaka veza između računala ili nekog drugog uređaja na mreži i preklopnika kao posrednika, jedini dijeljeni medij. Stoga je moguće istovremeno i primati i slati podatke. Naime prva komponenta između vas i susjednog računala je upravo preklopnik, prema kojem svako računalo može imati dvosmjernu komunikaciju i na fizičkom sloju (Engl. OSI *Layer 1*). Pošto preklopnici rade na mrežnom sloju dva (Engl. OSI *Layer 2*) mrežne pakete obrađuje preklopnik i preklapa (prebacuje) ih s jednog računala, preko sebe, do drugog računala i obratno, za svaki paket zasebno. Tako da je dvosmjerna komunikacija zadržana. Ovu mogućnost dvosmjerne komunikacije odnosno i primanje i slanje paketa mrežom u isto vrijeme nazivamo ***Full Duplex***.

Starije *Ethernet* veze (do 100Mbps) postižu ***full-duplex*** rad istovremenom upotrebom dvije parice (dva pâra kabela) unutar istog mrežnog kabela (***UTP***) ili upotrebom dva optička vlakna koja su izravno povezana sa svakim umreženim uređajem: jedan par ili vlakno služi za primanje paketa, dok se drugi koristi za slanje paketa. Međutim, novije varijante *Etherneta*, počevši od 1000BASE-T (1Gbps *Ethernet*), koriste sve četiri parice unutar (**UTP**) kabela, na način da se isti kanali koriste za komunikaciju u svakom smjeru istovremeno (za slanje i primanje podataka). U svakom slučaju, s ***full-duplex*** radom, sam kabel postaje okruženje bez kolizije te se udvostručuje maksimalni ukupni prijenosni kapacitet (tzv. kumulativna propusnost) koju podržava svaka *Ethernet* veza.



Pogledajte i poglavlje:
**19.2.1.1. Bakar.**

To znači da, ako recimo naša propusnost (kolokvijalno rečeno *brzina*) mreže iznosi 1.000 Mbps (1 Gbps) to znači kako tom propusnošću možemo istovremeno i primati i slati podatke, što opet znači kako maksimalna ukupna propusnost onda iznosi 2.000 Mbps (2 Gbps), što neki proizvođači i iznose u karakteristikama mrežnih uređaja ili komponenti, pa je stoga to vidljivo, kao u tablici:

| ***Standardna (jednosmjerna)*** **propusnost (*brzina*) komunikacije** | ***Full Duplex*** **(dvosmjerna) propusnost (*brzina*) komunikacije** |
|---|---|
| 10 Mbps | 20 Mbps |
| 100 Mbps | 200 Mbps |
| 1.000 Mbps | 2.000 Mbps (2 Gbps) |
| 10.000 Mbps (10 Gbps) | 20.000 Mbps (20 Gbps) |
| 100.000 Mbps (100 Gbps) | 200.000 Mbps (200 Gbps) |



***Ethernet*** standard i njegove pod varijante možete vidjeti u standardu: **IEEE 802.3**.

**Izvori informacija: (760),(K-6),(K-10).**

# 19.6. Auto Negotiation

Kako su *Ethernet* mreže rasle s brzinom od 10 Mbps prema 100 Mbps i 1.000 Mbps (1Gbps) ili dalje sa 10.000 Mbps (10 Gbps) i više, morala je postojati mogućnost korištenja smanjenja brzina odnosno kompatibilnosti unatrag.

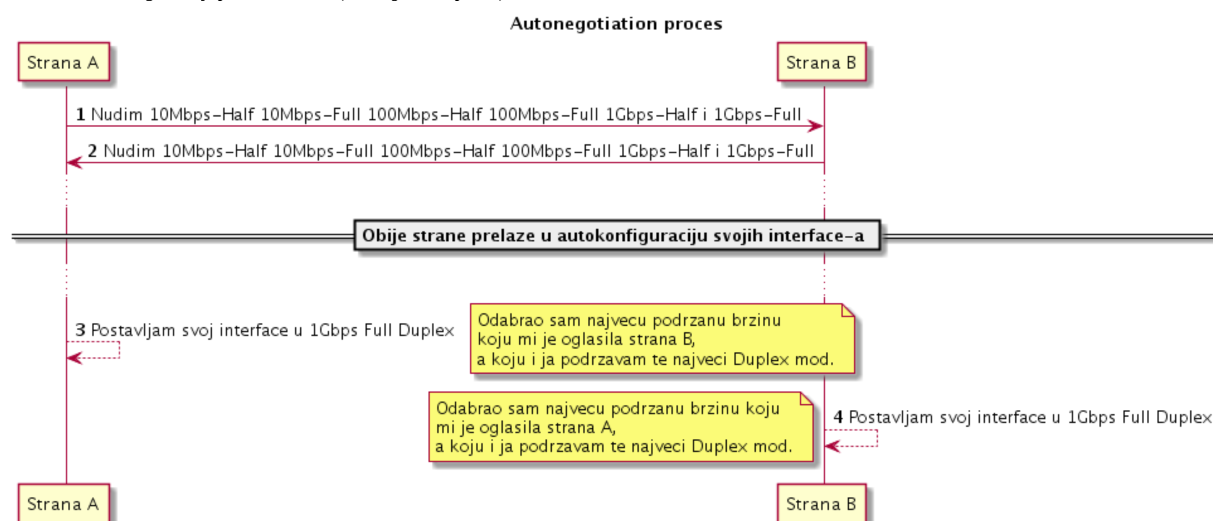
Dakle kako bi se sa 1.000 Mbps (1 Gbps) mrežnom karticom spojili na mrežu koja radi recimo na 100 Mbps ili 10Mbps.
Osim fizičke kompatibilnosti unazad te ručnog konfiguriranja brzine i *Duplex* načina rada uveden je mrežni protokol imena: ***Auto Negotiation*** koji omogućava da dvije direktno spojene strane u mrežnoj komunikaciji poput primjerice:
računalo ←→ preklopnik, dogovore:

- Brzinu rada: 10 Mbps, 100 Mbps, 1.000 Mbps (1Gbps) ili 10.000 Mbps (10 Gbps) i više.
- *Duplex* način rada: od ***Half*** do ***Full .***

Upotrebom ovog protokola, svaka strana šalje drugoj strani koje sve načine rada podržava. Nakon toga, svaka strana za sebe pronalazi najkompatibilniji način rada i postavlja svoje mrežno sučelje u taj način rada. Primjerice strana **A**: 100Mbps *Full Duplex*. Druga strana također postavlja svoje mrežno sučelje u najkompatibilniji način rada; primjerice strana **B** također postavlja 100Mbps *Full Duplex*. Dakle i brzina i *duplex* način rada se uvijek podešavaju od najveće brzine prema najmanjoj brzini i isto tako od ***Full Duplex*** do ***Half Duplex*** prema parametrima primljenim od druge strane, prema principu najvećeg zajedničkog nazivnika.

Navedene dvije strane u komunikaciji su zapravo preklopnik i uređaj (računalo) koji spajamo na njega.
Pogledajmo ovaj proces odnosno mehanizam rada u dijagramu (slika 157):

**Slika 157. Proces dogovaranje parametara rada (*Autonegotiation* proces).**




Problem može biti i u tome što nakon ovog procesa obje strane ne provjeravaju za koji način rada se suprotna strana odlučila.

Naime, razni proizvođači mrežne opreme: od mrežnih kartica, preko preklopnika, usmjerivača i drugih mrežnih komponenti često se **ne** drže standarda ili ih ne implementiraju na najbolji način. Stoga se može dogoditi da se dvije strane koje dogovaraju brzinu i *duplex* način rada, krivo dogovore te tada počinju problemi i usporavanja u mreži.



Preporuka je da se ***Auto Negotiation*** (761) ne koristi za međusobno spajanje preklopnika, usmjerivača, poslužitelja ili druge infrastrukturno važne mrežne opreme.

Najčešći vidljivi problem kod krivo dogovorene brzine je veliki gubitak paketa u oba smjera (*Upload/Download*) unutar **LAN** mreže. Dok je najčešći vidljivi problem krivo dogovorenog *duplex* načina rada velika nesimetrija odnosno razlika u brzinama ovisno o smjeru toka podataka: slanja ili primanja (tj. *Upload/Download*) unutar ***LAN*** mreže. U takvim slučajevima možemo primjerice imati veliku brzinu kopiranja podataka s klijenta na poslužitelj, ali u suprotnom smjeru (poslužitelj → klijent) brzina će biti višestruko manja.

**Zbog čega smo rekli "*unutar LAN mreže*"?**
Zbog toga što ovo nije mjerilo kod danas najčešće ***WAN*** tehnologije za "*izlaz na internet*" tj. ***ADSL***-a koji je i kako ime kaže ***asimetričan*** (*Asymmetric Digital Subscriber Line*) dakle projektiran da u jednom smjeru ima veliku brzinu (*Download*), a u drugom vrlo malu (*Upload*). Dok mrežne tehnologije u lokalnim (***LAN***) mrežama ne rade na takav način jer su ***simetrične***.

***Asimetričan** prijenos podataka (Download/Upload) imaju i neke druge mrežne tehnologije, poput mobilnih mreža:* ***3G/4G/5G.***

**Izvori informacija: (761),(1125),(K-6).**

## 19.6.1. Konfiguracija brzine te *duplex* i *auto negotiation* načina rada

U Linuxu pomoću programa odnosno naredbe `ethtool` možemo pristupiti mrežnoj kartici na najnižoj razini pa joj stoga možemo i mijenjati navedene parametre rada.

Kako bismo mogli koristiti ovaj program, moramo instalirati dva programska paketa na sljedeći način (iako su obično već instalirani):

```
yum -y install ethtool net-tools
```

**Pogledajmo i nekoliko primjera za `eth0` mrežnu karticu**

Ali prvo pogledajmo kako provjeriti postavke: ***duplex***, ***brzine (speed)*** i ***auto-negotiation*** parametara rada mrežne kartice:

```
ethtool eth0 | egrep "Speed|Duplex|Auto-negotiation"
      Speed: 1000Mb/s
      Duplex: Full
      Auto-negotiation: on
```

← vidimo da je za mrežnu karticu ***eth0*** postavljena brzina 1000Mbps (1Gbps) te ***Full Duplex***, a ***auto-negotiation*** je uključen.

**1.** Pogledajmo kako isključiti ***auto negotiation*** način rada:

```
ethtool -s eth0 autoneg off
```

**2.** Odnosno kako uključujemo ***auto negotiation*** način rada:

```
ethtool -s eth0 autoneg on
```

**3.** Sada isključimo *auto negotiation*, te ručno postavimo *duplex* na *full* i brzinu na 1.000Mbps (1Gbps):

```
ethtool -s eth0 speed 1000 duplex full autoneg off
```



Sve navedeno što smo konfigurirali, bit će aktivno sâmo do trenutka restarta računala.
***S navedenim naredbama mijenjamo postavke kojima*** `ethtool` ***pristupa preko MII sučelja mrežne kartice!.***

4. Za trajnu konfiguraciju postavki iz primjera **3.**, za **eth0** mrežno sučelje, potrebno je u takozvani *ifcfg* datoteku: `/etc/sysconfig/network-scripts/ifcfg-eth0` dodati sljedeći redak:

```
ETHTOOL_OPTS="speed 10000 duplex full autoneg off"
```



Za trajnu konfiguraciju mrežne kartice i detalje oko konfiguracije (**Red Hat v.6- v.8**), pogledajte poglavlje:
**25.6.5.1.2 Trajna (permanentna) ručna konfiguracija mreže.**

Za novije sustave (**Red Hat 9+**), za trajnu konfiguraciju postavki mreže, pogledajte poglavlja:
**26.1. Network Manager.**
**25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**



Važno je razumjeti da naredba `ethtool` radi na **OSI** slojevima **1** i **2**.



Za druge detalje oko rada naredbe **ethtool**, pogledajte poglavlje:
**25.7.10. Naredba ethtool.**

**Izvor informacija:** `man ethtool`.

# 19.7. Latencija odnosno kašnjenje

Latencija odnosno kašnjenje (engl. delay) je, s općeg gledišta, vremensko kašnjenje između određenog uzroka i posljedice, neke fizičke promjene u promatranom sustavu. Obično od trenutka slanja do trenutka zaprimanja poslanog signala. Mrežno kašnjenje je karakteristika dizajna i performansi telekomunikacijske mreže, a određuje kašnjenje pojedinog bita podataka koji putuju mrežom od jedne komunikacijske krajnje točke do druge. Obično se mjeri u djelićima sekunde. To znači da se kašnjenje odnosno ***Latencija*** odnosi na vrijeme potrebno informaciji odnosno bîtu podataka da prođe put od izvora do odredišta. U konačnici ovo je vrijeme koje je potrebno kako bi podaci preko mreže došli od jednog do drugog računala koja međusobno komuniciraju.

**Vrijeme prijenosa podataka**
Vrijeme od početka do kraja prijenosa poruke naziva se **vrijeme prijenosa (***engl. Transmission time***)**.
Kod računalnih mreža odnosno prijenosa digitalnih podataka, to je vrijeme od slanja prvog bita do posljednjeg bita poruke koja je napustila prijenosni čvor (računalo/mrežni uređaj).
Vrijeme prijenosa paketa u sekundama može se dobiti iz veličine paketa u bitovima i brzine prijenosa u bitovima kao:

$$Vrijeme\ prijenosa\ paketa = \frac{Veličina\ paketa}{Brzina\ prijenosa\ (propusnost)}$$

Pretpostavimo brzinu prijenosa *Ethernet* mreže od 100 Mbit/s i maksimalne veličine paketa od **1526** bajta, pa imamo:

$$Maksimalno\ vrijeme\ prijenosa = \frac{1526\ x\ 8\ bitova}{100\ Mbps\ x\ 10^6\ bitova/s} = 122\ \mu s$$

Sada pogledajmo usporednu tablicu za razne brzine *Ethernet* mreža te vrijeme koje je potrebno da se kroz mrežu prenese mrežni paket veličine **1526** bajta:

| **Propusnost** | 100 Mbps | 1 Gbps | 10 Gbps | 20 Gbps | 40 Gbps | 100 Gbps |
|---|---|---|---|---|---|---|
| **Vrijeme prijenosa (engl. *transmission time*)** | **122,08 µs** | **12,21 µs** | **1,22 µs** | **0,61 µs** | **0,31 µs** | **0,12 µs** |

U praksi, ukupno kašnjenje (*latencija*) se događa zbog:
- Prolaska mrežnih paketa kroz medij. Mrežni medij pri tome može biti: kabel, optičko vlakno ili eter (*zrak*).
- Vremena potrebnog mrežnim uređajima kako bi zaprimili, obradili i proslijedili dalje mrežni okvir ili paket.
- Vremena potrebnog odredišnoj točki (obično računalu) da prihvati i obradi mrežni paket.

Okvirna vremena kašnjenja, koja možemo vidjeti u stvarnim uvjetima su sljedeća:
- U lokalnim mrežama (***LAN***), između računala unutar lokalne mreže: ispod mili sekunde (ms) ili do par mili sekundi (ms).
- U većim lokalnim (***LAN***) mrežama s nekoliko usmjerivača: do nekoliko mili sekundi (ms).
- Za ***WAN*** *mreže (mreže koje pokrivaju velike udaljenosti)***:** x**DSL,** optika ili slično, unutar jednog grada: <10 ms.
- Za ***WAN*** mreže - pr. x**DSL** optika ili slično, između gradova (pr. 200-300km) oko 20-30 ms.

Pogledajte i tablicu sa stvarnim izmjerenim prosječnim kašnjenjem (latencijom) između navedenih gradova:

| | **New York** | **Paris** | **Sydney** | **Tokyo** | **Zagreb** |
|---|---|---|---|---|---|
| **Amsterdam** | 78 ms | 11 ms | 249 ms | 233 ms | 24 ms |
| **Frankfurt** | 89 ms | 10 ms | 256 ms | 250 ms | 18 ms |
| **London** | 75 ms | 15 ms | 257 ms | 233 ms | 34 ms |
| **Los Angeles** | 73 ms | 143 ms | 180 ms | 108 ms | 158 ms |
| **New York** | — | 73 ms | 208 ms | 174 ms | 112 ms |
| **Sydney** | 208 ms | 239 ms | — | 327 ms | 249 ms |
| **Tokyo** | 174 ms | 257 ms | 327 ms | — | 218 ms |

**Izvor mjerenja latencije među gradovima: (1447).**

**Mjerenje latencije**
S obzirom da uvjeti na mreži često fluktuiraju, mjerenje latencije u prijenosu paketa između iste dvije krajnje točke više puta može davati različite rezultate. Zbog toga latencija bilo kojeg pojedinačnog paketa nije potpuno mjerodavna.
Primjerice i **TCP** kao transportni protokol izračunava latenciju pomoću posebnog polja unutar TCP paketa (kako je definirano u **RFC 1323**). Dakle TCP izračunava prosječno povratno vrijeme (tzv. *Round Trip Time [**RTT**]*), kako bi uopće mogao ispravno funkcionirati. Štoviše TCP protokol periodički ponovno preračunava **RTT** vrijeme, da se prilagodi trenutnim uvjetima na mreži.



Vezano za kašnjenje (latenciju) i druge parametre koji utječu na nju na razini **TCP** protokola, pogledajte poglavlja:
**24.2.8. Kontrola protoka (*TCP Window* i *Window scaling*).**
**24.2.8.1. Veza između latencije, propusnosti i TCP Window size parametra.**
**24.2.9. Nadzor zagušenja (*Congestion control*).**

Osim navedenog, ovisno o transportnim mrežnim uređajima, odnosno njihovoj konfiguraciji i opterećenosti, kao i odabirom drugog puta paketa, latencija se može mijenjati s vremenom (i mijenja se).



Vezano za kašnjenje (latenciju) ovisno o konfiguraciji mrežnih uređaja, pogledajte i poglavlje:
**24.2.14. Primitak paketa izvan redoslijeda slanja (*Out-Of-Order*).**

Pogledajmo kako na jednostavan način vidjeti latenciju između našeg i udaljenog računala, pomoću naredbe `ping`.
Pomoću ove naredbe šaljemo poseban *ICMP* paket na odredišnu IP adresu, s koje dobivamo odgovor, te se za svaku ovu poruku izračunava vrijeme odziva. Pogledajmo kako poslati četiri ***ping*** poruke na *Google* **DNS** poslužitelj na IP adresi 8.8.4.4.

```
ping -c 4 8.8.4.4
PING 8.8.4.4 (8.8.4.4) 56(84) bytes of data.
64 bytes from 8.8.4.4: icmp_seq=1 ttl=118 time=16.4 ms
64 bytes from 8.8.4.4: icmp_seq=2 ttl=118 time=17.6 ms
64 bytes from 8.8.4.4: icmp_seq=3 ttl=118 time=16.8 ms
64 bytes from 8.8.4.4: icmp_seq=4 ttl=118 time=18.2 ms

--- 8.8.4.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 16.423/17.264/18.204/0.709 ms
```

Iz ispisa vidimo da se šalju četiri *icmp* mrežna paketa (*ping* poruke), te se redom na njih dobivaju odgovori, iz kojih se vide vremena latencije, redom: 16,4ms, 17,6ms, 16,8ms i 18,2ms. Potom imamo vidljive statistike da su poslana četiri paketa te je odgovoreno na sva četiri, što znači da nema gubitaka (`0% packet loss`) te da je ukupno vrijeme trajanja testiranja: `time 3010ms` (dakle 3010ms to jest oko 3 sekunde).
Zatim u zadnjem retku vidimo sumirane statistike latencije, označene kao `rtt` (*Round-trip-time*) i to:

- `min` – minimalno vrijeme odziva (latencije).
- `avg` – prosječno vrijeme odziva (latencije).
- `max` – maksimalno vrijeme odziva (latencije).
- `mdev` – označava odstupanje (*devijaciju*) vremena odziva (latencije) izračunato prema formuli za *medijan devijaciju*.



Vezano za naredbu `ping`, pogledajte poglavlje:
**25.7.1. Naredba ping.**

**Treperenje/podrhtavanje (engl. *Jitter*)**
Treperenje ili podrhtavanje (engl. *Jitter*) i latencija mjerni su podaci koji se koriste za procjenu performansi mreže. Osnovna razlika između *treperenja* i *latencije* je da se latencija definira kao kašnjenje u mreži, dok se treperenje definira kao promjena (*devijacija*) u latenciji. U svakom slučaju povećanje treperenja i latencije negativno utječe na performanse mreže, stoga ih je važno redovito nadzirati. Nadalje, kada se brzina dvaju uređaja razlikuje, latencija i treperenje se povećavaju; pri tome nastala zagušenja mogu uzrokovati prepunjavanje međuspremnika te odbacivanje paketa i prekidanje (*pucanje*) veze.
U prijašnjem ispisu naredbe `ping` u zadnjem retku smo vidjeli `mdev` statistiku. Ona izračunava razliku (*devijaciju*) u mjerenjima odnosno pokazuje eventualno treperenje u latenciji. Naime, u normalnom radu svake mreže latencija se povremeno mijenja, zbog već navedenih razloga. Međutim stalne promjene latencije se smatraju treperenjem; primjerice, ako nam je odziv na *icmp* (**ping**) prema krajnjoj točki konstantno oko 17 ms., a onda svako malo imamo povećanje na primjerice 30 ms. ili imamo još veća i češća odstupanja. To znači da nešto nije u redu s mrežom, mrežnim uređajima, s našim računalom, ili ako se mjerenje radi s virtualnog računala, eventualno s *hipervizorom* odnosno fizičkim računalom na kojem se pokreće virtualno računalo. Ako se radi o virtualizaciji, uzroci mogu biti i mnogi od mrežnih slojeva unutar virtualizacijske platforme.
Pogledajmo jedno mjerenje latencije uz izraženo treperenje na mreži, koje se pojavilo tri puta kada je latencija rasla do 115.ms.

**Slika 158. Mjerenje latencije uz naglašeno treperenje (jittering) koje se u mjerenju pojavilo u tri navrata.**




**Izvori informacija: (762),(1446),(1447),(1449),(K-6), `man ping`, `man 7 tcp`, RFC 1072, RFC 1323.**

# 19.8. OSI model i TCP/IP model

**OSI** model (Engl. *Open Systems Interconnection model*) je konceptualni slojeviti model mrežne komunikacije koji opisuje funkcije i međusobnu komunikaciju između njegovih slojeva. Održava se od strane ***ISO*** organizacije (*International Organization for Standardization*) pod oznakom ISO/IEC 7498-1. Na osnovi *OSI* modela razvijen je **TCP/IP** model mrežne komunikacije. Naziv TCP/IP potječe od dva pojma: *Transmission Control Protocol* (**TCP**) i *Internet Protocol* (**IP**) kao prva dva protokola koja su bila razvijena u skupini TCP/IP protokola. Današnja skupina odnosno grupa TCP/IP protokola sadrži cijeli nîz raznih protokola; zbog toga govorimo o skupu protokola naziva TCP/IP.

Ova grupa protokola često se naziva i ***DoD*** model zbog toga što je inicijalno razvijan od strane **DARPA** agencije (*Defense Advanced Research Projects Agency*) unutar Ministarstva obrane Sjedinjenih Američkih Država (*Department of Defense*) krajem 1960. i početkom 1970. godine. Tijekom narednih godina razvijale su se nove inačice: TCP v1, TCP v2, TCP v3 i IP v3, te **TCP/IP v4** koji se koristi i danas (uz IP v6.). TCP/IP čini skup protokola nužan za komunikaciju između računala i mrežnih uređaja. Današnji internet je baziran na TCP/IP skupu protokola.
On se sastoji od nekoliko slojeva od kojih svaki odrađuje svoj dio funkcionalnosti te predaje podatke na obradu sljedećem sloju.

U tablici je vidljiva usporedba OSI i TCP/IP modela te može poslužiti kao primjer komunikacije TCP/IP protokola.

| OSI model | TCP/IP model |
|---|---|
| 7. Aplikacijski sloj | 4. Aplikacijski sloj |
| 6. Prezentacijski sloj | |
| 5. Sloj sesije | |
| 4. Transportni sloj | 3. Transportni sloj [primjerice: **TCP** ili **UDP** ili **SCTP** protokoli] |
| 3. Mrežni sloj | 2. Mrežni sloj [**IP** protokol] |
| 2. Sloj podatkovne veze | 1. Sloj podatkovne veze |
| 1. Fizički sloj | |

Prema ovoj tablici će se najbolje shvatiti što se događa tijekom komunikacije dva računala TCP/IP protokolom. Upoznajmo se i okvirno s komunikacijom prema OSI odnosno TCP/IP modelu komunikacije.

**U komunikaciji između dva računala:** *Izvor/pošiljatelj [Source] → Odredište/primatelj [Destination],* **dolazi do sljedećeg:**

**Aplikacija** odnosno na ***aplikacijskom*** sloju *[ovo je TCP/IP sloj 4, tj. po **OSI**-ju su to slojevi: 7,6 i 5]*. Primjerice Web preglednik se spaja na Web poslužitelj. Pri tome aplikacija (Web preglednik) formira podatke koje želi slati na drugo računalo (poslužitelj) i prosljeđuje ih nižem sloju.

**Transportni sloj;** primjerice TCP *[ovo je TCP/IP sloj 3, tj. po **OSI**-ju je to sloj 4]* preuzima podatke od gornjeg sloja i dodaje svoj dio: u ovom slučaju izvorišne i odredišne TCP *portove*, *CRC* provjerni zbroj (engl. *checksum*) za provjeru integriteta podataka, te druge dijelove paketa pa ih potom prosljeđuje nižem sloju.

**Mrežni sloj** odnosno **Internet sloj** koji koristi IP protokol *[ovo je TCP/IP sloj 2, tj. po **OSI**-ju je to sloj 3]* preuzima podatke od gornjeg sloja i dodaje svoj dio, a to su: *izvorišna IP adresa* (našeg računala) *i odredišna* IP adresa (web poslužitelja na koji se spajamo) kao i drugi dijelovi, te ih prosljeđuje još nižem sloju.

**Sloj podatkovne veze** odnosno ***Network Access*** sloj *[ovo je TCP/IP sloj 1, tj. po **OSI**-ju su to slojevi 2 i 1]* preuzima podatke od gornjeg sloja i dodaje svoj dio (**MAC** adrese pošiljatelja i primatelja te zaglavlja) te šalje podatke sâmoj mrežnoj kartici koja ih pretvara u električne (ili optičke) signale i šalje ih na mrežni medij odnosno mrežu. Važno je znati da mrežnim medijem prolaze paketi podataka dogovorene maksimalne veličine. Ako je na početku procesa ugnježđivanja (*enkapsulacije*) podataka utvrđeno da treba poslati više podataka nego što je maksimalna veličina paketa (pr. 1500 bajta = 1.5kB) tada dolazi do njihovog razlamanja na manje pakete odnosno fragmente. Pri tome svaki od fragmentiranih paketa sadrži svoj serijski broj, kako bi se ispravno posložili na odredišnoj strani, te sastavili u izvorni oblik. S druge komunikacijske strane, na odredišnom računalu, proces je obrnut.

S druge strane mrežna kartica prima električne impulse pretvara ih u slijed digitalnih podataka (0 i 1) koje preuzima prvi sloj, obrađuje ih i prosljeđuje drugom sloju,… i tako redom sve do aplikacijskog sloja koji to sve slaže u prvobitni oblik podataka, koji je poslan s izvorišnog računala. Dakle sve se vraća u suprotnom smjeru (od TCP/IP sloja 1. do sloja 4.).
Ovaj cijeli proces ćemo detaljnije objasniti u poglavljima koja slijede.



Za više detalja dobro proučite poglavlja koja slijede!

**Izvor informacija: (763),(K-6),(K-10).**

## 19.8.1. TCP/UDP Portovi

Upotrebom TCP/IP protokola u mrežnoj komunikaciji, podaci koji se šalju ili primaju s mreže razlamaju se u manje cjeline koje nazivamo mrežni paketi. Pri tome svaki mrežni paket osim samih podataka koje prenosi, mora sadržavati identifikatore veze, bez kojih se mrežni paket ne može poslati na mrežu (navodimo ih od nižih do viših OSI slojeva):

- **OSI 2** – izvorišna i odredišna **MAC** adresa odnosno *Ethernet* [**MAC**] adresa pošiljatelja i primatelja paketa.
- **OSI 3** – izvorišna i odredišna **IP** adresa: IP adresa pošiljatelja i IP adresa primatelja paketa.
- **OSI 4** – TCP/UDP portovi koji su identifikatori klijentske aplikacije i poslužiteljske strane (aplikacijskog poslužitelja):
    - Izvorišni (TCP ili UDP) port.
    - Odredišni (TCP ili UDP) port.

Svaki pojedini TCP/IP mrežni paket, izgleda okvirno ovako (prikazali smo OSI slojeve paketa uz samo neke osnovne detalje):

| ***OSI 1*** | ***OSI 1*** | ***OSI 2*** | ***OSI 2*** | ***OSI 2*** | ***OSI 3 (IP)*** | ***OSI 4 (TCP/UDP)*** | ***OSI 2*** | ***OSI 1*** |
|---|---|---|---|---|---|---|---|---|
| *Preambula* | *SFD* | *Dest **MAC*** | *Src **MAC*** | ***EtherType*** (*za OSI 3*) | *Src IP, Dst IP,* ***Protokol*** (za OSI 4), *TTL, …* | *Src **port**, Dst **port**, (SEQ Nr. , ACK Nr.,…)* **OSI 5 - 7 Podaci [DATA]** | *FCS* | *Interframe gap* |

Upotrebom TCP/IP protokola, svaki mrežni paket, kako smo vidjeli, osim **MAC** adresa i **IP** adresa koje specificiraju pošiljatelja i primatelja paketa, na transportnom sloju mora imati definirane i takozvane ***portove***. Preciznije svaka strana konekcije, na TCP/UDP sloju, u komunikaciji ima dodijeljenu 16-bitnu oznaku kojom se identificira aplikacijski protokol, za obje strane komunikacije; kako za slanje tako i za primanje, koja se zove ***port***. TCP ili UDP dopuštaju maksimalno 65.536 portova [$2^{16}$]. Naime za svaki aplikacijski protokol je definiran ***port*** preko kojega se on koristi. Pomoću ***portova*** svako računalo može u svakom trenutku paralelno komunicirati s više aplikacija (protokola) jer svaka od njih koristi drugačiji ***port***. Portovi se dijele na:

- **Rezervirane** i točno definirane portove iz opsega portova: **0-1023**.
- **Registrirane** portove, registrirane od strane organizacija i tvrtki (iz opsega: **1024-49151**).
- **Dinamičke** portove odnosno sve ostale koji nisu prethodno definirani (obično iz opsega: **49152-65535**).
    - **Efemerni portovi** su unutar gornjeg opsega (ili posebnih opsega), a koriste se kao privremeni portovi:
        - Prema **RFC 6056**, oni su iz opsega: **1024–65535.**
        - Prema **IANA** i **RFC 6335** oni su iz opsega: **49152–65535**.
        - Za **Linux** je to često opseg: **32768–60999**, dok za duge operativne sustave ovisi kojeg se pravila pridržavaju.

Neki od rezerviranih portova i pripadajućih protokola koji ih koriste vidljivi su u tablici:

| Broj porta | Naziv | Opis |
|---|---|---|
| 20 | FTP Data | *File Transfer Protokol* za podatke. |
| 21 | FTP | *File Transfer Protokol* za kontrolu. |
| 22 | SSH, SFTP i SCP | *Secure Shell*: udaljeni pristup shell-u i *Secure File Transfer* Protokol te *Secure Copy* protokol. |
| 25 | SMTP | Simple Mail Transport Protokol. |
| 53 | DNS | Domain Naime Server protokol (***DNS***). |
| 67 i 68 | BOOTP/DHCP | BOOTP i DHCP Protokoli. |
| 69 | TFTP | Trivial FTP Protokol. |
| 80 | HTTP | Hyper Text Transfer Protokol. |
| 123 | NTP | Network Time Protokol. |

***Odredišni port definiran od klijenta prema poslužitelju definira aplikacijski protokol kojim će se odvijati komunikacija.***

Osim portova važno je i s kojim transportnim protokolom se koristi niži protokol. Od transportnih protokola unutar *TCP/IP* grupe protokola nalaze se **TCP** i **UDP** protokoli, kao i poseban **SCTP**. Oni su zaduženi za prijenos paketa preko mreže.

**Slijedi napredni dio!**
Kod sustava koji moraju ostvarivati veliki broj aktivnih konekcija; poput rada u funkciji usmjerivača, ***load balancera***, ***proxy*** poslužitelja, ali i drugih poslužitelja koji opslužuju veliki broj aktivnih konekcija na sustav (pr. 10.000 ili više) poput web poslužitelja, standardne postavke sustava često nam nisu dovoljne.

Naime svaka otvorena konekcija odnosno mrežni *socket* identificira se s pet parametara: protokolom, izvorišnom IP adresom i pripadajućim portom te odredišnom IP adresom i njenim pripadajućim portom. S obzirom na činjenicu kako je u ***TCP/IP*** protokolu (konkretno u transportnom sloju) za broj portova definirana 16 bitna vrijednost ($2^{16}$), to znači kako je moguće imati maksimalno 65.535 portova za upotrebu, za svaku pojedinu konekciju odnosno za svaku određenu IP adresu.



Vezano za detalje oko broja portova, pogledajte poglavlje:
**24. Transportni protokoli (OSI sloj 4).**

To kod primjene usmjerivača, posredničkih (***proxy***) poslužitelja ili sličnih namjena gdje imamo jednu jedinu IP adresu preko koje se odvija sav promet prema unutarnjoj mreži, ali i prema *internetu* (ili nekoj vanjskoj mreži), to postaje problematično.
Jedan od primjera ovakvog rada je tzv. ***NAT*** odnosno translacija adresa.



Vezano za detalje oko ***NAT*** translacija adresa, pogledajte poglavlja:
**23.4.4.Translacija adresa (*NAT*).**
**23.4.4.1. Source NAT (SNAT).**

U slučaju upotrebe jedne IP adrese preko koje teče sav mrežni promet prema *internetu*, potrebno je identificirati sve dolazne i odlazne mrežne konekcije pri čemu je osnovni i u ovom slučaju problematični identifikator, upravo ovaj broj *porta*. Međutim na sustavu za upotrebu nemamo dostupne sve *portove* jer su *portovi* do 1.024 rezervirani i ne mogu se uopće koristiti, a dodatno je rezerviran i određeni broj *portova* za druge potrebe. Sve to u konačnici daje nam znatno manji opseg slobodnih *portova* koji se ovdje mogu koristiti kao identifikatori svake pojedine konekcije. Naime kada se recimo s web preglednikom spajamo na udaljeni web poslužitelj (odredišni port 80 (*HTTP*)), kao naš izvorišni odnosno *Source/lokalni port* uzima se nasumični *port* (iz navedenog opsega *portova*). Ovaj izvorišni port identificira našu lokalnu aplikaciju, a taj port se naziva ***efemerni*** port.



Vezano za detalje oko ***izvorišnog*** porta pogledajte poglavlje:
**24.2.2. Kako se uspostavlja TCP veza.**

Dok se odredišni *port* postavlja na 80 i takav mrežni paket se šalje prema odredišnoj IP adresi (web poslužitelja) na njegov odredišni *port* (80) koji i identificira komunikacijski servis (*HTTP*/*Web*). Pri tome je naš izvorišni (*Source*) *port* identifikator sustavu da sadržaj mrežnog paketa treba dostaviti našoj aplikaciji (*Web* pregledniku) za konkretni komunikacijski kanal.

Pogledajmo koji opseg *portova* nam je dostupan za korištenje u Linuxu, a nalazi se postavljen u *sysctl* varijabli: `net.ipv4.ip_local_port_range`:

```
sysctl net.ipv4.ip_local_port_range
net.ipv4.ip_local_port_range = 32768    60999
```

Vidimo kako je to opseg od `32768` do `60999` što nam ukupno daje ***28.231*** *portova* za upotrebu po svakoj pojedinoj IP adresi.
Iz ovoga proizlazi da naš usmjerivač ili *proxy* poslužitelj ne može nikako imati više od upravo ***28.231*** aktivnih konekcija.
S time da ne zaboravimo da konekcije, a ovdje mislimo na TCP, i kada se zatvore imaju neko vrijeme čekanja prije isteka, obično 60 sekundi pa u međuvremenu ovi *portovi* nisu oslobođeni odnosno TCP konekcije nisu u potpunosti zatvorene. Stoga se ovaj broj *portova* dijeli na sve otvorene i poluotvorene odnosno poluzatvorene konekcije.



Vezano za detalje oko zatvaranja ***TCP*** konekcija, pogledajte poglavlje:
**24.2.16.1. Timeri i stanja veze.**

*Navedeni opseg portova se naziva i opsegom: prolaznih, trenutnih, odnosno privremenih portova (engl.* ***Ephemeral ports****).*

U ovakvim slučajevima primjene stoga možemo povećati ovaj opseg; primjerice: ***10.000*** – ***65.535*** čime dobivamo ukupno na upotrebu ***55.535*** portova. To možemo trajno promijeniti dodavanjem sljedećeg retka u datoteku: `/etc/sysctl.conf`:

```
net.ipv4.ip_local_port_range = 10000    65535
```

I potom ponovnim čitanjem ove datoteke i primjenom njenih postavki. Sa sljedećom naredbom učitavamo nove vrijednosti:

```
sysctl -p
```

I to je to, osposobili smo naš Linux sustav da može imati istovremeno otvoreno ***55 535*** portova za svaku pojedinu IP adresu.



Vezano za ograničenje maksimalnog ukupnog broja aktivnih konekcija na računalu, pogledajte i poglavlje o ***vatrozidu*** i to cjelinu: **26.7.2.1. Primjeri.**

Naime u ovoj cjelini vidjet ćete kako se globalno ograničava maksimalan broj aktivnih konekcija na računalo (na razini kernela odnosno ***Netfilter*** pod komponente) kod upotrebe *vatrozida*.

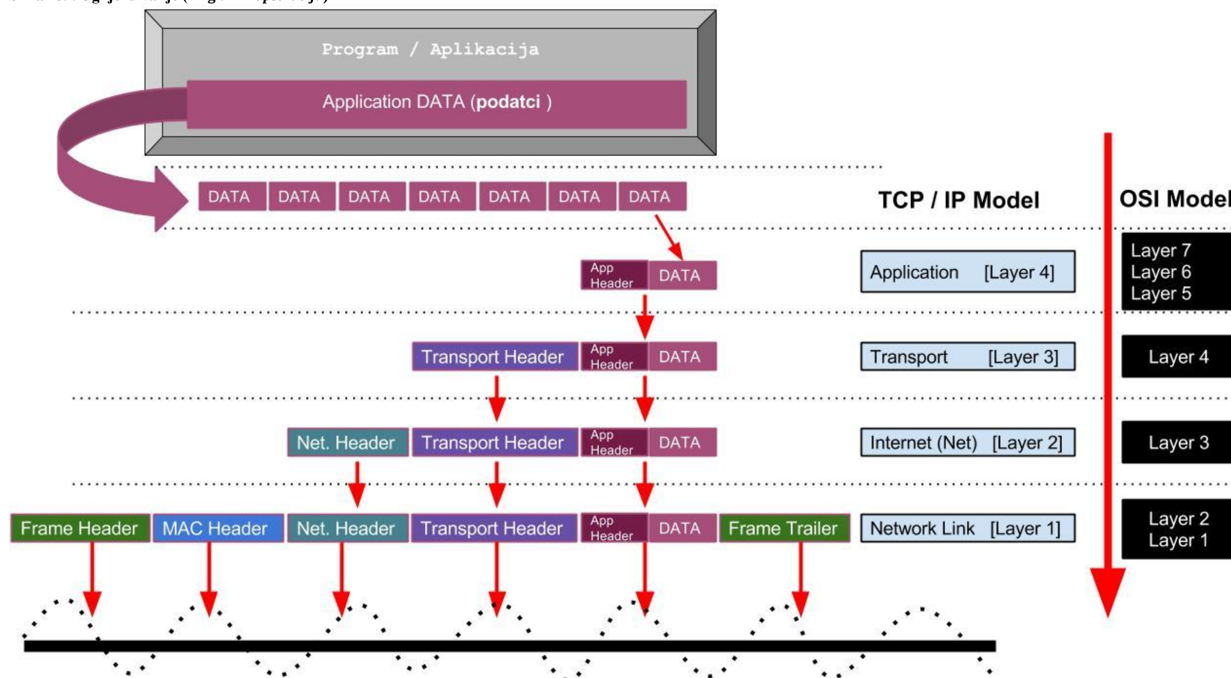
Osim toga, vezano za ograničenja na **TCP** razini pogledajte i poglavlja:
**25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa.**
**24.2.3. Sigurnosni aspekt uspostavljanja TCP veze (*SYN*).**

**Izvori informacija: (335),(769),(770),(771),(773),(774),(775),(1406),(K-6),(K-10),`man 7 tcp`, RFC 6056, RFC 6335.**

## 19.8.2. Ugnježđivanje (Enkapsulacija)

Sada pogledajmo detaljnije što se događa kod ***TCP/IP*** komunikacije. Slika 159 prikazuje ugnježđivanje (*enkapsulaciju*) prema TCP/IP protokolu, s odredišnog (*source*) računala.
Ovdje će vam biti još malo jasniji tôk podataka i veza između TCP/IP slojeva u komunikaciji.

**Slika 159. Ugnježđivanje (Engl. *Enkapsulacija*)**




Zamislimo sljedeći primjer: Naš Web preglednik se želi spojiti na web stranicu na koju upisujemo korisničko ime i lozinku. Promatrat ćemo sâmo dio komunikacije u kojoj se obrađuju uneseno korisničko ime i lozinka.
Sada naš **TCP/IP** protokol uskače u pomoć.

**Aplikacijski sloj** uzima podatke iz sâme aplikacije; u našem slučaju naše korisničko ime i lozinku te ih razdvaja na manje dijelove (*DATA*) koji mogu stati u budući mrežni paket i to u formatu koji je razumljiv višem sloju; konkretno aplikaciji.
Ti naši podaci (ovdje su naše korisničko ime i lozinka koje smo upisali) su na slici prikazani kao [*DATA*].

**Transportni sloj -** transportni sloj preuzima paketić podataka od aplikacijskog sloja te dodaje svoj dio [*Transport Header*], a ovdje se dodaju i izvorišni *(Source)* i odredišni *(Destination) portovi* te se paketi označavaju rednim brojevima. Zatim se dodaju i ostale stvari ovisno koriste li se za transport **TCP** ili **UDP** protokol. Između ostaloga za *TCP* se izračunava i dodaje *TCP* provjerni *zbroj* odnosno *tzv. cheksum*. Na kraju se sve zajedno sa svime iz prethodnog sloja (*Aplikacijskog*) prosljeđuje sljedećem sloju. Prema OSI modelu ovo je *OSI sloj* četiri (***Layer 4***). Izvorišni *port* odnosno ***Source*** *port* se nasumično odabire: minimalni broj je 1.024 (odnosno ***Ephemeral port***) + slučajni broj. Odredišni *port* odnosno ***destination*** *port* označava protokol za komunikaciju (pogledajte tablicu s *portovima*), dakle ovdje će to biti *port* 80 koji predstavlja HTTP protokol.

**Internet sloj** preuzima paket od *transportnog sloja* i dodaje svoj dio koji sadrži i izvorišnu *IP* adresu (pošiljatelja), *odredišnu IP* adresu (primatelja), ali i *IPv4* jednostavni provjerni zbroj odnosno *header checksum* koji se izračunava **SAMO** za zaglavlje, a ne i za podatke (ne izračunava se za DATA dio) i to znatno jednostavnijim algoritmom nego u nižem sloju (dolje).
Potom se dalje sve što sadrži ono što su dodali prethodni slojevi i ovaj sloj, prosljeđuje sljedećem sloju.
Prema OSI modelu ovo je *OSI sloj tri* odnosno ***Layer 3***.

**Network Access/Link** sloj preuzima paket od Internet sloja te dodaje i *izvorišnu* i *odredišnu MAC* adresu. Dakle MAC adrese mrežne kartice ovog računala i računala na koje se šalje paket [ovisno nalazi li se na istoj mreži]. Na ovom sloju se izračunava provjerni zbroj za provjeru integriteta svih podataka, pomoću *CRC* algoritma te se zapisuje u svoje *FCS* polje. Prema OSI modelu ovo je *sloj dva (OSI 2)* odnosno ***Layer 2***. Tada se sve preuzeto od prijašnjeg sloja i onoga što je ovaj sloj dodao, šalje na najniži sloj koji dodaje svoj dio (svoja polja) te sve šalje na mrežnu karticu. Najniži sloj je OSI sloj jedan (***Layer 1***).
Nadalje se sve pretvara u signale za slanje na mrežni medij:

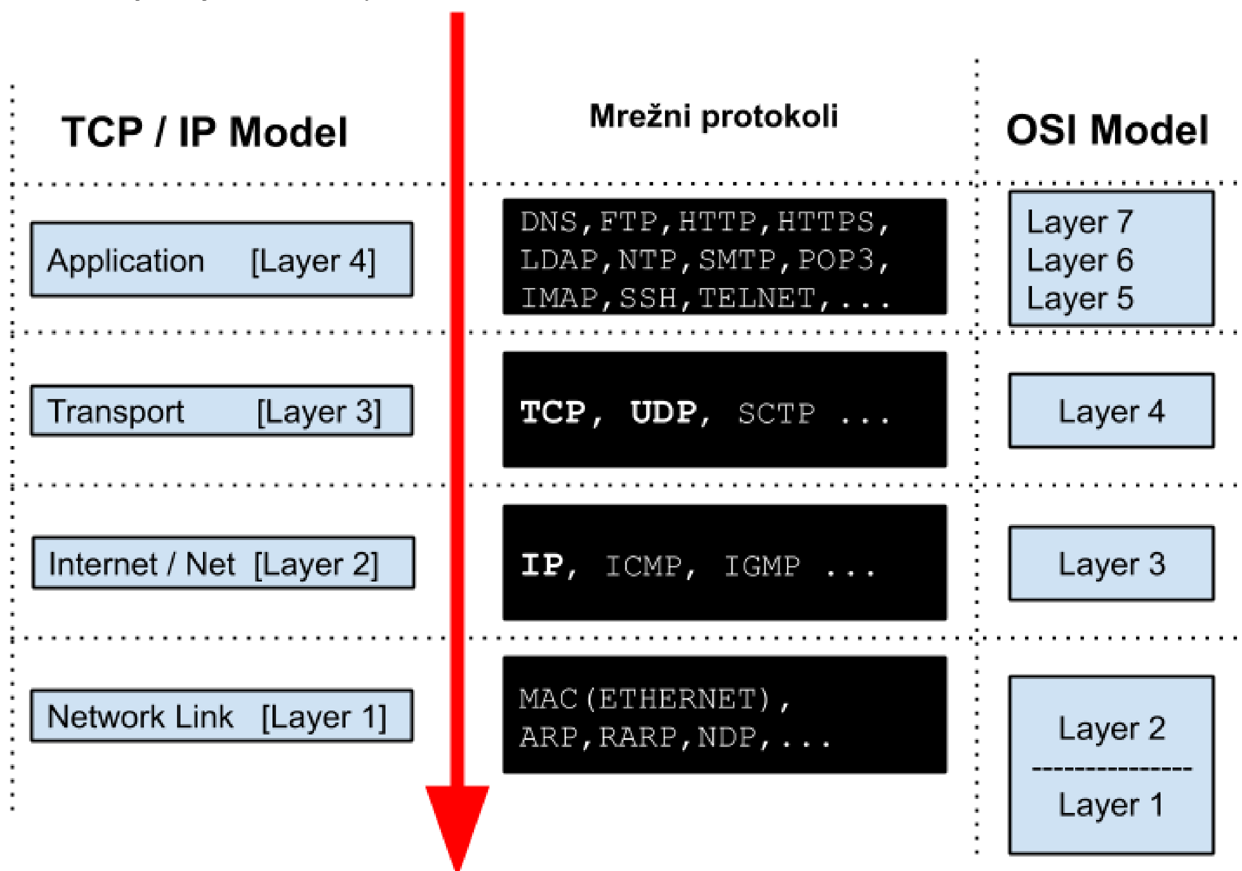
- *Bakar* preko **RJ-45** utičnice odnosno u konačnici u električne signale.
- *Ili Optiku* preko **LC**, **SC**, **ST** ili sličnih konektora odnosno sve se pretvara u svjetlosne impulse.

Ovaj proces se na računalu koje prima podatke odvija u suprotnom smjeru. Odnosno *Web* poslužitelj iz primjera, na koji smo se spajali, prima s mrežnog medija signale koji se pretvaraju u podatke koje preuzima **Network Access/Link** (*OSI Layer 2*) sloj koji provjerava svoj dio i provjerava integritet podataka pomoću novog *CRC* izračuna i usporedbe s onim *CRC* zapisom koji se nalazi unutar mrežnog okvira (paketa). Potom se skida ovaj sloj i prosljeđuje **Internet sloju** (*OSI Layer 3*) koji provjerava svoj dio, te radi svoj provjerni zbroj, i ako je sve u redu, skida svoje dijelove i prosljeđuje sve **Transportnom sloju** (*OSI Layer 4*) koji provjerava svoj dio. Ako je za transport bio korišten TCP, provjerava se ispravan redoslijed primljenih podataka i provjerni zbroj (*TCP checksum*) i ako je sve u redu, skida se dio koji je od ovog sloja i prosljeđuje **Aplikacijskom sloju** uz naznaku broja *porta* jer on naznačava aplikacijski protokol i na kraju krajeva aplikaciju.

**Aplikacijski sloj** izvlači podatke i prosljeđuje ih našoj aplikaciji. Ako je bilo više podataka nego li ih stane u jedan mrežni paket, obrađuju se i primaju svi ostali paketi iz ovog nîza iz kojih se potom izvlače podaci. Svi podaci iz više paketa koji pripadaju jednom nîzu se tada spajaju u konačan oblik podataka koji u ovom slučaju dolaze do Web poslužitelja koji ih onda i prima.
U našem slučaju su to korisničko ime i lozinka.

Cijelo vrijeme govorimo kojem **OSI** sloju pripada koja komunikacija, iako koristimo **TCP/IP** protokol.
To je stoga što se sve uspoređuje prema **OSI** modelu, pa ćete tako i kod karakteristika raznih uređaja vidjeti upravo oznake za **OSI** slojeve (Engl. *OSI Layers*). Sada još pogledajmo koji protokoli se nalaze u kojem od **TCP/IP** slojeva (slika 160).

**Slika 160. Mrežni protokoli prema OSI i TCP/IP slojevima**




Pogledajmo sada sve od navedenog, malo preglednije u tablici, uz popis dodatnih mrežnih sučelja koja nalazimo na navedenim slojevima mreže: **TCP/IP** u odnosu na **OSI** model.

| TCP/IP model | Mrežni protokoli na njemu | OSI model | Posebna mrežna sučelja u Linuxu koja ga također koriste: |
|---|---|---|---|
| Aplikacijski sloj [*Layer 4*] | **DNS, FTP, HTTP, HTTPS,LDAP, NTP, SMTP, POP3,IMAP, SSH,TELNET, SNMP,DAP, MQTT, NNTP,RTP, RIP, SIP, BGP, XMPP,** *NetBIOS, PAP, SMB, RPC* **…** | Slojevi 7,6 i 5 | |
| Transportni sloj [*Layer 3*] | **TCP** ili **UDP** ili **SCTP** kao i: *AH, ESP, NetBIOS, iSCSI,* **…** | Sloj 4 | **VxLAN** |
| Internet sloj [*Layer 2*] | **IP** kao i: *ICMP, IGMP, VRRP, RIP, OSPF, IPSEC, ARP,* **…** | Sloj 3 | **TUN,IPVLAN,IPVTAP** |
| Sloj mreže [*Layer 1*] | **MAC** (za *Ethernet* mreže), te: **ARP**, **RARP**, **NDP**, *STP, LACP, PPP, PPTP, PAP, L2TP, …* | Sloj 2 i Sloj 1 | **TAP,VETH,BRIDGE, BOND,VLAN(802.1Q, MACVLAN,IPVLAN, MACVTAP,MACSEC, VCAN** i **VXCAN** |

**Izvori informacija: (764),(765),(766),(767),(K-6),(K-10).**

# 20. Kako radi mreža na OSI sloju 2

**U ovom cijelom poglavlju obradit ćemo tehnologije, pojmove i mrežna sučelja koja rade na OSI sloju dva (OSI 2), mreže Linuxa.**

## Spustimo se na OSI slojeve dva i jedan

Kako bismo bolje razumjeli pojmove s kojima ćemo se malo kasnije upoznati, moramo detaljnije razumjeti što se događa na OSI slojevima dva i jedan (*OSI 2* i *OSI 1*). Dalje u tekstu govorit ćemo o ***Ethernet*** vrsti mrežnih okvira.
U praksi postoji nekoliko vrsta ***Ethernet*** mrežnih okvira:

- ***Ethernet*** *II* ***(802.3)*** - znan i kao ***Ethernet*** v.2 ili ***DIX***, koji je najčešći u upotrebi, i na kojem ćemo se bazirati.
- ***802.2 LLC*** (*Logical Link Control*), kao i ***802.2 SNAP*** (*Subnetwork Access Protocol*) okviri.

Mrežni okvir je naziv za blok podataka na OSI slojevima jedan (OSI 1) i dva (OSI 2), a koji su uokvireni u ono što često pogrešno nazivamo mrežnim paketima. Možemo sve pojednostaviti i reći: naziv mrežni (***Ethernet***) okvir jest na OSI slojevima jedan i dva (*OSI 1 i OSI 2*), a na višim slojevima ih nazivamo paketima.



**Ethernet** je mrežna tehnologija najčešće u potrebi u lokalnim (**LAN**) mrežama, dok se u mrežama za povezivanje superračunala, zbog potrebe za vrlo malom latencijom i velikom propusnosti, češće koriste: **InfiniBand** i **Omni-Path**.

---

## 20.1. Oblik mrežnih okvira na OSI slojevima 2 i 1

Pogledajmo kako izgledaju poruke odnosno mrežni okviri na OSI slojevima dva i jedan (*OSI 2* i *OSI 1*).
Pogledat ćemo zaglavlje ***Ethernet*** okvira za najmanje mrežne okvire od 64 *bajta*.
Izgled mrežnog ***Ethernet*** okvira, prema standardu **802.3**, na drugom sloju mreže (*OSI 2*) vidljiv je u tablici:

| DMAC | SMAC | ETHER TYPE | PAYLOAD | FCS |
|---|---|---|---|---|
| Odredišna **MAC** | Izvorišna **MAC** | Vrsta (oznaka) protokola s višeg sloja | Podaci (DATA) i sve ostalo s gornjih slojeva (pr. TCP/UDP/IP) | ***CRC*** *Checksum* |
| 6 *bajta* | 6 *bajta* | 2 *bajta* | od 46 do 1500 *bajta* | 4 *bajta* |

Slijedi opis polja:

- `DMAC` – ovo je odredišna (*destination*) *MAC* adresa, na koju se šalje mrežni okvir.
- `SMAC` – ovo je izvorišna (*source*) *MAC* adresa, s koje se šalje mrežni okvir.
- `ETHER TYPE` – ovo je polje u koje se zapisuje o kojem vršnom protokolu se radi odnosno koji se koristi.
  Vršnih protokola postoji cijeli nîz, a navest ćemo ih samo nekoliko [*cijela lista je vidljiva u* (**1195**)]:
  - `0x0800` – predstavlja ***IP*** protokol i to ***IP v.4*** [**RFC 7042**].
  - `0x0806` – predstavlja ***ARP*** protokol [**RFC 7042**].
  - `0x8035` – predstavlja ***RARP*** protokol [**RFC 903**].
  - `0x8100` – predstavlja ***VLAN oznake (802.1Q*** protokol), dok `0x88A8` predstavlja ***QinQ*** (***802.1ad***).
  - `0x8181` – predstavlja ***STP*** protokol (*Spanning Tree*).
  - `0x86DD` - predstavlja ***IP*** protokol i to ***IP v.6***.
  - `0x876C` - predstavlja ***GRE*** protokol [**RFC 1701**].
  - `0x880B` - predstavlja ***PPP*** protokol [**RFC 7042**].
  - `0x8863` i `0x8864` – predstavljaju **PPPoE** (*session stage i access control*) [**RFC 2516** i **RFC 8822**].
  - `0x88CC` - predstavlja ***LLDP*** protokol (*Link Layer Discovery Protocol).*
  - `0x88E5` - predstavlja ***MAC*** *Sec* protokol (*Media Access Control Security*), . . .
- `DATA` - ovo su podaci odnosno ono što se prosljeđuje višim slojevima - prema OSI sloju tri [pr. TCP/UDP/IP i podaci].
- `FCS` – ovo je *frame check sequence*, a sadrži provjerni zbroj (izračunat pomoću *CRC* algoritma), za cijeli mrežni okvir uključujući i zaglavlje s ovog sloja. Dakle za svaki mrežni okvir na ovom sloju se za sva polja računa provjerni zbroj pomoću navedenog *CRC* algoritma, te se sprema u ovo (*FCS*) polje. Kada mrežni okvir dođe na drugu stranu tj. na odredišno računalo, ono ponovno izračunava provjerni zbroj pomoću *CRC* algoritma. Ako je sve u redu: i novo izračunati *CRC* i onaj u paketu su isti, paket se dalje obrađuje, a ako nije u redu, paket se odbacuje.
  Prolaskom mrežnog okvira kroz preklopnik (*switch*), ako se radi o imalo "inteligentnijem" uređaju i sâm preklopnik ponovno, na svakom svom mrežnom sučelju, za svaki pojedini primljeni mrežni okvir ponovno izračunava provjerni zbroj (*FCS*), te ako nije dobar, mrežni okvir se odbacuje kako ne bi nepotrebno zagušivao mrežu odnosno krajnji uređaj na koji se šalje, a koji bi ga kao neispravnog ionako odbacio.

Sada pogledajmo kako naš najmanji mrežni okvir koji sadrži samo 46 *bajta* podataka izgleda na drugom i prvom sloju mreže. Dakle pogledajmo što sadrži *Ethernet* mrežni okvir na drugom sloju mreže (***Layer 2** tj.* ***OSI 2***) u tablici dolje:

| DMAC | SMAC | ETHER TYPE | PAYLOAD | FCS |
|---|---|---|---|---|
| Odredišna **MAC** | Izvorišna **MAC** | Vrsta (oznaka) protokola s višeg sloja | Podaci (DATA) i sve ostalo s gornjih slojeva (pr. TCP/UDP/IP) | ***CRC*** provjerni zbroj odnosno *Checksum* |
| 6 *bajta* | 6 *bajta* | 2 *bajta* | 46 *bajta* | 4 *bajta* |
| ← U k u p n o 64 b a j t a → | | | | |

Zatim pogledajmo što sadrži *Ethernet* mrežni okvir na prvom sloju mreže (***Layer 1** tj.* ***OSI 1***):

| Preambula | SFD | Sve s gornjeg (OSI 2) sloja | Interframe gap (razmak) |
|---|---|---|---|
| Označava početak okvira | Nakon *Start Frame Delimitera* slijedi dio s višeg sloja | Sadrži sve s gornjeg sloja mreže (s **OSI 2**) | Označava razmak između mrežnih paketa/okvira |
| 7 *bajta* | 1 bajt | 64 *bajta* | 12 *bajta* |
| ← U k u p n o 84 b a j t a → | | | |

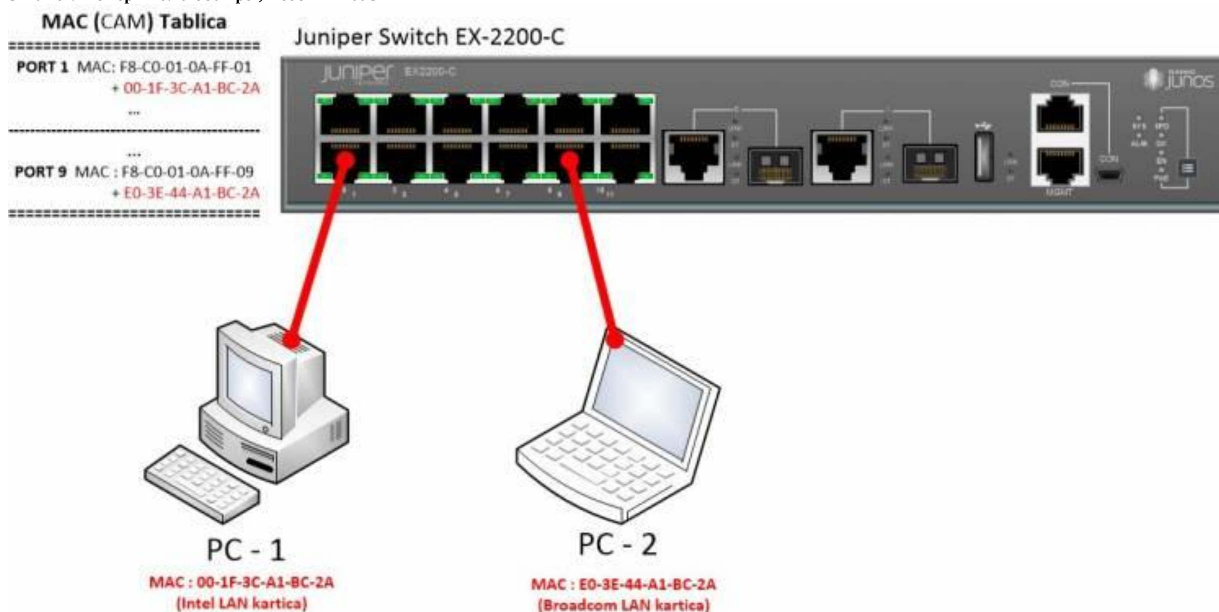
Opis polja na OSI sloju jedan (*OSI 1*) ***Ethernet*** mreža:

- `Preambula` – označava početak mrežnog okvira, veličine 7 *bajta* i sastoji se od izmjeničnog nîza nula i jedinica, kako bi se lako prepoznao početak mrežnog okvira, koji ovaj nîz i predstavlja te kako bi se sinkronizirali bîtovi sa strane primatelja.
- `SFD` – odnosno *Start Frame Delimiter* je vrijednost od jednog bajta, koja označava kraj *preambule* te indicira početak mrežnog okvira na OSI sloju dva (*OSI 2*).
- `DATA/PAYLOAD` – ovo je dio koji sadrži sve s gornjeg *OSI 2* sloja i viših slojeva, što je ugniježđeno u ovo polje.
- `Interframe gap` – označava kraj okvira i razmak između sljedećeg mrežnog okvira, koji je veličine 12 *bajta*. Tek nakon ovih 12 *bajta* moguće je poslati sljedeći mrežni okvir (koji ponovno počinje sa svojom *preambulom*).

Dakle za 46 *bajta* podataka na drugom sloju mreže (*OSI 2*) se dodaje ukupno još 18 *bajta* zaglavlja tako da mrežni okvir raste na 64 *bajta*. Takav okvir se prosljeđuje najnižem sloju (*OSI* sloj *1*), a koji nadodaje još svojih 20 *bajta* zaglavlja.
Tako se na kraju na mrežu šalje ukupno 84 *bajta* podataka za svaki pojedini mrežni okvir, navedene inicijalne veličine.

**Sada se prisjetimo kako se računala spajaju na preklopnik, kako je vidljivo na slici 161.**

**Slika 161. Preklopnik tvrtke Juniper, model EX2200C**

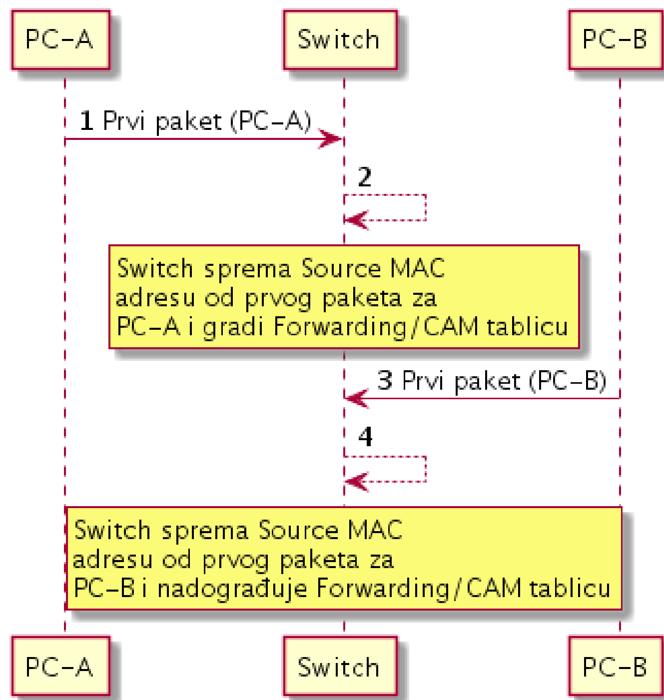



Ponovimo što se događa kod spajanja računala na preklopnik (*switch*). Zamislimo konfiguraciju kao na slici. Računalo (**PC-1**) želi komunicirati s računalom (**PC-2**), a oba računala su tek uključena. Računalo **PC-1** je spojeno na preklopnikovo mrežno sučelje (*port*) broj **1**, a računalo **PC-2** na preklopnikovo mrežno sučelje (*port*) broj **9**.
U današnjim mrežama za međusobno umrežavanja/spajanje računala, poslužitelja i ostale mrežne opreme koriste se preklopnici koji standardno rade na OSI sloju dva (*OSI 2*), dakle odluke o preklapanju (prebacivanju) paketa donose na osnovi MAC adresa zapisanih unutar svakog pojedinog mrežnog paketa.

**Što se zapravo ovdje događa u radu?**
Spajanjem bilo kojeg računala ili uređaja na neko mrežno sučelje (*port/interface*) na preklopniku, sâm preklopnik prvo mora saznati njegovu MAC adresu te ju spremiti u svoju internu tablicu koja sadrži pâr:

***`Source MAC adresa <--> mrežno sučelje na preklopniku`***

**Slika 162. Rad preklopnika (switcha).**




Sada zamislimo konfiguraciju kada računalo **A** (**PC-A**) želi komunicirati s računalom **B** (**PC-B**), a oba računala su tek uključena. Računalo **PC-A** je spojeno na preklopnikovo mrežno sučelje **1**, a računalo **PC-B** na mrežno sučelje preklopnika **2.** (slike **161** i **162**).

**1.** Računalo **A** (**PC-A**) se pokreće te šalje prvi mrežni paket prema preklopniku.

**2.** Preklopnik preuzima taj paket i čita **FCS** polje i ponovno preračunava provjerni zbroj (*Checksum*). Tek ako je s paketom sve u redu u paketu se gleda *Source MAC* adresa (to je MAC adresa od računala **A)**. Pošto tu MAC adresu nema u tablici, zapamti ju na mrežnom sučelju (*port*) na koji je spojen **PC-A** (***port*** **1**). Preklopnik sada gradi svoju tablicu:

**Port 1 – MAC adresa od PC-A**

Zatim preklopnik gleda dio paketa u kojem je odredišna MAC adresa (to je MAC od računala **PC-B**) i traži ima li negdje tu MAC adresu.

**3**. Pošto je računalo **B** (**PC-B**) upravo poslalo prvi paket on na osnovu njegove *Source MAC* adrese dodaje novi unos u svoju *Forwarding* tablicu:

**Port 2 - MAC od PC-B**

U slučaju ako preklopnik nema MAC adresu od računala **B** (**PC-B**), preklopnik bi paket od računala **A** (**PC-A**) prvo poslao na sva svoja sučelja (*portove)* i čekao odgovor od računala **B** (**PC-B**). Čim bi primio prvi odgovor (paket) od njega, na osnovi izvorišne (*source*) MAC adrese bi saznao njegovu MAC adresu te bi dodao ovaj novi unos u ***Forwarding/CAM*** tablicu.

**4**. Sada je preklopnik u mogućnosti za svaki novi mrežni paket koji treba ići od **PC-A** do **PC-B** i u obrnutom smjeru, preklapati pakete sa svog *porta* **1** na *port* **2** i obratno. ***Ovo se ponavlja za svaki paket na mreži, koji prolazi kroz preklopnik***.
Tablica na preklopnicima u kojoj se povezuje: ***MAC*** **adresa-port/** *mrežno sučelje* **(***interface***)**, se naziva ***`CAM tablica`*** (*Content Addressable Memory*) ili ponekad ***forwarding*** tablica.

**5**. Ako proširimo analizu na mrežne pakete koje zaprima računalo s druge strane, ono (**PC-B**) radi sljedeće: preuzimanjem ovog paketa, ponovno čita **FCS** polje i ponovno preračunava provjerni zbroj (*Checksum*). Ako je s paketom sve u redu, sada se čita **`ETHER TYPE`** polje, na osnovi kojeg se odlučuje što s tim paketom. Primjerice ako se radi o klasičnom **IP** paketu (**`ETHER TYPE`** ima vrijednost `0x0800`) on se upućuje dalje na obradu na više OSI slojeve unutar mrežnog stôga operativnog sustava prema aplikacijama, na osnovi podataka koji se tada konkretno nalaze na višem sloju to jest na IP sloju. Konkretno se provjerava broj odredišnog IP porta koji definira odredišnu aplikaciju (pr. port 80 označava ***HTTP*** protokol) koja treba zaprimiti i obraditi paket. Međutim da se radilo o nekoj drugoj vrsti paketa (primjerice da **`ETHER TYPE`** ima vrijednost `0x08100`) on bi se uputio dalje na obradu na OSI slojeve koji su zaduženi za baratanje s ovom vrstom mrežnih paketa. U konkretnom primjeru, da smo imali `0x08100` to bi značilo da se radi o paketu koji nosi ***VLAN*** oznaku prema ***802.1Q*** standardu. Stoga bi ovakav paket trebala preuzeti aplikacija koja radi na nižem sloju mreže i zadužena je za rad s ***802.1Q*** protokolom to jest s **VLAN**ovima.

 Za detalje oko **802.1Q** pogledajte poglavlje: **20.6.2. VLANovi odnosno virtualne lokalne mreže.**

Prisjetimo se TCP/IP komunikacije. Svaki pojedini mrežni paket mora imati (uz ostale dijelove):

- Izvorišnu i odredišnu (*Source* i *Destination*) ***MAC*** adresu.
- Izvorišnu i odredišnu (*Source* i *Destination*) ***IP*** adresu.

Pošto govorimo o standardnom preklopniku, koji je uređaj koji radi na OSI sloju dva (*OSI 2*), on gleda svaki paket i to sâmo dio s *izvorišnom (source) MAC* adresom i odredišnom *(destination)MAC* adresom. Potom na osnovu njih odrađuje preklapanje odnosno slanje mrežnih paketa s jednog (svog) mrežnog sučelja (porta) na drugo; to jest prema odredišnom računalu.

## Sumirajmo mogućnosti preklapanja (tzv. *Layer 2 switching*)

- *Bridge-ing* brzinom hardvera znan i kao "*Hardware based bridging*" tj. "*Wire Speed performance*".
  Dakle prebacivanje paketa s izvorišnog na odredišno mrežno sučelje se odvija velikom (hardverskom) brzinom.
- Kolizijska domena je na razini mrežnog sučelja (porta), dok upotrebom ***Full Duplex*** rada ona ne postoji.
- Nema mrežnog prometa između VLAN-ova (virtualnih mreža) odnosno on je nemoguć bez usmjerivača(*OSI 3* uređaja).
- ***Broadcast*** domena se proteže na sva mrežna sučelja (portove) i sve međusobno spojene preklopnike (OSI 2).

**Izvori informacija: (44),(1194),(1195),(K-6),(K-7),(K-10), Standardi: 802.2, 802.3, 802.1Q, 802.1ad.**

# 20.2. Usmjerivači koji rade na slojevima 3 i 4 te *Multilayer* preklopnici

Pogledajmo neke činjenice o usmjerivačima odnosno uređajima koji rade na OSI sloju tri i četiri (*OSI 3* i *OSI 4*):

- ***Broadcast*** domena je unutar jednog mrežnog sučelja (*porta/interface-a*).
- Procesiranje na OSI sloju tri (***Layer 3*** tj. ***OSI 3***) se obično odrađuje od strane CPU-a i samim time je veće kašnjenje od preklapanja na OSI sloju dva (*switchinga)*. Naime usmjerivači s mrežnim paketima rade u granicama milisekundi (ms), a preklopnici u granicama mikro sekundi (µs) odnosno 1.000 puta brže (ubrzanje u obradi mrežnih paketa od 100.000%).
- Visoka cijena prema broju mrežnih sučelja (*portova* to jest *interface-a*).

## Što su *Multilayer* preklopnici?

Osim standardnih preklopnika koji rade na OSI sloju dva, postoje i tzv. *Multilayer* preklopnici, odnosno *preklopnici* koji rade na više OSI slojeva:

- Na OSI sloju dva (*Layer* 2), što je klasično preklapanje (*switching*).
- Na OSI sloju tri (*Layer* 3), koje je u kategoriji usmjeravanja (*routinga*) jer se za TCP/IP protokol ovdje radi o usmjeravanju na osnovi IP adresa.
- Na OSI sloju četiri (*Layer 4*), koje je u kategoriji usmjeravanja (*routing/switchinga*), a ovdje se (za TCP/IP) radi o baratanju s TCP/UDP portovima. Sve to radi uz vrlo nisku latenciju, ako se koriste ***ASIC*** sklopovi (*čipovi*), koja je oko 1.000 puta manja od one koju imaju standardni usmjerivači (*routeri*).

Većinu operacija s mrežnim paketima *Multilayer preklopnik* koji koristi ***ASIC*** sklopove odrađuje oko 1.000 puta brže od klasičnih usmjerivača. Možemo reći da je *Multilayer preklopnik* spoj klasičnog *preklopnika* koji radi na OSI sloju dva i usmjerivača, ali koji radi brzinom dolaska mrežnih paketa odnosno na tzv. "*Wire Speed*" brzini, unutar jedne hardverske platforme. Sve operacije koje on obavlja, obavljaju i klasični *preklopnik* i standardni usmjerivač, ali sve na hardverskoj brzini unutar mikro sekundi, isključivo i sâmo ako se koriste ***ASIC sklopovi***. Upotrebom ovakvih uređaja s **ASIC** sklopovima možemo povećati performanse cijele mreže, ali uz pravilan dizajn i pravilno dimenzioniranje mreže.

**Što se događa na kojem sloju i kojom brzinom, za preklopnike koji imaju *ASIC* hardver:**

- Na OSI sloju dva sve preklapanje (*switching*) se odrađuje na osnovi MAC adresa, brzinom hardvera.
- Na OSI sloju tri operacije preklapanja, ovdje govorimo o usmjeravanju (*routing-u*), odrađuje se na osnovi IP adresa, a sve se odrađuje brzinom hardvera za razliku od usmjerivača kod kojih sve odrađuje CPU sa znatnijim kašnjenjem.
- Na OSI sloju četiri dodaje se mogućnost rada na transportnom sloju (*TCP/UDP portovi*) te možemo reći da je ovaj sloj svjestan aplikacija jer (*TCP/UDP*) *portovi* definiraju sâmu aplikaciju odnosno aplikacijski protokol.

**O čemu se radi kod ASIC sklopova (čipova)?**

ASIC (*Application Specific Integrated Circuit*) je integrirani krug/sklop (***IC***) to jest čip koji se ovisno o izvedbi i modelu preklopnika praktično nalazi na svakom njegovom mrežnom sučelju (*portu/interface-u*). Naime *ASIC* je sklop prilagođen za točno određenu uporabu, a ne namijenjen za opću namjenu odnosno opće zadaće, poput primjerice CPU-a.

Moderni *ASIC* sklopovi često imaju integrirane mikroprocesor(e), memorijske elemente, uključujući; *ROM*, *RAM*, *EEPROM*, *flash* memoriju, ali i druge elemente. Takvi ASIC sklopovi se često nazivaju ***SoC*** elementima (Engl. *System On Chip*).

Pojednostavljeno: *ASIC* sklopovi unutar sebe imaju hardverski implementirane logičke sklopove za konkretnu namjenu: zamislimo *ASIC* kao štampanu ploču s hardverskim elementima (tranzistori, kondenzatori, otpornici, diode i sl.) te drugim sklopovima koji odrađuju točno određeni zadatak. Zbog toga se taj zadatak odrađuje brzinom hardvera na kojem je izveden, odnosno maksimalnom mogućom brzinom.

Mrežni ***ASIC*** sklopovi su u mogućnosti unutar granica *mikrosekundi (**1.000 puta brže od bilo kojeg usmjerivača bez ASIC-a**)*:

- Primiti i obraditi.
- Proslijediti svaki mrežni paket, na odredišno mrežno sučelje (*port*).

Mrežne ***ASIC*** sklopove standardno koriste svi preklopnici proizvođača poput: ***Cisco***, ***Juniper*** i nekih drugih, te neki modeli proizvođača poput: ***Brocade***, ***Allied Telesin***, ***HP, Huawei*** i drugih.

S druge strane manji proizvođači poput tvrtki: ***D-Link***, ***TP Link***, ***Netgear***, ***Mikrotik*** uglavnom uopće **nemaju *ASIC*** ili imaju sâmo neki od jednostavnijih ***ASIC*** sklopova u (naj)snažnijim inačicama svojih proizvoda.

***ASIC*** može odrađivati i mnoge funkcionalnosti koje pripadaju u *OSI sloju dva* ili su na granici između *OSI* 2 i *OSI* 3, poput:

- Baratanja s ***VLAN***-ovima (virtualnim mrežama) i usmjeravanjem između ***VLAN****ova*.
- *Agregaciji/Etherchannel/Bonding/Teaming*: poput upotrebe ***LACP***, ali i sličnih protokola.

Dakle sve se može odrađivati unutar ***ASIC***-a (ako postoji); unutar granica **1.000** puta brže od uređaja koji ih nemaju, ako to konkretan ***ASIC*** podržava. Upotreba ***ASIC-a*** sa sobom nosi i veći dio razlike u konačnoj cijeni uređaja. Primjerice apsolutno svi preklopnici tvrtke ***Juniper*** imaju ugrađene snažne ***ASIC*** sklopove (od bazične serije ***EX-2300***). Veliko ubrzanje u radu ***ASIC***-a uz upotrebu vrlo brze ***SRAM*** memorije donosi i upotreba sljedećih komponenti integriranih u sâm ***ASIC***.



O prednostima ***SRAM*** u odnosu na ***DRAM*** memoriju, pogledajte poglavlje:
**12.2.1. SDRAM vrsta memorije.**

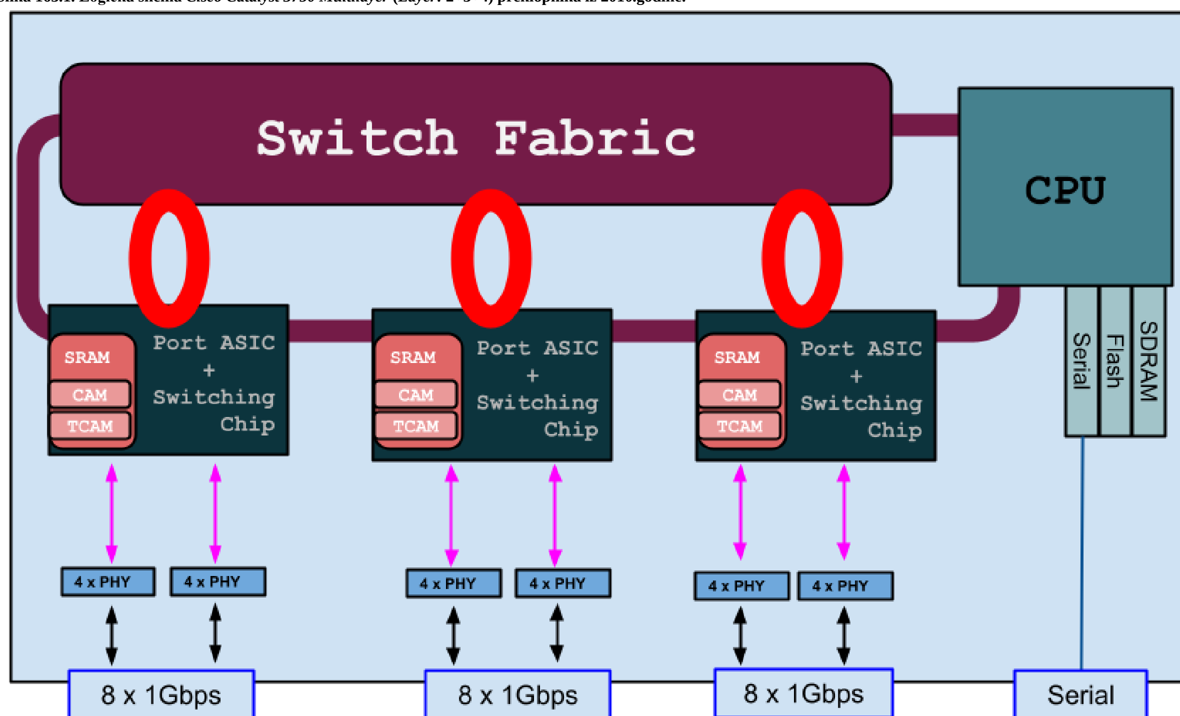
Primjerice ***Cisco** preklopnici (switchevi)* već od serije ***Catalyst 3750*** (iz 2010.g.) ili snažniji, u ***SRAM*** memoriji imaju:

- ***CAM*** memoriju (tablice) za funkcionalnosti na OSI sloju dva (*OSI 2*) odnosno za preklapanje.
- ***TCAM*** memoriju (tablice) za funkcionalnosti na OSI sloju tri (*OSI 3*) odnosno za usmjeravanje.

Zbog gore navedenih karakteristika, ovakvi ***ASIC*** sklopovi (pr. ***Cisco 3750, Juniper EX2300*** ili snažniji modeli) omogućavaju obradu sljedećih funkcija, hardverskom brzinom:

- Obradu i prosljeđivanje mrežnih okvira *(Traffic forwarding*) te usmjeravanje (obrada na *OSI 3 sloju*) paketa.
- Usmjeravanje između VLAN mreža (tzv. *Inter VLAN routing*).
- *QoS* (*Quality of Service*) te pristupne (*access)* liste *tzv. ACL lookup* odnosno funkcionalnost vatrozida/filtriranja paketa.
- Druge funkcionalnost usmjeravanja (*Route Processing)* i podršku za protokole za usmjeravanje (pr. ***OSPF/RIP****/…*).
- Podršku za *Spanning Tree protokol* i druge mrežne protokole slične namjene.

**Slika 163.1. Logička shema *Cisco Catalyst 3750 Multilayer* (*Layer*: 2+3+4) preklopnika iz 2010.godine.**




Vezano za OSI sloj tri, kod klasičnih usmjerivača koji koriste CPU za obradu i prosljeđivanje mrežnih paketa, a nemaju ***ASIC*** sklopove koji bi to ubrzali, postoje sljedeća ograničenja u propusnosti i mogućnosti obrade mrežnih paketa koja su uvjetovana:

- Brzinom procesora (CPU) te sabirnice na koju su spojene mrežne kartice te optimizacijama iste.
- Odabirom sâme mrežne kartice (i njenih funkcionalnosti) te njenog upravljačkog programa i njegovih opcija.
- Optimizacijama na razini signala prekida (**IRQ**) i načina dohvaćanja paketa (***Pooling***) kao i direktnog pristupa memoriji (***DMA***) mrežne kartice odnosno mrežnih sučelja.
- Vremenom obrade svakog mrežnog paketa.



Pogledajte primjere u poglavlju:
**10.7.1.3. CPU registri još detaljnije**.

- Optimizacijama na razini mrežnog stôga operativnog sustava i svih mrežnih slojeva operativnog sustava.
- I u konačnici sâmu mogućnost obrade mrežnih paketa.



Pogledajte poglavlje:
**20.4. *Gbps* i *Mpps* – u čemu je veza**.

**Izvori informacija: (1017), (K-6),(K-7).**

## 20.2.1. Dizajn preklopnika

**Slijedi napredna cjelina!**
U ovoj cjelini, bez obzira na već navedeni dizajn preklopnika koji smo vidjeli na slici 163.1., analizirat ćemo dizajn preklopnika od početka, kako bismo bolje razumjeli problematiku u preklapanju odnosno usmjeravanju mrežnih paketa.

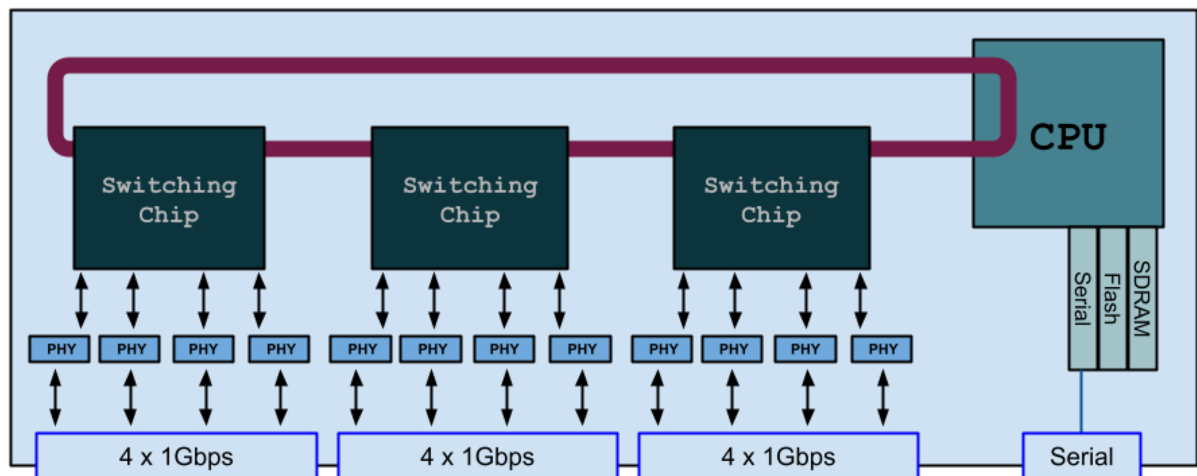
Naime u konačnom dizajnu preklopnika, u pravilu postoje dva pristupa:
**1.** Preklopnik koji koristi *CPU* u kombinaciji sa ***Switching*** *sklopom (chipom)* ili
**2.** Preklopnik koji koristi *CPU* u kombinaciji sa ***Switching*** *sklopom (chipom)* te ***ASIC*** *sklopom (chipom)*.

Već kod standardnih preklopnika koji odluke o preklapanju (prosljeđivanju mrežnih paketa) donose na OSI sloju dva (OSI 2), funkcionalnosti koje su implementirane u ***Switching*** *sklopove* nisu dostatne za sve operacije koje jedan napredni preklopnik treba podržavati. U njih su uglavnom implementirane najrudimentarnije stvari pa dobar dio mrežnih protokola mora odrađivati centralni procesor (CPU) samog preklopnika.

Naime *Switching* sklop je integrirani sklop koji u baziĉnoj implementaciji odrađuje osnovne funkcionalnosti, poput:
- Prihvata mrežnih paketa i analize zaglavlja svakog paketa.
- Provjere ispravnosti paketa: provjerom razlike zaprimljenog ***FCS*** polja te izračuna novog ***FCS*** provjernog zbroja.
- Provjere **MAC** adresa paketa i spremanje istih za izgradnju **CAM** tablice (tablice preklapanja).
- Konačnog preklapanja paketa, na osnovi **MAC** adresa i odredišnog mrežnog sučelja (iz **CAM** tablice).

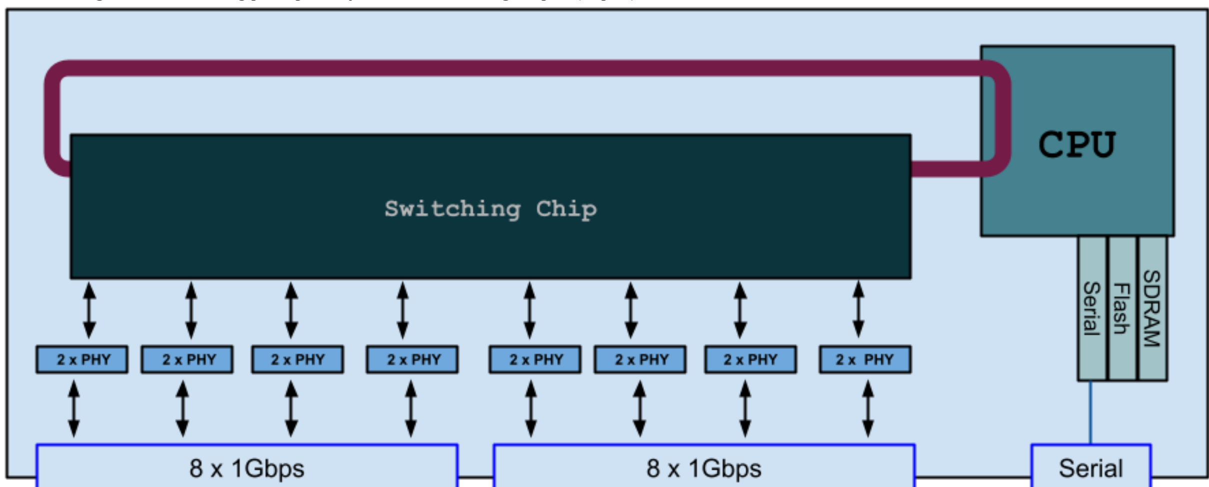
Ovdje je važnost u odabiru dobrog *Switching sklopa* krucijalna jer određeni dio njih već u nižoj srednjoj klasi nudi dobar dio funkcionalnosti implementiranih u sam sklop te se samim time rasterećuje centralni procesor koji ionako nije sposoban za obavljanje tih zadataka čak i na gigabitnim brzinama. Pogledajmo nekoliko varijanti pristupa pri dizajnu preklopnika.

**Slika 163.2. Logička shema *klasičnog* preklopnika koji ima više switching sklopova (*chipova*).**




Dizajn koji je vidljiv na slici 163.2., koriste mnogi proizvođači preklopnika jer drastično snižava cijenu. Osnovni problem u ovom dizajnu je sporost sabirnice koja povezuje *Switching* sklopove. To znači da je propusnost (brzina) komunikacije zadovoljavajuća samo, ako se odvija unutar jednog *switching* sklopa. Naprednija inačica dizajna bi bila upotreba jednog velikog *switching* sklopa, na kojem se nalaze sva mrežna sučelja, jer tada više nema sabirnice kao uskog grla (slika 163.3).

**Slika 163.3. Logička shema *klasičnog* preklopnika s jednim većim switching sklopom (chipom).**



Međutim mana ovog dizajna je problematika obrade mrežnih paketa. Naime *switching* sklopovi imaju ograničene mogućnosti rada s mrežnim paketima, pa sve osim bazičnih funkcija mora i dalje odrađivati centralni procesor (CPU), koji je ograničen propusnosti sabirnice prema *switching* sklopu, ali i svojom sposobnosti odnosno snagom procesiranja, koja je u pravilu iznimno mala, već i za gigabitne brzine. Vezano za procesor preklopnika, uglavnom se tu radi o odabiru primjerice **ARM** procesora s malim brojem jezgri koji nikako nije u mogućnosti obrađivati (procesirati) dovoljan broj mrežnih paketa (u sekundi).

Primjerice za 1Gbps propusnost na 24 mreža sučelja (*porta*), za 64.bajtne mrežne pakete, potrebno je moći procesirati: 24 x 1.488Mpps = 35.712 milijuna paketa u sekundi, što je, ako to odrađuje procesor, ekstremno zahtjevno i praktično neizvedivo.



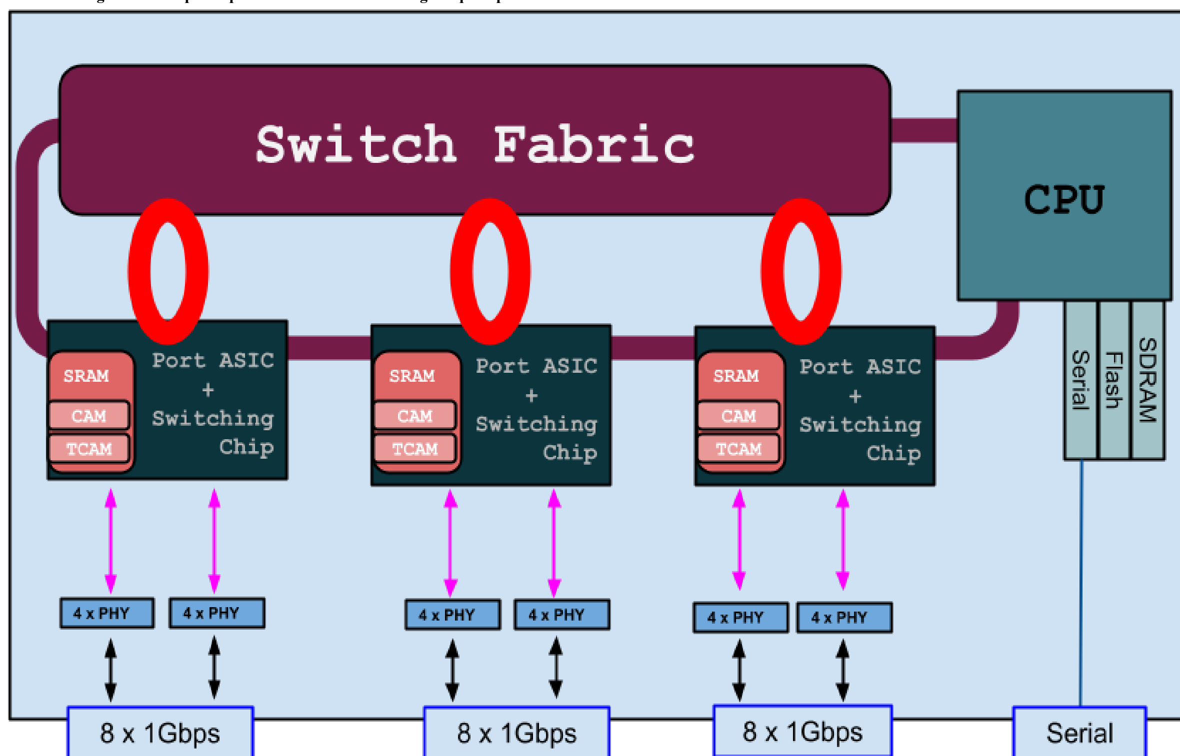
Vezano za resurse računala potrebne za prihvaćanje i obradu mrežnih paketa, pogledajte poglavlje:
**9.3.1. Context switching (**i to cjelinu „**Vrijeme obrade**“**).**



Dodatno, pogledajte i poglavlje:
**10.7.1.3 CPU registri još detaljnije** i to cjelinu: „**Vrijeme obrade mrežnih paketa**“.

Zbog toga je potreban drugačiji pristup, upotrebom specijaliziranih **ASIC** sklopova, koji su u mogućnosti rasteretiti centralni procesor (CPU), te na sebe preuzeti zadaće prihvaćanja, obrade i prosljeđivanja mrežnih paketa ekstremno velikim brzinama.

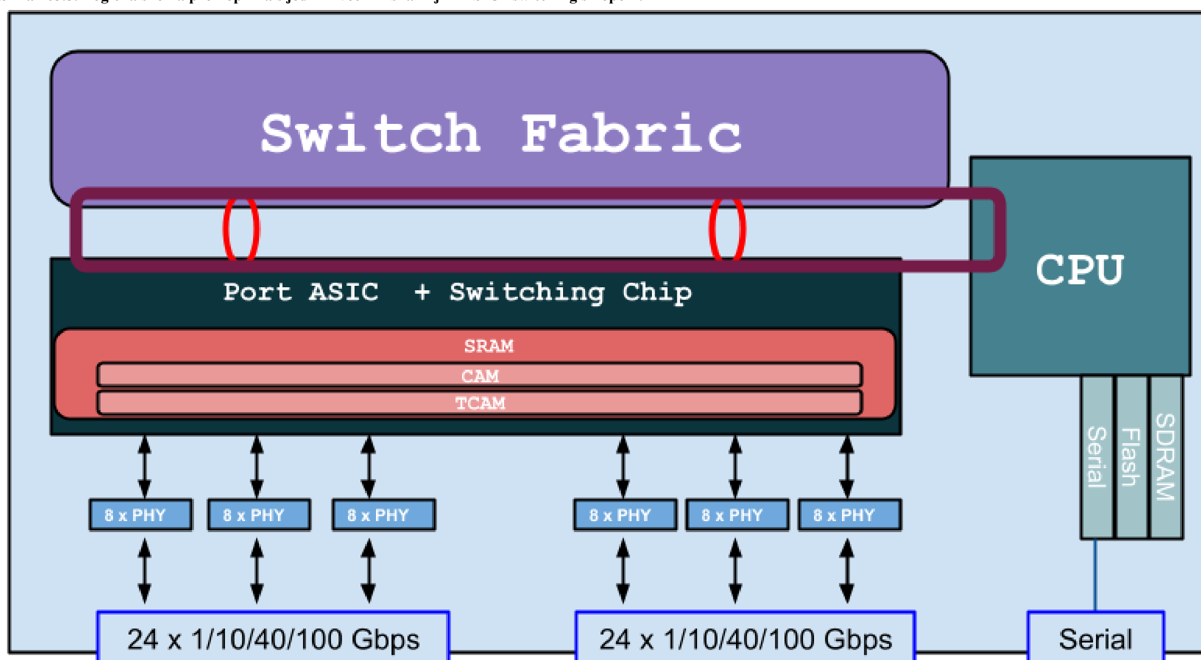
**Slika 163.4. Logička shema preklopnika s više ASIC i switching sklopova povezanih sabirnicom.**




Prva varijanta **ASIC** dizajna (slika 163.4) je upotrebom više manjih ***ASIC*** sklopova sa ***Switching*** sklopovima, koji su zaduženi za rad nekolicine mrežnih sučelja (***portova***). Ovakav dizajn je nešto jeftiniji, ali mu je mana propusnost sabirnice, pa će u komunikaciji između mrežnih sučelja (portova) koja su na različitim ***ASIC+Switching*** sklopovima, dolaziti do zagušenja, jer ta sabirnica u pravilu nije dovoljno velike propusnosti.

Stoga se kod skupljih uređaja koristi sljedeći dizajn (slika 163.4).

Slika 163.5. Logička shema preklopnika s jednim većim i snažnijim ASIC i switching sklopom.




Kod ovakvog dizajna nema slabosti jer su sva mrežna sučelja spojena na jedan ***ASIC+Switching*** sklop (slika 163.4), vrlo visoke propusnosti. Jedina mana je vrlo visoka cijena ovakvih sklopova i cijelog uređaja u konačnici.

**Izvor informacija: (11),(12),(13),(14),(49),(53.1),(53.2),(1017),(K-7).**

# 20.3. Ne blokirajući i blokirajući dizajn preklopnika

Na dizajnu Cisco *3750* preklopnika (iz 2010.g.), slika 163.1, vidljivo je kako ovaj ***ASIC*** ima i vrlo brzu **SRAM** memoriju za **CAM** i **TCAM**, te dio **SRAM** memorije za širu upotrebu unutar ***ASIC*** sklopa te kako je ona povezan s centralnom sabirnicom (***Switch Fabric-om***). *Switch Fabric* je zapravo posebna vrsta ***ASICa*** na čiju sabirnicu se spajaju svi ***ASIC*** sklopovi mrežnih sučelja i CPU, ali i takozvana *Stack* sučelja (ako ih uređaj ima ovako implementirane), a preko kojih se ova vrsta *preklopnika (Cisco 3750)* može povezivati u prsten (*Stack*).

***Switch Fabric*** odnosno ovakva sabirnica je ekstremno velike brzine i propusnosti.
Konkretni *Switch fabric* preklopnika *Cisco Catalyst 3750* ima propusnost od 128 Gbps.

Dodatno, u ovom konkretnom dizajnu, postoje dvije vrste sabirnica:

- *Između* ***Switch Fabrica*** i svakog pojedinog sučelja (*porta)* **ASIC** sklopa. Kod ovog modela se radi o prstenastoj sabirnici propusnosti 32 Gbps prema svakom *ASIC sklopu.*
- Između ***Switch Fabrica*** i svakog pojedinog sučelja (*porta)* **ASIC** sklopa te *CPUa*. Ovdje se isto radi o prstenastoj sabirnici.

Ako pogledamo donji dio slike 163.1, svaki od ovakvih ***ASIC*** *port sklopova* podržava spajanje do 8x1Gbps **PHY** (*Physical Layer*) sklopova koji su zaduženi za sâm mrežni medij (*interface*), a obično je to bakrena veza odnosno RJ-45 konektor.

***PHY*** sklopovi mogu biti za bakar (RJ-45 konektor) ili optiku (bilo koja vrsta utikača za optiku) ili ***SFP*** port u koji se uključuje ***SFP*** adapter koji može biti ili za "bakar" ili za optiku. Dakle **PHY** *sklop* je u konačnici *integrirani sklop* koji sve mrežne pakete pretvara u električne signale ili optičke impulse za slanje na mrežu, odnosno na sâm mrežni medij.

**(SRAM) memorija**
**CAM** dio (*Content Addressable Memory*) je zapravo tablica u **SRAM** memoriji u kojoj se zapisuju i pretražuju poveznice: ***MAC adresa + port*** (sučelje) na preklopniku, na osnovu koje se radi preklapanje na OSI sloju dva (*OSI 2*).
**TCAM** dio (*Ternary Content Addressable Memory*) je zapravo tablica u **SRAM** memoriji s poveznicama na višim OSI slojevima (*OSI 3*). Dakle ovdje se spremaju tablice usmjeravanja (*routing*) i drugi meta podaci vezani za OSI sloj tri (*OSI 3*).

Dakle **CAM** i **TCAM** tablice se nalaze u posebnoj ekstremno brzoj **SRAM** memoriji **ASIC** *sklopa* ili u starijoj generaciji uređaja kao zaseban *integrirani sklop (**SRAM** čip)*, a koji omogućava nevjerojatno brzi pristup i pretraživanje navedenih tablica. Zbog toga imamo i ekstremno brze funkcionalnosti na OSI slojevima dva (preklapanje/*switching*) i OSI sloju tri (usmjeravanje/*routing*). Naime osim velike brzine i propusnosti ***ASIC** sklopova* i *Switch Fabric-a* važno je znati kako je vrlo važna i brzina RAM memorije, stoga se i koriste **SRAM** memorije velike propusnosti i brzine rada. Bez obzira jesu li ove vrste RAM memorije integrirane u sam *ASIC* ili su izvedene kao vanjske komponente, one su znatno većih brzina od svih današnjih RAM memorija, poput DDR3 ili DDR4 (ili čak DDR5).

Tako je primjerice propusnost DDR4 memorije oko 205 Gbps. Kod vrlo velikih brzina mreža, s mrežnim sučeljima koja rade na 10Gbps, 50Gbps, 100Gbps ili više, ovo postaje veliki problem. Neki proizvođači poput tvrtke ***Juniper Networks*** za svoje najsnažnije serije preklopnika, uz razvoj vrlo snažnih ***ASIC** sklopova* razvili su posebne vrste ekstremno brzih odnosno propusnih **SRAM** memorija u zasebnim integriranim sklopovima. ***Juniper Networks*** naziva ove *sklopove* ***Hybrid Memory Cube***. Ovakvom kombinacijom ***Juniper Q5 ASIC*** i vanjske RAM memorije (*Hybrid Memory Cube*) postiže se propusnost od i prema ovoj memoriji od 1 Tbps (1.000 Gbps) između svakog ***Q5 ASICa*** i njegove ***Hybrid Memory Cube*** RAM memorije.
S time se dobiva ogromna propusnost jer se svaki par ***Q5 ASICa + Hybrid Memory Cube*** obično spaja na ***Switch Fabric*** , koji je također ekstremno visoke propusnosti.
Tvrtka *Juniper Networks* razvila je **Q5** ASIC čipove s mogućnostima dodatne analitike, dok su **TRIO** i **EXPRESS ASIC** čipovi namijenjeni za različite svrhe. *TRIO ASIC* čipovi su optimizirani za MX serije usmjerivača, dok su EXPRESS ASIC čipovi namijenjeni za PTX (*Packet Transport Routers*) seriju usmjerivača za mreže visokih propusnosti.

Pogledajmo usporednu tablicu navedenih ASIC čipova tvrtke *Juniper Networks:* TRIO i EXPRESS, različitih generacija:

| **ASIC** | **TRIO 1** | **TRIO 2** | **TRIO 3** | **TRIO 4** | **TRIO 5** | **TRIO 6** |
|---|---|---|---|---|---|---|
| **Propusnost** | 120 Gbps | 240 Gbps | 480 Gbps | 1 Tbps | 1,6 Tbps | 9,6 Tbps |

| **ASIC** | **EXPRESS 1** | **EXPRESS 2** | **EXPRESS 3** | **EXPRESS 4** | **EXPRESS 5** |
|---|---|---|---|---|---|
| **Propusnost** | 60 Gbps | 120 Gbps | 9,6 Tbps | 14,4 Tbps | 28,8 Tbps |


Kod nekih proizvođača postoje namjerni ili slučajni propusti odnosno odabir takvog dizajna preklopnika kod kojega preklopnik nije u mogućnosti isporučivati mrežne pakete u slučajevima kada sva njegova mreža sučelja generiraju vršni ili maksimalan mogući mrežni promet.

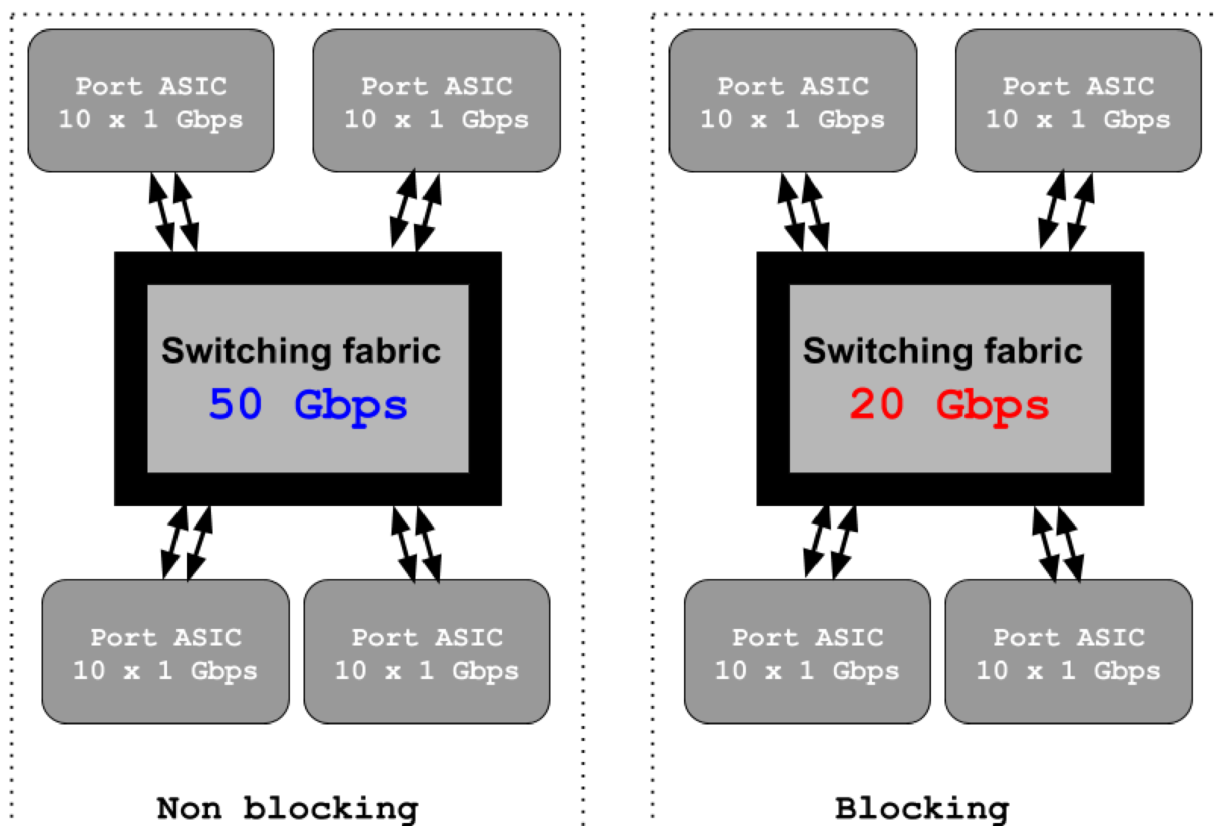
Zamislimo dizajn preklopnika u kojem je više **ASIC** sklopova povezano među sobom sa sabirnicom, kako smo vidjeli na slici **163.4**. Kod ovakvog dizajna preklopnika, za ono što bi bio ne blokirajući odnosno ***Non blocking*** način rada, minimalno isto toliko mrežnih sučelja koliko ih svaki ***ASIC*** sklop propušta prema van (na mrežna sučelja) bi morala biti i veza prema svakom susjednom ***ASICu*** i ***Switch fabricu*** odnosno unutarnjoj sabirnici unutar preklopnika.

Isto tako, čak i kada imamo dizajn preklopnika sa samo jednim **ASIC** i **switching** sklopom na koji su spojena sva mrežna sučelja (*portovi*) preklopnika, kako je vidljivo na slici **163.5**, unutarnja moć obrade i procesiranja mrežnih paketa **ASIC** sklopa mora biti jednaka zbroju propusnosti svih vanjskih mrežnih sučelja cijelog uređaja (preklopnika).
To isto vrijedi i za vezu *switching* sklopova, u slučaju kada se ne koristi ***ASIC*** već sâmo ***switching*** sklopovi, u dizajnu kada je više *switching* sklopova povezano međusobno određenom vrstom sabirnice, kao što je vidljivo na slici **163.2**.


Vezano problematiku propusnosti sabirnice, pogledajte poglavlje:
**10.1.1. Matična ploča, CPU, *chipset* i sabirnica(e).**


Dodatno pogledajte i poglavlje:
**10.7.1.3 CPU registri još detaljnije** i to cjelinu: „**Vrijeme obrade mrežnih paketa**“

Slika 164. Modeli rada preklopnika: blokirajući i ne blokirajući




Dakle ***Non blocking*** odnosno ne blokirajući dizajn se odnose na ukupnu unutarnju propusnost preklopnika, odnosno je li on u mogućnosti osigurati nesmetan mrežni promet odnosno komunikaciju bez blokiranja, usporavanja ili zaustavljanja, ako apsolutno sva njegova vanjska *mrežna sučelja (portovi)* rade na maksimalnoj mogućoj brzini rada.

To znači da, ako imamo preklopnik s 24 mrežna sučelja s brzinom od 1Gbps te su na njih spojena 24 računala koja generiraju maksimalan mogući promet (1Gbps *Full duplex* svako), da bi preklopnik morao moći obrađivati mrežne pakete ukupnom propusnošću 24 x 1Gbps *Full Duplex* bez ikakvog usporavanja ili blokiranja.
To za taj preklopnik i to konkretno za *Full Duplex* načina rada znači: 48Gbps propusnosti u dvosmjernoj komunikaciji, bez blokiranja ili usporavanja rada.



Za detalje oko teorije, mjerenja i propusnosti mreže, pogledajte sljedeća poglavlja:
**20.4. *Gbps* i *Mpps* – u čemu je veza.**
**20.4.1. Mjerenja propusnosti bez ASIC sklopova.**
**20.4.2. Mjerenje propusnosti sa ASIC sklopovima.**

Lijevi (***Non blocking***) dizajn je takav da može podnijeti puno opterećenje na svim vanjskim mrežnim sučeljima (*portovima*) uređaja, bez usporavanja rada, odbacivanja mrežnih paketa ili bilo kakvih drugih problema u radu.

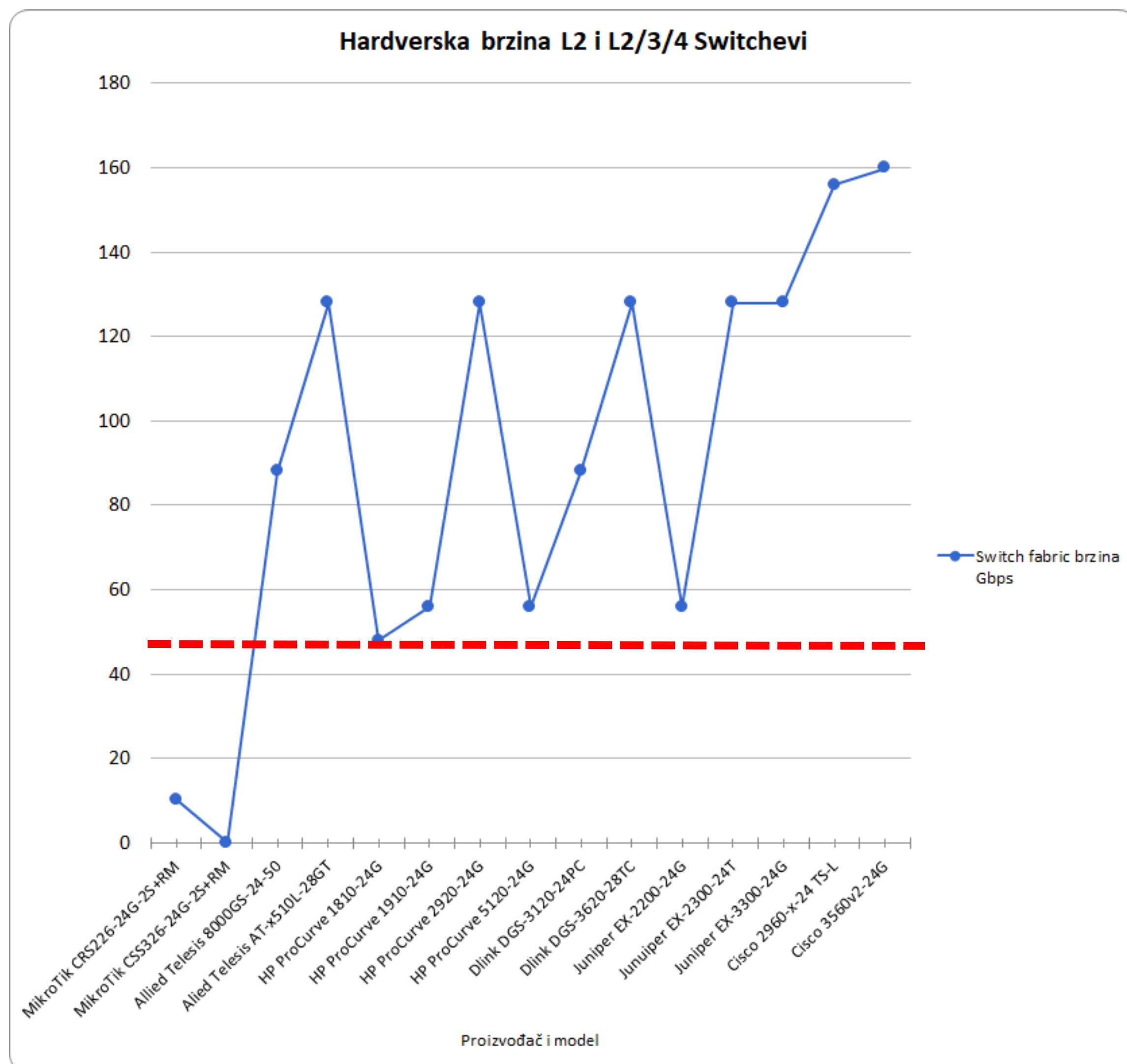
S druge strane desni (slika 164.) odnosno takozvani ***Blocking*** dizajn je taj koji ne može podnijeti puno opterećenje na svim mrežnim sučeljima jer je njegova unutarnja propusnost znatno manja od zbroja propusnosti svih njegovih vanjskih mrežnih sučelja. Naime kod ***Blocking*** dizajna u konačnici nije važno je li uzrok slabije propusnosti loš odabir **switching** sklopova, **ASIC** sklopova, sporije sabirnice koja ih povezuje ili odabir navedenih sklopova čija snaga procesiranja (obrade) mrežnih paketa nije dovoljna, rezultat je isti.

**Izvori informacija:(45),(46),(1018),(1570),(1571),(1572),(1573),(1574),(1575),(K-7).**

# 20.4. *Gbps* i *Mpps* – u čemu je veza

Oznake ***Gbps*** ili ***Mpps*** možemo najčešće vidjeti na deklaraciji proizvođača mrežnih uređaja i to najčešće preklopnika (*switcheva*), ali i usmjerivača (*routera*). Međutim iste oznake vidimo i primjerice na raznim vatrozidima (*firewallima*).
Oznaka ***Gbps*** (gigabita u sekundi) označava ukupnu propusnost uređaja, a obično preklopnika, koju dijele sva njegova mrežna sučelja. U slučaju propusnosti svih mrežnih sučelja unutar uređaja sumirano, govorimo o takozvanoj *Fabric*/*Bus* brzini odnosno propusnosti unutarnje sabirnice sâmog uređaja. Minimalna brzina koju bi uređaj morao imati je jednaka zbroju propusnosti (brzina) svih mrežnih sučelja uređaja. Iako ovdje govorimo o uređajima generalno, to je primjenjivo i mjerljivo i na računalima koja se koriste kao preklopnici odnosno kao mrežni mostovi (*bridgevi*), usmjerivači ili kao neki drugi specijalizirani uređaji. U slučaju kada primjerice imamo preklopnik s 24 x 1Gbps mrežnih sučelja, pošto uređaj na gigabitnoj brzini mora moći raditi u *Full Duplex* načinu rada, to znači da za 24 x 1Gbps mora moći podnijeti: 24 x 2Gbps = 48Gbps ukupne propusnosti. Pogledajmo karakteristike nekoliko modela preklopnika raznih proizvođača dostupnih kod nas, koje smo uzeli u razmatranje, gledajući specifikacije proizvođača vezane za (unutarnju) propusnost cijelog uređaja (slika 165).

**Slika 165. Rezultati mjerenja unutarnje propusnosti preklopnika.**




**Što znači kada uređaj sa 24x1Gbps mrežnim sučeljima mora moći obrađivati pakete propusnošću od 48Gbps?**
Oznaka ***Gbps*** (gigabita u sekundi) označava propusnost odnosno kolokvijalno rečeno brzinu pojedinog mrežnog sučelja, a u slučaju sa slike 165 gore, označava ukupnu propusnost preklopnika koju dijele sva njegova mrežna sučelja zajedno.
Na slici 165 je vidljivo kako su neki uređaji dizajnirani tako da ne mogu isporučiti punu propusnost kod većeg opterećenja. Dakle koriste ***Blocking*** dizajn o kojem smo pričali u prethodnoj cjelini.
S druge strane, na slici 165, svi uređaji koji mogu isporučiti 48Gbps ili više, koriste ***Non-Blocking*** dizajn uređaja.

Dodatno vrlo važno pitanje je i koliko stvarno mrežnih paketa u jedinici vremena (sekundi) može obraditi određeni preklopnik ili mrežni uređaj, a pogotovo s kojom veličinom mrežnih paketa. Odgovor na ovo pitanje slijedi.

**Prvo malo računanja**
Pretvorimo brzinu od 1 Gbps u bitove u sekundi: 1Gbps = 1.000.000.000 ***bps.***
Pretvorimo bitove u bajte odnosno bajte u sekundi (***Bps***); ne zaboravimo kako je 8 bitova (b) = 1 bajt (B).
Mi moramo sve pretvoriti u ***Bps*** pa dobivamo:

$$\frac{1.000.000.000}{8} = 125.000.000\ \boldsymbol{Bps}$$

Dakle za brzinu od 1Gbps imamo protok od 125 milijuna *bajta* u sekundi.
Prvo ćemo analizirati najmanje mrežne pakete odnosno mrežne okvire veličine **64** bajta.

**Za sve Ethernet okvire vrijedi sljedeće:**

- **18** bajta je zaglavlje okvira (*Engl Frame Header*) - ovo dodaje mrežni sloj dva (OSI sloj 2).
- **20** bajta je preambula i dio za početak okvira (*Engl. Start of frame delimeter* - **SFD**) te na kraju i standardni minimalni razmak između mrežnih okvira (Engl. *Interframe Gap* - ***IFG***) - ovo dodaje mrežni sloj jedan (OSI sloj 1).

Prema **Ethernet standardu** najmanji paket je veličine 46 *bajta*, ali uz njega ide i **18** *bajta* zaglavlja (*Frame Header)* te još ukupno **20** *bajta* ostalih dijelova s čime konačni paket poslan na mrežu raste na minimalno **84** *bajta*.
Naime ovdje govorimo o mrežnim paketima odnosno okvirima na drugom sloju (OSI sloj 2) mreže, to jest o njihovoj veličini koja se odnosi na ovaj sloj.
**I sada za mrežne pakete veličine 64 bajta dobivamo:**

$$\frac{125.000.000}{46+18+20} = 1.488.095 \text{ pps (paketa u sekundi)} = 1{,}488 \text{ Mpps} = 1{,}488 \text{ mlijuna paketa u sekundi}$$

Dakle za brzinu mreže od 1Gbps, odnosno 1 x 1Gbps, za mrežne okvire veličine **64** bajta, potrebno je moći procesirati odnosno obraditi **1,488** milijuna mrežnih paketa u sekundi.
Pogledajmo kako to izgleda za druge brzine, za istu veličinu (**64** bajta) mrežnih paketa odnosno mrežnih okvira:

| **Brzina** (Propusnost) | **Broj mrežnih paketa u sekundi** |
|---|---|
| 1 **Gbps** | 1,488 **Mpps** |
| 10 **Gbps** | 14,88 **Mpps** |
| 40 **Gbps** | 59,2 **Mpps** |
| 100 **Gbps** | 148,8 **Mpps** |

**Što je s većim mrežnim okvirima od primjerice 1500 bajta**?
Naime i ovdje kod 1500 bajtnih mrežnih okvira mislimo na 1500 *bajta* sadržaja (Engl. *Payload).*
Dakle dio korisnog dijela mrežnog okvira koji sadrži nama korisne podatke je veličine 1500 *bajta*.

To znači kako je ukupna veličina ovakvog mrežnog paketa sljedeća:
*1500 bajta podataka + 18 bajta zaglavlja + 20 bajta ostali već objašnjeni dio = 1538 bajta*

**To znači da se za 1500 bajta podataka, na mrežu šalje mrežni okvir veličine 1538 *bajta***
Pretvorimo bajte u bitove:
*1538 bajta x 8 = 12.304* bitova po paketu, što znači kako jedan ovakav paket sadrži *12.304* bitova.

Za 1Gbps brzinu, pretvorimo bitove u sekundi (**bps**) u bajte u sekundi (**Bps**):
*bps = 1.000.000.000 / 8 = 125.000.000* ***Bps***

**I sada dobivamo:**

$$\frac{125.000.000}{1500+18+20} = 81.274\ \boldsymbol{pps} = \boldsymbol{81{,}274\ kpps} = 81\ tisuća\ paketa\ u\ sekundi$$

Dakle za brzinu mreže od 1Gbps odnosno 1 x 1Gbps, za mrežne okvire veličine 1500 bajta potrebno je moći procesirati odnosno obraditi **81** tisuću (81 ***kpps***) mrežnih paketa u sekundi.

Pogledajmo kako to izgleda i za druge brzine, za veličinu mrežnih paketa (okvira) od 1500 bajta:

| **Brzina** (Propusnost) | **Broj mrežnih paketa u sekundi** |
|---|---|
| 1 Gbps | 81 kpps (81.000 p/s) |
| 10 Gbps | 812 kpps (812.000 p/s) |
| 40 Gbps | 3,25 Mpps (3.250.000 p/s) |
| 100 Gbps | 8,12 Mpps (8.120.000 p/s) |

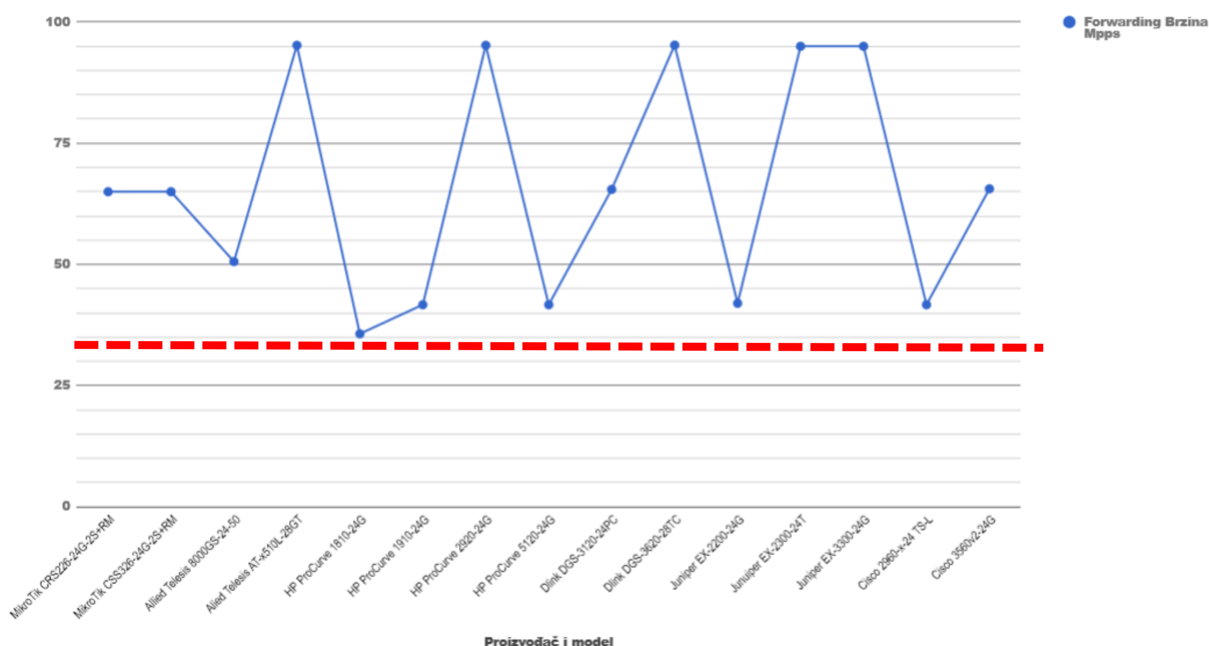
**Zaključak:**
Kao što je i bilo za očekivati, najzahtjevnije je procesirati male mrežne pakete od 64 bajta; za ukupnu propusnost od 1Gbps potrebno ih je obrađivati **1,48** milijuna u sekundi, kako bi se zadržala navedena propusnost. S druge strane za najveće mrežne pakete od 1500 bajta, za ukupnu propusnost od 1Gbps, potrebno ih je moći obrađivati samo **81** tisuću (81 ***kpps***) u sekundi.



Prisjetite se poglavlja:
**20.1. Oblik mrežnih okvira na OSI slojevima 2 i 1.**

Pogledajmo sada karakteristike istih modela preklopnika koje smo već razmatrali, prema mogućnosti obrade mrežnih paketa veličine 64 bajta. Napomena: ovdje je donja granica "***ozbiljnosti***" odnosno upotrebe koja se očekuje od preklopnika sa 24 x 1Gbps mrežnih sučelja (***Non-blocking***), s punom propusnošću minimalno ***36Mpps*** odnosno **36** milijuna paketa u sekundi.

**Slika 166. Rezultati mjerenja (vanjske) propusnosti preklopnika.**




Sve preko toga (**36 *Mpps***) je nepotrebno osim, ako uređaj ima dodatnih SFP/SFP+ ili sličnih sučelja: svaki dodatni 1Gbps dodaje potrebu za povećanjem propusnosti od 1,488 *Mpps* odnosno 1,488 milijuna paketa u sekundi. Ili ako se koristi i kao platforma za snažnije modele preklopnika, koji recimo imaju dvostruko veći broj mrežnih sučelja ili dodatna SFP ili SFP+ mrežna sučelja. *Mjerenja se obično rade prema standardnu RFC 2544 koji definira metode i načine testiranja.*

**Što to znači za preklopnike iz ove kategorije koji nisu u stanju isporučiti minimalno 36Mpps za 64 bajtne pakete?**
Preklopnik koji je primjerice u mogućnosti isporučiti samo do 10Mpps, punom brzinom može opskrbiti maksimalno do 7 x 1Gbps. Dakle ako imamo do maksimalno sedam (7) mrežnih uređaja (računala/poslužitelja i sl.) spojenih na preklopnik, sve će raditi dovoljno brzo, pri punoj brzini, ali već spajanjem osmog uređaja dolazi do usporavanja. Naime platili smo preklopnik s 24 x 1Gbps, a dobili smo preklopnik sa sedam mrežnih sučelja koji ima dodatnih 17 mrežnih sučelja koja ne mogu podnijeti punu propusnost odnosno brzinu rada.

U slučajevima kada imamo veća odstupanja između rezultata mjerenja propusnosti uređaja odnosno njegove sabirnice (slika 165) i mjerenja mogućnosti obrade mrežnih paketa (slika 166) to nam uglavnom govori da konkretni uređaj hardverski ne može isporučiti odnosno obraditi onu količinu mrežnih paketa, koja je obično navedena kao ***Mpps*** (ili ***kpps***) ili da rezultati mjerenja ovise s kojim testovima su napravljena mjerenja (u stvarnosti).
U konačnici s takvim uređajima obično treba biti vrlo oprezan kod odabira odnosno kupnje.

Ako koristimo usmjerivač, vatrozid ili slični uređaj odnosno računala ili uređaje koji nemaju specijalizirane ***ASIC*** sklopove, poput primjerice primjene Linuxa/Unixa[610] kada ga koristimo kao usmjerivač, vatrozid i slično, u određenim primjenama možemo doći do sličnih ograničenja.

Naime važna stvar kod odabira mrežnog uređaja, a pogotovo ako govorimo o usmjerivačima odnosno uređajima koji rade na OSI slojevima tri i četiri (OSI3/4), a nemaju ***ASIC*** već svo potrebno procesiranje odrađuje **CPU**, s problematikom koju smo spomenuli u poglavlju **20.2.**, potrebno je naglasiti sljedeće:

- Uređaji koji će raditi kao klasični usmjerivači, odnosno sâmo kao veza između ***LAN*** i ***WAN*** mreže, naročito prema ***WAN*** mreži (internetu), većinom rade s većim mrežnim paketima (***MTU***=**1300+**) pa su efektivno nešto manje opterećeni nego sljedeća kategorija uređaja. Mada i ovdje moramo uzeti u obzir stvarni promet (pogledajte mjerenja **166.1**).



Prisjetite se poglavlja:
**20.2. Usmjerivači koji rade na slojevima 3 i 4 te *Multilayer* preklopnici.**

- Uređaji na lokalnoj mreži (***LAN***) koji su zaduženi za usmjeravanje (OSI3/4) između više lokalnih mreža; primjerice između ***VLAN*** mreža unutar lokalne (***LAN***) mreže, posebna su priča:
    - U ***LAN*** mrežama se očekuju eksponencijalno veće brzine rada nego prema ***WAN*** mrežama (internetu). Stoga ovakvi uređaji (usmjerivači ili ***Multilayer*** preklopnici) moraju biti ekstremno brzi u obradi mrežnih paketa. Klasični usmjerivači bez ***ASIC***-a imaju kašnjenje odnosno vrijeme obrade mrežnih paketa u rangu mili sekundi (ms), dok oni koji koriste ***ASIC*** to sve odrađuju unutar granica mikro sekundi (µs). Dakle u pravilu i u praksi 1.000 puta odnosno 100.000% brže. To naravno utječe na rad lokalne mreže. Svakako možemo zaključiti da je već na gigabitnim brzinama, ako nam je mreža segmentirana upotrebom ***VLAN***-ova na preklopnicima, jedini odabir koji u potpunosti zadovoljava ovakve brzine: upotreba ili usmjerivača koji pomoću ***ASIC*** sklopova odrađuju usmjeravanje između ***VLAN*** mreža ili upotreba takozvanih *Multilayer* preklopnika koji za istu operaciju također moraju koristiti ***ASIC***. U oba slučaja upotrebe ***ASIC*** sklopova podrazumijeva se upotreba takvih ***ASIC*** sklopova koji u potpunosti moraju moći odraditi sve radnje vezane za usmjeravanje, bez uplitanja CPU-a.
    **Međutim to nije slučaj za sve *ASIC* sklopove, pa pri odabiru uređaja ponovno treba biti vrlo oprezan!**.

Ne zaboravimo da u lokalnim (***LAN***) mrežama imamo i veliki broj paketa različitih (***MTU***) veličina; jer ovdje nije nužno da je većina mrežnih paketa maksimalne (***MTU***) veličine koju naša ***LAN*** mreža dopušta.
Naravno sve ovisi o konkretnoj mreži.

- U nekim slučajevima imamo mreže u kojima se većinom nalaze osobna računala koja se vrlo često spajaju na Internet za razne potrebe: surfanje webom, čitanje članaka i slično. Kao i zbog spajanja na određene mrežne servise; primjerice za spremanje ili dohvaćanje programskog kôda, poput*****: ***Github***, ***Gitlab***, ***Bitbucket***, ***SourceForge*** i drugih ili sličnih namjena, mogu, barem za spajanje prema centralnom usmjerivaču koristiti veće mrežne pakete. Dakle pakete koji su obično veličine (**MTU**) oko 1500 bajta. Ipak i u ovakvom radu ne zaboravimo na ***TCP*** protokol i njegovu potrebu potvrđivanja zaprimljenih paketa, koji će činiti veći dio vrlo malih paketa, kao i fragmentaciju paketa i to konkretno na fragmente od većih paketa; na koje također otpada određeni postotak mrežnog prometa. Drugi postotak manjih paketa otpada na pakete odnosno upite koji sâmi po sebi ne sadrže mnogo podataka, pa sâmim time stanu u znatno manji paket. Primjer ovakvih paketa su definitivno ***DNS*** upiti i odgovori; koji su obično u vrlo malim paketima. Ne zaboravite da je sâmo tijekom otvaranja jedne Web stranice, potrebno napraviti razlučivanje i nekoliko desetaka domena, naravno s ***DNS*** upitima. Pogledajte poglavlje: **25.8.5.4. Vrste DNS upita**. U njemu je objašnjen ovaj proces. Dodatno, ako svaki klijent (računalo) ima otvoreno sâmo desetak Web stranica, sâmo broj *DNS* upita može narasti na stotine. Ako tome dodamo i broj korisnika (računala) na mreži, te *DNS* odgovore, dobivamo još veći broj znatno manjih paketa. Nadalje, ne zaboravimo i na to da primjerice *HTTP* i *HTTPS* protokoli rade tako da se *HTTP* i *HTTPS* paketi na mrežu šalju s postavljenim ***Dont Fragment*** bîtom (poglavlje: **23.2. IP fragmentacija**) što znači da će njihova veličina u konačnici; odnosno primjerice u dolasku, biti također vrlo vjerojatno manja od ***MTU*** vrijednosti na našoj lokalnoj mreži odnosno vjerojatno manja od 1500 bajta. Dodatno u slučaju upotrebe *HTTP* ili *HTTPS* protokola ne zaboravimo primjene u kojima korisnici (računala) dohvaćaju manje blokove sadržaja; poput primjene u radu i dohvaćanju programskog kôda sa servisa za tu namjenu*****.
Vezano za optimizacije, pogledajte i poglavlje: **25.3. Statistike, analiza i praćenje mrežnih paketa**.
Nadalje, neki protokoli poput **BitTorrent-a**, za preuzimanje samo jedne datoteke, otvaraju pedesetak (50+) ili više paralelnih konekcija; pomnožimo taj broj s brojem otvorenih *Torrent* datoteka u vašoj lokalnoj mreži.

- Druga vrsta mreža su mreže u kojima se većinom nalaze razni poslužitelji, mrežna oprema i uređaji. Oni ovisno o svojoj namjeni mogu uglavnom koristiti veće pakete od 1500 bajta ili čak veće (poglavlje: **23.4.3. Veliki mrežni okviri (*Jumbo frames*)**), ali u kombinaciji s TCP potvrdnim paketima (*TCP ACK*) i cijelim nîzom protokola koji koriste i manje pakete.

Osim navedenog postoji i cijeli nîz kontrolnih i drugih mrežnih protokola koji se standardno koriste u mreži, a da toga možda nismo niti svjesni. Mi smo napravili mjerenje na lokalnoj mreži (***LAN***) i to na mrežnim sučeljima na koja su spojeni: centralni usmjerivač prema internetu (na dijagramu označen kao ***Internet/Router***) te računala i poslužitelji, za koje je napravljen prosjek na dijagramu označen kao ***Delta:PC+Server***.

Rezultat našeg mjerenja i statistike, na slici **166.1.** pokazuje da možemo imati čak i do 40% (ili više) mrežnih paketa veličina od 64 bajta do 1024 bajta, u odnosu na 50-ak postotaka paketa većih od 1024 bajta.

Jasno je da što imamo manje mrežne pakete na mreži, a koje mora obraditi CPU, da je ta zadaća procesorski zahtjevnija.
Ne zaboravimo da je za 64 bajtne pakete za 1Gbps ukupne propusnosti potrebno obraditi čak 1,488 milijuna paketa u sekundi, (1.488.000 paketa u sekundi) dok je za pakete veličine 1500 bajta potrebno samo 81.000 paketa (***81 kpps***).

To znači da, ako imamo usmjerivač koji nema ***ASIC*** ili ima ***ASIC*** koji nije u stanju odraditi sve što je potrebno za puni proces usmjeravanja, a na njemu imamo pet (5) mrežnih sučelja na brzini 1Gbps; da bi za puno opterećenje od strane CPU-a bilo potrebno obraditi 7,44 milijuna paketa u sekundi (7.440.000 paketa), ako bi imali samo najmanje pakete od 64 bajta.
Ipak, stvarni uvjeti su negdje između najvećih i najmanjih paketa, kako je i vidljivo na našem mjerenju.

Naravno odstupanja od ovog mjerenja ovise o konkretnoj lokalnoj mreži (***LAN***) odnosno servisima i protokolima koji se na njoj najviše koriste. U konkretnom slučaju iz naših mjerenja, uz malo olakotniju okolnost imamo 59% paketa veličine između 1024 i 1632 bajta te ukupno 41% manjih paketa (7% od 128 do 1023 bajta, 21% od 65 do 127 bajta te 13% od 64 bajta). Ovdje možemo zaključiti da uređaj koji mora obrađivati ove mrežne pakete, troši resurse na 42% manjih mrežnih paketa u odnosu na 58% paketa koji su veće veličine, kako bi se zadržala ista propusnost. Naime da su ovih 42% manjih paketa bili veći paketi, dobila bi se drastično veća iskoristivost odnosno propusnost mreže.

Nećemo više teoretizirati već ćemo pogledati specijalizirani i optimiziran uređaj koji ima ***Intel® Xeon™ E3-1275V6 3.8Ghz Quad Core***; sa 16GB RAM i profesionalnim *Intelovim* mrežnim karticama (***I210-AT*** i ***X710-BM1***)** koje donekle pomažu ublažiti ovaj problem. Ako pogledamo službena mjerenja ovog uređaja[610] tada vidimo da je on u stanju konstantno isporučivati: ***1,5Mpps*** odnosno 1.500.000 paketa u sekundi.

Ne zaboravite uzeti u obzir i činjenicu da što je više paralelnih konekcija otvoreno prema mrežnom uređaju, to je uređaj opterećeniji (CPU ali i RAM), a sâmim time ima i manju efektivnu propusnost.



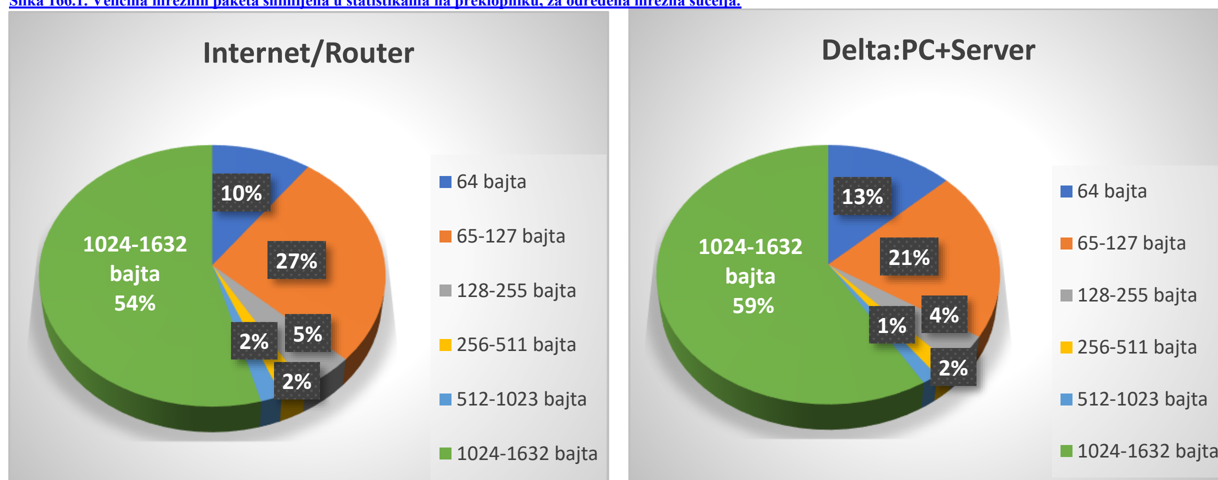
***Za prihvaćanje i obradu svakog mrežnog paketa potrebni su resursi računala odnosno uređaja: IRQ, DMA, CPU, RAM, mrežna kartica, … . Otegotna okolnost što se konačne propusnosti uređaja tiče, upravo je veličina mrežnih paketa koje on obrađuje jer veći paketi prenose više podataka. Za navedeni uređaj, koji može isporučiti 1,5Mpps za najveće pakete od 1500 bajta nam to daje propusnost od 18,4Gbps, dok nam obrada istog broja paketa za najmanje pakete (64 bajta) daje propusnost od samo malo više od 1Gbps.***



*Za točan izračun propusnosti ovisno o veličini i broju paketa, pogledajte poglavlje:*
***20.4. Gbps i Mpps – u čemu je veza.***

Pogledajmo i rezultate navedenih mjerenja.

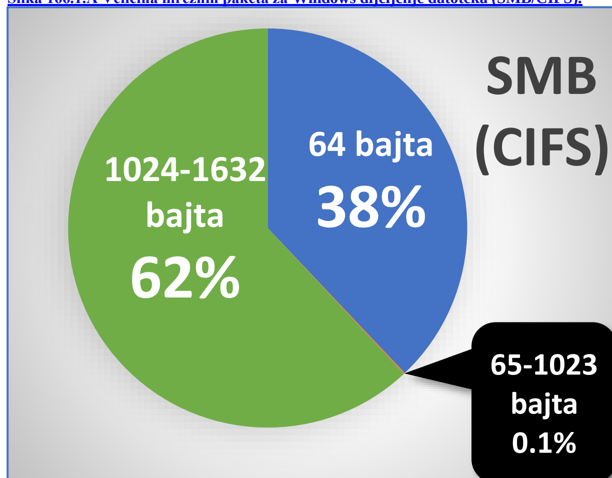
Slika 166.1. Veličina mrežnih paketa snimljena u statistikama na preklopniku, za određena mrežna sučelja.



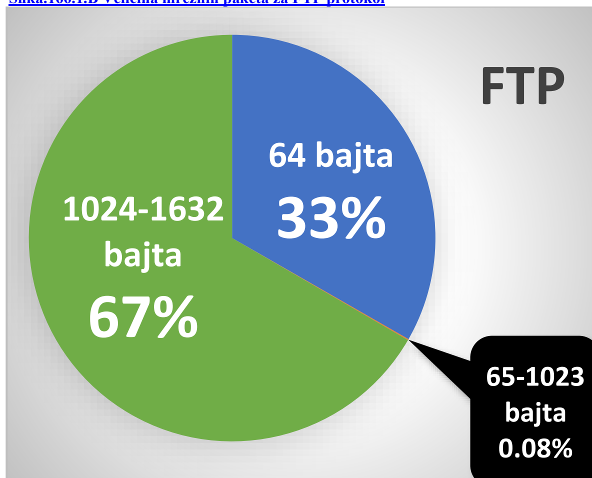

Probajmo saznati, barem na osnovi nekoliko mrežnih protokola za dijeljenje datoteka preko mreže, kolike su veličine mrežnih paketa prilikom kopiranja datoteka s poslužitelja na klijenta (osobno računalo).

Pogledajmo rezultate mjerenja u dijagramima koji slijede.

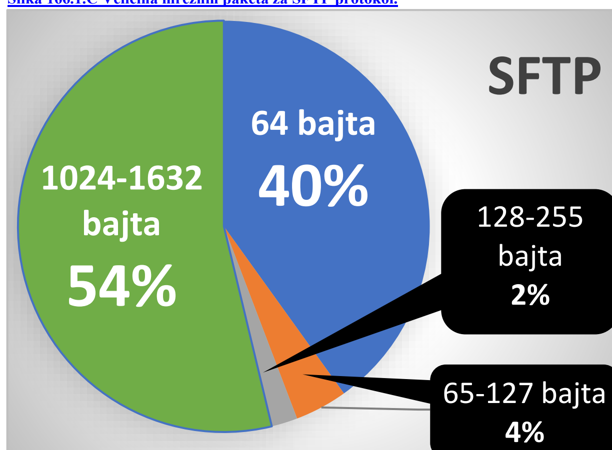
Slika 166.1.A Veličina mrežnih paketa za Windows dijeljenje datoteka (SMB/CIFS).



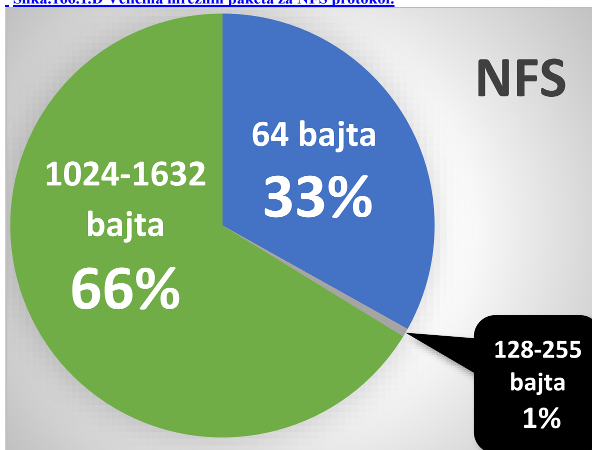

Slika.166.1.B Veličina mrežnih paketa za FTP protokol




Slika 166.1.C Veličina mrežnih paketa za SFTP protokol.




Slika.166.1.D Veličina mrežnih paketa za NFS protokol.




U ovom testu smo kopirali veće datoteke 14+GB preko mreže, s poslužitelja na klijentska računala, sve u lokalnoj mreži, korištenjem nekoliko često korištenih mrežnih protokola.
Cilj ovog testa nije bila brzina prijenosa već analiza veličine paketa!.

Što nam govore rezultati testa?

**A.)** U prvom testu (**Slika 166.1.A.**) koristili smo *Windows* protokol za mrežno dijeljenje datoteka (SMB/CIFS), pri čemu smo kao rezultat mjerenja dobili sljedeće: 38% paketa je bio veličine 64 bajta, a ostatak od 62% paketa je bio veličina od 1024 do 1632 bajta, uz neznatan broj paketa (~0.1%) čije veličine su bile negdje između!. Rezultat mjerenja govori nam kako iako je ***MTU*** na mreži 1500 bajta, imamo 62% većih paketa, a čak 38% ekstremno malih mrežnih paketa od 64 bajta.

**B.)** U sljedećem testu (**Slika 166.1.B.**) koristili smo FTP protokol preko kojega smo kopirali iste datoteke; rezultat je sljedeći: 33% paketa je bio veličine 64 bajta a ostatak od 67% paketa je bio veličina od 1024 do 1632 bajta, uz neznatan broj paketa (~0.08%) čije veličine su bile negdje između!. I ovdje je zaključak isti, s time da je ovdje ipak 5% više većih paketa u odnosu na ***SMB/CIFS*** protokol, ali i dalje postoji vrlo veliki postotak malih paketa od 64 bajta (čak 33%).

**C.)** U ovom testu (**Slika 166.1.C.**) smo koristili SFTP protokol (koji koristi ***SSH*** servis/protokol u pozadini), koji u pozadini sve podatke prvo kriptira odnosno dekriptira na prijemnoj strani (na klijentu). Rezultati su redom: 40% paketa je veličine 64 bajta a 54% je veličina od 1024 do 1632 bajta, uz sada znatniji broj drugih manjih paketa (4% paketa od 65 do127 bajta te 2% paketa veličine od 128 do 255 bajta). Dakle ovdje imamo još nešto manji postotak velikih mrežnih paketa uz veći postotak jako malih paketa te dodatan mali postotak srednje malih paketa.

**D.)** U zadnjem testu smo koristili NFS protokol za dijeljenje datoteka preko mreže koji je pokazao sljedeće: 33% paketa je bio veličine 64 bajta a ostatak od 66% paketa je bio veličina od 1024 do 1632 bajta, uz neznatan broj paketa od 1% čije veličine su između 128 i 255 bajta. Dakle ovaj protokol je među najboljima, što se tiče većeg postotka većih paketa.

Rezultate mjerenja smo pratili na konfigurabilnim mrežnim preklopnicima koji imaju te mogućnosti, poput svih **Cisco** i **Juniper** preklopnika, kao i nekih preklopnika drugih proizvođača. Istu statistiku smo mogli dobiti i na računalu, snimanjem mrežnog prometa prema odredišnoj adresi (poslužitelju), u programu ***Wireshark***, te kasnijim odabirom: **Statistics → Packet Lengths**. Rezultate ovih mjerenja možemo uzeti u obzir informativno, upravo u svrhu spoznaje o tome koji mrežni protokoli kreiraju koje veličine mrežnih paketa u lokalnoj mreži (*LAN*), u kojoj je ***MTU*** vrijednost postavljena na ***1500*** bajta.

Moglo bi se očekivati da barem u lokalnoj (*LAN*) mreži koja ima veće ***MTU*** vrijednosti od *WAN* mreža (prema internetu), mrežni protokoli uspijevaju bolje iskoristiti veće ***MTU*** vrijednosti, tako da je veći postotak mrežnih paketa što veće veličine (do granice ***MTU***-a od 1500 bajta), što često nije slučaj za ***TCP*** protokol (ovisno i o njegovoj optimizaciji). Kod ***UDP*** protokola to nije (***nužno***) slučaj.



***Analiza s programom Wireshark pokazuje da su osnovni uzrok TCP paketi za potvrđivanje primitka; zbog TCP mehanizama potvrđivanja ispravnog primitka paketa (ACK poruke), koje su činile veći dio postotka malih paketa (oni su bili veličine 64 bajta), kao i neke specifičnosti rada svakog od testiranih mrežnih protokola.***

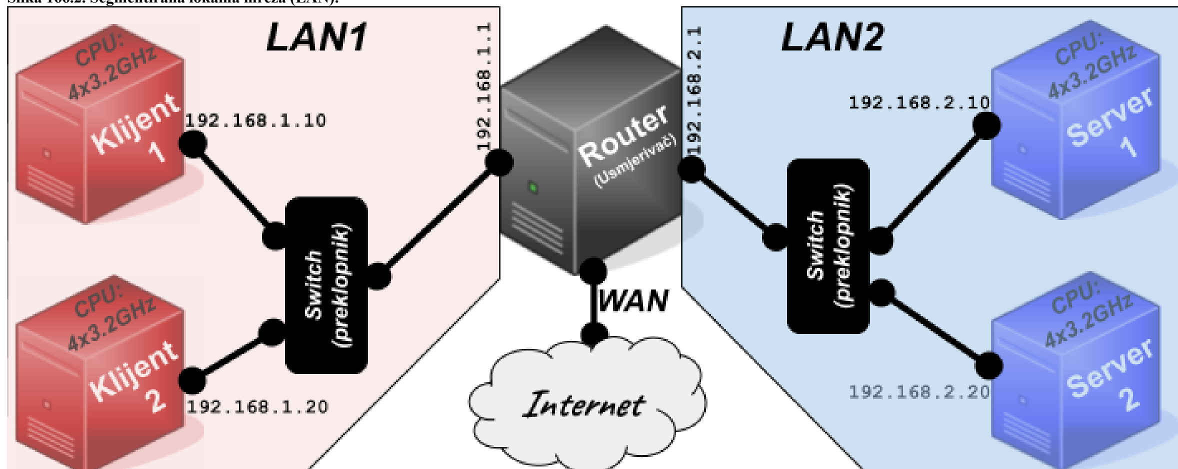
***Rješenje može biti optimizacija na razini servisa (ovisno o servisu: SMB/CIFS, FTP, NFS,…) ili djelovanjem na veličinu TCP prozora, što je recimo za NFS moguće izvesti.***

**Izvori informacija: (610),(K-7), RFC 2544.**

## 20.4.1. Mjerenja propusnosti bez ASIC sklopova

Za potrebe sljedećeg testa koristili smo logičku shemu spajanja, kako je vidljivo na slici **166.2**. Dakle lokalnu mrežu (***LAN***) smo segmentirali na dvije odvojene mreže: ***LAN1*** i ***LAN2*** uz opcionalnu upotrebu ***WAN*** mreže, koja nam trenutno nije važna.

**Slika 166.2. Segmentirana lokalna mreža (LAN).**




Za funkciju usmjerivača (***Router*** na slici) smo odabrali mikro računalo ***NanoPI R2S*** s ***ARM*** procesorom koji ima četiri *ARM* jezgre ***Cortex-A53*** na 1.3GHz te 1GB RAM memorije. Ono ima dvije ***Gbps*** mrežne kartice: jednu ***Realtek*** RTL8153 na internoj USB 3.0 sabirnici te drugu ugrađenu u sâm procesor (*Rockchip* RK3328). Ovakvo računalo je procesorski i što se tiče mrežnih kartica, kao i propusnosti među njima znatno snažnije od većine usmjerivača za male i srednje tvrtke dostupne na tržištu. Za analizu rada snažnijeg hardvera, rezultate ovog mjerenja možete ekstrapolirati.

Na ovo mikro računalo smo instalirali ***ARMbian*** Linux.
Na strani ***LAN2*** mreže, imamo dva računala (***Server1*** i ***Server2***) na kojima pokrećemo nekoliko *iperf* procesa od kojih se svaki veže za jednu CPU jezgru *(A0 – A3)*, i na točno određeni TCP port. `Iperf3` softverski paket se nalazi na ***EPEL*** repozitoriju.
Pogledajmo skraćeni primjer pokretanja, za **Server1,** sa samo dva pokrenuta ***iperf3*** procesa:

```
iperf3 -s -A0 -p 5201
iperf3 -s -A1 -p 5202
```

Na klijentskim računalima (***Klijent1*** i ***Klijent2***) pokrećemo klijentsku stranu *iperf* aplikacije; prvo samo za jednu konekciju, a potom povećavamo broj paralelnih konekcija (`-P XY`) prema poslužiteljima (***Server1*** i ***Server2***).

Primjerice za **200** konekcija bi to bilo:

```
iperf3 -Z -c 192.168.2.10 -t 600 -P 100 -p 5201
iperf3 -Z -c 192.168.2.10 -t 600 -P 100 -p 5202
```

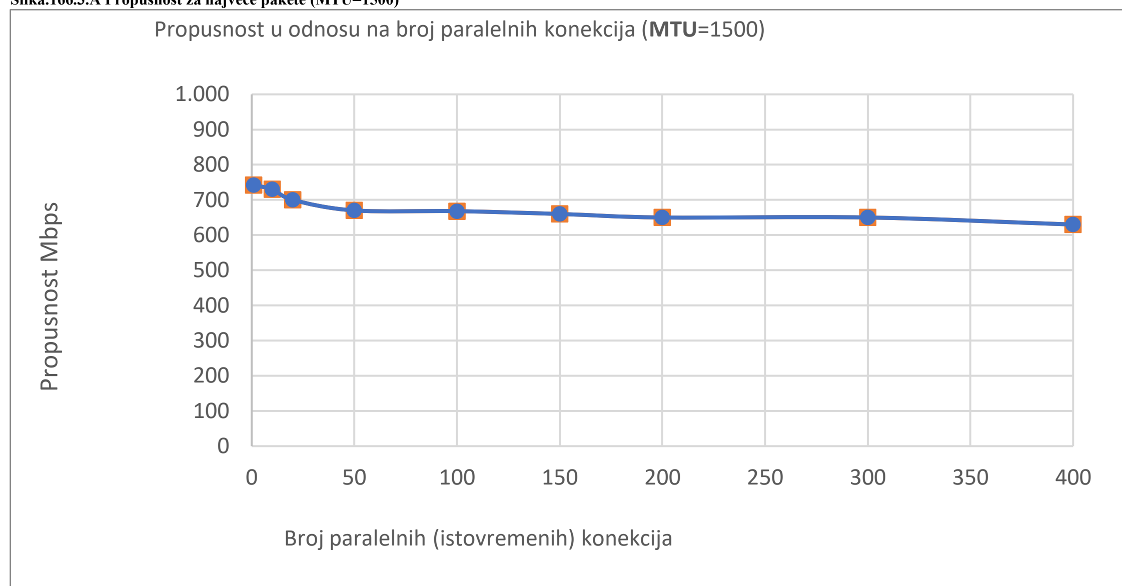
*Iperf3* prihvaća maksimalno 100 paralelnih konekcija, te zbog toga pokrećemo više *iperf3* serverskih i klijentskih procesa na različitim portovima. Pogledajte rezultate mjerenja, tijekom povećanja broja paralelnih konekcija od 1 do 400 za najveće pakete (**Slika.166.3.A**) (***MTU***=1500), te za mješovite veličine paketa: 50% sa ***MTU***=1500 te 50% sa ***MTU***=127 bajta, na slici **166.3.B.** Manju veličinu paketa možemo dobiti s prekidačem `-M 167` koju smo odabrali kao prosjek manjih paketa s prethodnih statistika.

Manji ***MTU*** smo dobili smanjivanjem veličine TCP segmenta odnosno ***MSS*** veličine (**Pogl. poglavlje: 24.2.1.**) koja indirektno utječe na *MTU* veličinu (***MTU=MSS***-40 bajta).
***Ovdje smo donekle pokušali kreirati mješovite veličine paketa, poput IMIX preporuka. Pogledajte usporedbu mjerenja na raznim uređajima: IPERF s maksimalnim MTU vs IMIX***(611)**.**

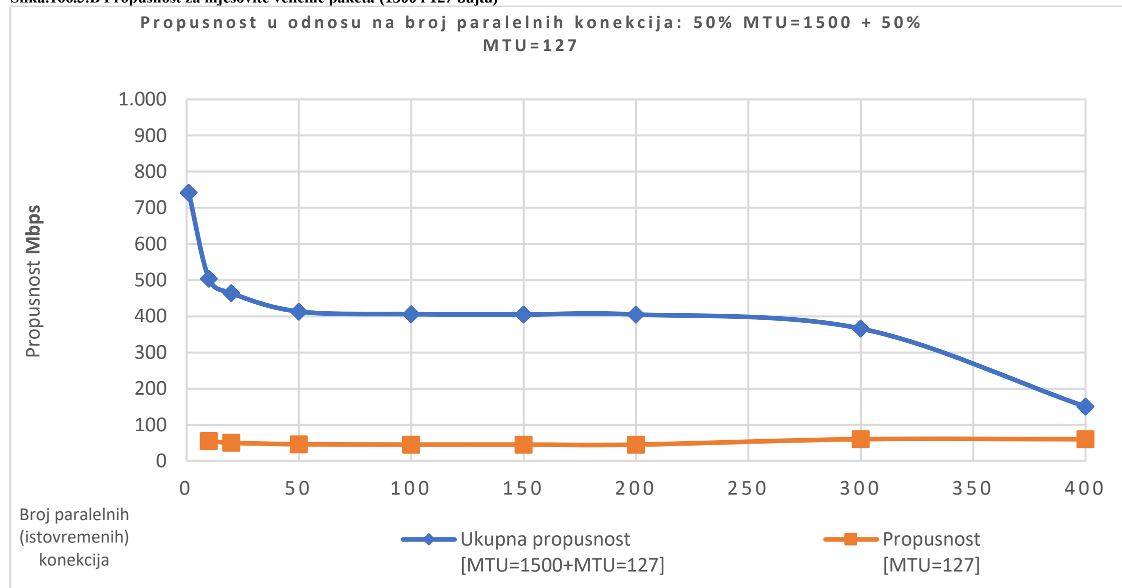


Za optimizacije mrežnog sustava, pogledajte poglavlje:
**25.3. Statistike, analiza i praćenje mrežnih paketa.**

**Slika.166.3.A Propusnost za najveće pakete (MTU=1500)**




**Slika.166.3.B Propusnost za mješovite veličine paketa (1500 i 127 bajta)**




Na propusnost utječe i postavka veličine **TCP** prozora (poglavlje. **24.2.8. Kontrola protoka (TCP Window i Window scaling)**) što možemo mijenjati na strani klijenata (za **Windows** je to 8kB). Primjerice s prekidačem: `-w 32768` ju možemo povećati na 32kB. Za Linux je ona standardno veća pa je i propusnost veća. Na usmjerivaču nismo imali uključenu translaciju adresa (**NAT**), već samo obično usmjeravanje između mrežnih sučelja (**LAN1 - LAN2**), i to bez i jednog dodatnog pravila filtriranja odnosno bez uključenog vatrozida, koje bi još više smanjilo efektivnu propusnost uređaja. Vidljivo je kako se pojavom manjih paketa na mreži uređaj, nakon određenog broja paralelnih konekcija, uređaj zagušuje te mu se ukupna efektivna propusnost počinje drastičnije smanjivati (slika **166.3.B**). To se konkretno događa na granici preko 300 paralelnih konekcija.

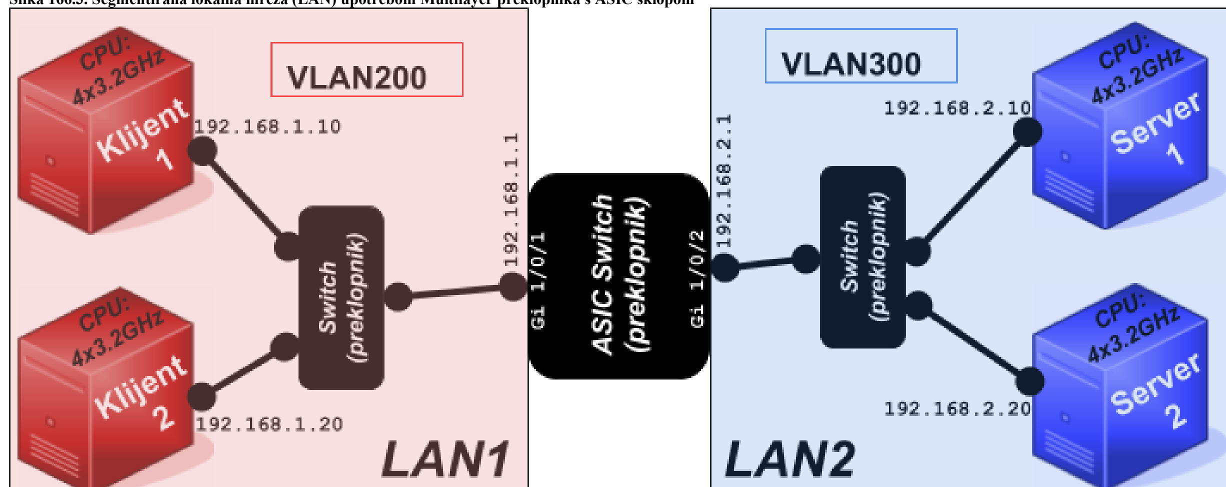
**Izvori informacija: (610),(611),(1057),(K-7)**,`man iperf3`.

## 20.4.2. Mjerenje propusnosti sa ASIC sklopovima

U ovom testu kreirat ćemo sličnu topologiju kao i prethodnu, ali upotrebom *Multilayer* preklopnika koji ima ***ASIC*** sklopove koji hardverski odrađuju sve funkcije usmjeravanja. Konkretan preklopnik koji smo koristili je ***Cisco 3750*** serija preklopnika (iz 2004.g.), konkretno: WS-C3750G-24TS-S koji ima deklariranu propusnost za 64 bajtne pakete od ***35,7 Mpps***. Mada ***Cisco*** ima i slabije modele koji su isto *Multilayer* preklopnici koji rade na OSI 2/3/4 slojevima, poput serije ***3650***.

Međutim mi smo imali navedeni model pri ruci, pa smo ga i iskoristili. *U istoj **ASIC** kategoriji se nalazi i primjerice **Juniper** EX2300-24T (**95 Mpps**).*

**Slika 166.3. Segmentirana lokalna mreža (LAN) upotrebom Multilayer preklopnika s ASIC sklopom**




Na ovom preklopniku smo kreirali dva ***VLAN***-a: **200** i **300**, s kojima smo segmentirali mrežu na dva djela. Iako smo svako računalo mogli spojiti na svoje zasebno mrežno sučelje na ovakvom preklopniku, zbog još veće brzine. Odlučili smo testirati sâmo jedno mrežno sučelje preklopnika, na svakoj **LAN** mreži (***LAN1*** i ***LAN2***), kako bi imali uvjete slične prvom testiranju.



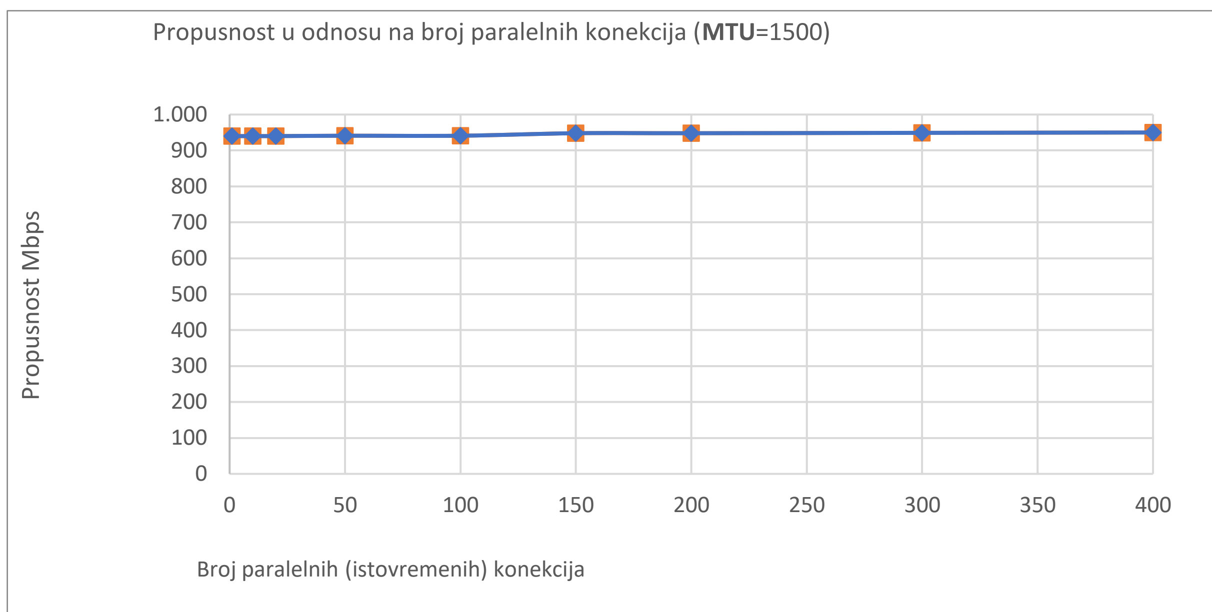
Za ***VLAN***-ove pogledajte poglavlje:
**20.6.2. VLANovi odnosno virtualne lokalne mreže.**

Konfiguracija preklopnika, koja se tiče sâmo usmjeravanja i ***VLAN*** mreža je:

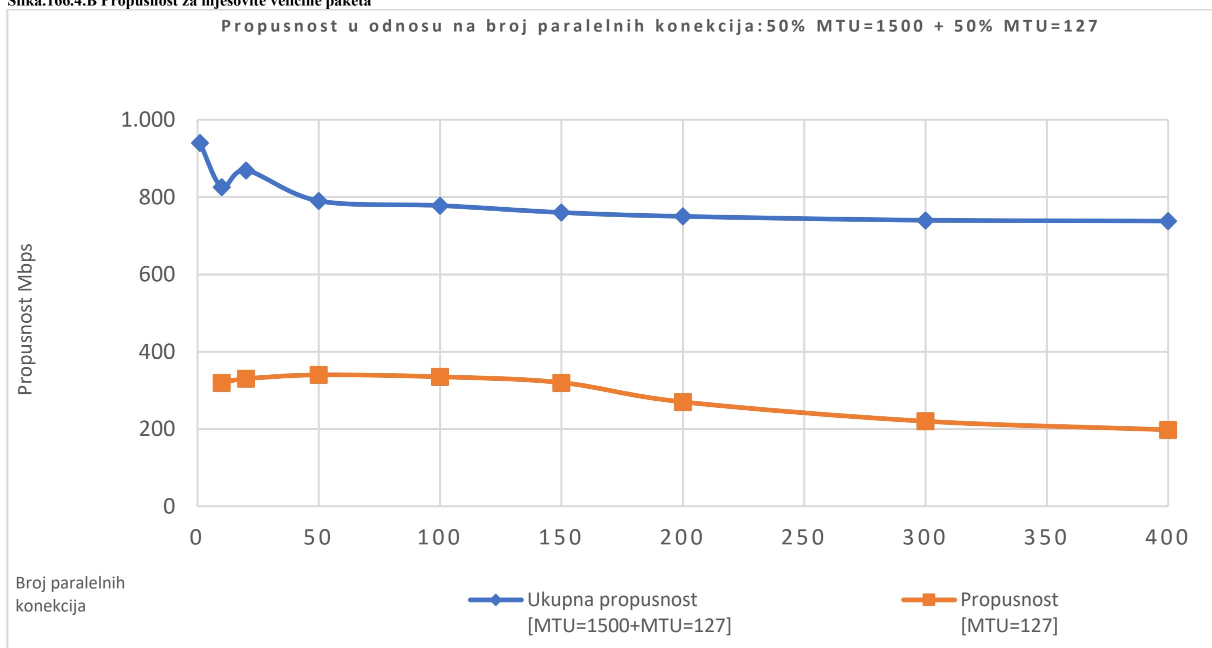
| Globalna konfiguracija usmjeravanja (`configure terminal`) te kreiranje VLAN mreža | Konfiguracija VLAN mrežnih sučelja |
|---|---|
| `conf t`<br>`ip routing`<br>`vlan 200`<br>`name LAN-1`<br>`vlan 300`<br>`name LAN-2` | `interface Vlan300`<br>`ip address 192.168.2.1 255.255.255.0`<br>`interface Vlan200`<br>`ip address 192.168.1.1 255.255.255.0` |
| Konfiguracija mrežnog sučelja **Gi1/0/1** | Konfiguracija mrežnog sučelja **Gi1/0/2** |
| `interface GigabitEthernet1/0/1`<br>`switchport access vlan 200`<br>`switchport mode access` | `interface GigabitEthernet1/0/2`<br>`switchport access vlan 300`<br>`switchport mode access` |

Rezultati mjerenja su ovdje vrlo zanimljivi jer smo na strani *iperf3* poslužitelja (***Server1*** i ***Server2***), došli do njegovog ograničenja.
S time smo dokazali da je u prvom testu hardver (***iperf3*** serveri i klijenti) bio zadovoljavajući.
U testu (**166.4.B**) s manjim paketima ***MTU***=127 bajta (prekidač `-M 167`) CPU jezgre su konstantno bile na 100% opterećenja te su one očito postale usko grlo.
I to ponajviše zbog obrade signala prekida jer je servis ***ksoftirq*** počeo zauzimati također 100% CPUa što znači da je računalo (***Server1*** i ***Server2***) toliko opterećeno softverskim signalima prekida, da nije bilo u stanju obrađivati mrežne pakete brzinom kojom su dolazili. S obzirom na usporedbu sa specijaliziranim hardverom**(Intel Xeon ***E3-1275V6 3.8Ghz*** CPU te mrežne kartice ***I210-AT*** i ***X710-BM1***) za koji se navodi da je u stanju isporučiti maksimalno ***1,5 Mpps***, to je bilo i očekivano.

Slika.166.4.A Propusnost za najveće pakete (MTU=1500)




Slika.166.4.B Propusnost za mješovite veličine paketa






Specijalizirani preklopnici koji imaju ***ASIC*** sklopove koji rade na OSI 2/3/4 slojevima, koji su u stanju obraditi minimalno ***35,7 Mpps*** (za 24x1Gbps), nemaju problema sa smanjenjem propusnosti, neovisno o broju istovremenih konekcija ili veličine paketa koje moraju obrađivati, kao niti istovremenom upotrebom više gigabitnih mrežnih sučelja koja su vršno opterećena.

**Izvori informacija: (610),(611), (1057),(K-7),** `man iperf3`**.**

# 20.5. Kontrola protoka (*flow control*) na OSI sloju 2

**Slijedi napredna cjelina!**
Kontrola protoka za *ethernet* mreže je mehanizam za privremeno zaustavljanje prijenosa podataka. Prvi mehanizam za kontrolu protoka odnosno mrežni okvir (paket) za stanku, definiran je standardom IEEE **802.3x**. Naime preopterećeni mrežni čvor odnosno element na mreži može poslati poruku za stanku, koja zaustavlja prijenos podataka za određeno vremensko razdoblje. Za tu funkcionalnost se šalje posebni mrežni okvir (*paket*) na OSI sloju dva odnosno poseban *media access control* (***MAC***) mrežni okvir. Dakle ovaj mrežni okvir koristi se za slanje naredbe za pauziranjem slanja daljih paketa na mrežu.
Unutar ovog mrežnog okvira se takozvano ***opcode*** polje postavlja na vrijednost 0x0001 (*heksadecimalno*). Međutim sâmo oni mrežni elementi (uređaji) koji su konfigurirani za *full-duplex* rad mogu slati ove takozvane *PAUSE* mrežne okvire.
Kada jedna strana želi privremeno zaustaviti drugu stranu veze, šalje ovaj okvir za stanku na jedinstvenu 48-bitnu odredišnu adresu (MAC adresu) ili na 48-bitnu rezerviranu, *multicast* adresu **01-80-C2-00-00- 01**. Zbog toga što svi uređaji koji podržavaju kontrolu protoka i prepoznaju ovu *multicast* MAC adresu, gubi se potreba slanja ovog paketa na točnu MAC adresu susjeda, kojem se šalje ovaj zahtjev, pa se ova procedura malo ubrzava.



Primjer mrežnog okvira u kojem se šalje pauza (*flow control*), dostupan je na sljedećoj web adresi:

**https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=get&target=Ethernet_Pause_Frame.cap**

Posebna *multicast* adresa odabrana je iz raspona adresa koje su rezervirane prema ***IEEE 802.1D*** standardu koji specificira rad preklopnika koji podržavaju *bridge-ing* i *Spanning Tree* funkcionalnost. Uobičajeno mrežni okvir s ovako definiranom *multicast* adresom koji se šalje na preklopnik prosljeđuje se na sva mrežna sučelja tog preklopnika. Međutim ova *multicast* adresa posebna je i neće biti proslijeđena od strane preklopnika koji podržava protokol ***802.1D*** (u načelu ***Spaning Tree***). Umjesto toga okviri koji se šalju na ovu *multicast* adresu podrazumijeva se da trebaju djelovati sâmo unutar preklopnika koji ih je primio. Ovaj mrežni okvir za stanku obuhvaća razdoblje zatvaranja vremena pauze u obliku dvo bajtnog odnosno 16 bitnog nepotpisanog broja (0 do 65.535). Ovaj broj definira vrijeme trajanja stanke. Vrijeme stanke mjeri se u jedinicama stanke "*quanta*" pri čemu je svaki ovaj bît jednak 512 bîtnoj vremenskoj jedinici vremena. Međutim ovaj protokol je slabo prihvaćen pa se tijekom 2004 godine pojavio i protokol ***802.3ar***, ali je i on odbačen jer je uveden noviji protokol: 802.1p. Zatim je znatnije prihvaćen protokol za podatkovne centre ***802.1Qbb*** (2010 godine). U svakom slučaju i klasični protokol IEEE **802.3x** je i dalje u upotrebi u klasičnim mrežama, ali se u praksi vrlo malo ili gotovo uopće ne koristi jer je kontrola protoka u većini slučajeva prepuštena višim protokolima odnosno protokolima na višim OSI slojevima poput ***TCP***-a koji ima isti mehanizam.



Pogledajte poglavlje:
**24.2.8 Kontrola protoka (TCP Window i Window scaling).**

Naime kontrola protoka na OSI sloju dva (OSI 2) se odrađuje prema definiranom protokolu 802.3x koji definira kako klijent odnosno strana koja prima mrežne pakete u slučaju kada je preplavljena s paketima i ne stigne ih obraditi, može na OSI sloju dva poslati posebnu MAC poruku strani koja odašilje ili poruku poslati sâmom preklopniku da privremeno zaustavi slanje. Ovaj okvir signalizira drugom kraju veze da se pauzira prijenos za određenu količinu vremena, što je navedeno u okviru.
Imajte na umu da *PAUSE* okvir koristi MAC LLC ugniježđenje (*enkapsulaciju*). Dakle ovdje se sve događa na OSI sloju dva pa primjerice TCP mehanizmi za istu namjenu istovremeno mogu reagirati na ovakav događaj. Naime pošto *Ethernet* kontrola protoka radi na nižem sloju od TCP ili IP sloja, ona je stoga neovisna o njima. Drugim riječima, ova kontrola protoka može se koristiti bez obzira na to koji se protokoli više razine koriste.



Važna nuspojava je to da niti TCP niti IP sloj ne znaju što ***Ethernet*** kontrola protoka radi; oni djeluju pod pretpostavkom da ne postoji druga kontrola protoka, osim one koju oni koriste. *Ethernet* kontrola protoka funkcionira sâmo i isključivo između dva izravno povezana mrežna uređaja, a mrežni paketi kontrole protoka nikad se ne prosljeđuju između međuveza koje povezuju krajnje uređaje u komunikaciji. Dakle dva računala koja su spojena preko preklopnika nikada neće poslati poruke o pauziranju slanja mrežnih okvira/paketa jedno direktno drugome, već jedino mogu poslati mrežne okvire na preklopnik; i obratno. Preklopnik može poslati ovakav mrežni okvir na računala. *Ethernet* okviri za pauziranje imaju ograničeno trajanje; oni će automatski *isteći* nakon određenog vremena. Vrijeme isteka postavlja uređaj koji odašilje ovaj paket. Pauzirana veza odnosno mrežno sučelje prema uređaju koje je poslalo ovaj zahtjev za pauziranjem slanja će spriječiti prijenos svih podataka preko ove veze, osim za nove pakete iste vrste (*pause*) sve dok ne istekne vrijeme pauziranja slanja.

**U čemu je problem?**
Što se događa kada se ***Ethernet*** kontrola protoka (*Ethernet flow control*) koristi zajedno s TCP kontrolom protoka?. Pretpostavimo da imamo dva izravno povezana računala od kojih je jedan sporiji od drugog. Brže računalo za slanje počinje slati puno podataka prema sporijem računalu. Prijemnik na kraju primjećuje da je preopterećen podacima te šalje paket za pauziranje (*MAC control: pause*) pošiljatelju. Pošiljatelj prima ovu poruku i privremeno prekida slanje. Kada istekne vrijeme pauziranja (stanke), pošiljatelj će nastaviti slati svoje podatke na drugo računalo.

Nažalost TCP sloj na pošiljatelju neće prepoznati da je prijemnik preopterećen jer nema izgubljenih podataka: prijemnik će obično zaustaviti pošiljatelja prije nego što izgubi sve podatke. Tako će pošiljatelj nastaviti ubrzavati eksponencijalnom stopom jer nije vidio izgubljene podatke, i krenuti će sa slanjem dvostruko brže; prema TCP mehanizmima. Budući da prijemnik ima stalan nedostatak brzine, to će zahtijevati da prijemnik dvaput počne slati poruke za pauziranje (na OS sloju 2) - ***Ethernet***.
Zbog toga se cijeli proces ponovno ponavlja i prijemnik će se u nekom trenutku toliko zagušiti da će početi ispuštati pakete te ih neće potvrđivati, a tek tada će pošiljatelj to primijetiti na TCP razini; jer neće dobiti potvrdu primljenih paketa (**TCP ACK**).
Je li to problem? Na neki način nije. Budući da je TCP pouzdan protokol ništa se nikada nije "izgubilo" i jednostavno će se izgubljeni paketi ponovno poslati. Naime **Ethernet** kontrola protoka postiže istu stvar kao i kontrola protoka na razini TCP-a u ovoj situaciji, jer oboje usporavaju prijenos podataka do brzine koju sporiji ili privremeno usporeni uređaj može podnijeti.



*Međutim u stvarnim uvjetima uvijek imamo kompleksniju mrežu i mrežnu komunikaciju u kojoj rijetko komuniciraju sâmo dva računala koja su direktno spojena preko preklopnika. Obično imamo veze preko više preklopnika, usmjerivača, i drugih uređaja, a dodatno su i njihove međuveze različitih brzina, što uvodi znatnije probleme istovremenom upotrebom i **Ethernet** i **TCP** kontrole protoka, zbog koje dolazi do nepotrebnog zaustavljanja prijenosa i/ili ponovnog slanja izgubljenih paketa. Dodatno je važno znati kako **Ethernet flow control** može i ne mora biti aktiviran i na preklopniku, ali i na računalima u komunikaciji. Isto tako **Ethernet flow control** može biti aktiviran u smjeru slanja (**send**) kao i u smjeru primanja (**receive**) ovakvih paketa.*

**Može i još gore**
S *Ethernet* kontrolom protoka koja je omogućena na preklopniku, preklopnik ima drugačiji pristup; on će poslati svoje vlastite poruke o pauziranju na svoje mrežno sučelje (*port*) preko kojeg šalje podatke, na sada preopterećenoj vezi odnosno mrežnom sučelju (portu). To znači kako će strana koja šalje podatke primiti ovu poruku za pauzu od preklopnika i prekinuti sve prijenose podataka na određeno vrijeme. Dakle kroz ovo mrežno sučelje (*port* na preklopniku) neće biti nikakvog slanja mrežnih paketa prema nikome, a ne sâmo prema izvoru koji je bio preopterećen!.
Dakle ovime se napravila blokada apsolutno svôg prometa od i prema ovom pošiljatelju.
Na kraju će vremenska stanka isteći i pošiljatelj će nastaviti slati podatke.
Nažalost TCP mehanizam na strani pošiljatelja neće znati da ništa nije u redu, te će nastaviti slati podatke, ponovno povećavajući brzinu te će opet preopteretiti primatelja. Kao i prije, ciklus će se nastaviti ponavljati u krug. Stoga je preporuka isključiti ***Ethernet flow control*** na preklopnicima, a na računalima se eventualno može ostaviti uključen, mada čak i ne u većini slučajeva.
Kako bismo provjerili je li *flow control* u upotrebi, potrebno je za `eth0` mrežno sučelje pokrenuti naredbu `ethtool`:

```
ethtool -a eth0

Pause parameters for eth0:
Autonegotiate:  on
RX:             on
TX:             on
RX negotiated:  on
TX negotiated:  on
```

Ovdje vidimo sljedeće:

- Metoda dogovaranja brzine i *duplex* načina rada (*auto negotiate*) je vidljiva u (`Autonegotiate`) i ovdje je uključena (`on`), a ona ima direktnu logičku vezu s *flow control* mehanizmom. Naime ne možemo ručno i direktno mijenjati odnosno uključivati i isključivati *flow control.* Dakle ako želimo mijenjati *flow control za RX i TX, prvo moramo isključiti* `Autonegotiate`. Naime u procesu dogovaranja brzine i *duplex* načina rada (*auto negotiate*) mrežne kartice, dvije strane se osim dogovaranja brzine i *duplex* načina rada, standardno dogovaraju i za aktivaciju *Ethernet flow control* mehanizma, koji se pak standardno uključuje (`on`) i za RX i TX. Za više detalja o izgledu mrežnog okvira koji se koristi za *auto negotiate*, pogledajte izvore:[(468),(469)] jer se tamo vidi gdje se u polju „***ability flags***" definira upotreba *Ethernet flow control*-a.
- U sljedeća dva reda (`RX`) i (`TX`) vidimo kako je *flow control* uključen i za prijem (RX) i slanje (TX).
- I u zadnja dva reda vidimo, za (`RX negotiated`) i (`TX negotiated`) kako su uključeni.

Dakle ako želimo isključiti *flow control* i za primanje (RX) i slanje (TX), tada moramo napraviti sljedeće:

```
ethtool -A eth0 tx off rx off autoneg off
```

Pripazite što radite i želite li ovo napraviti. Naime u nekim slučajevima visoko performantnih mreža se ovo i preporučuje isključiti jer unosi dodatno kašnjenje (latenciju), a kod mrežnih kartica s više hardverskih mrežnih nîzova (*queues*) unosi dodatne probleme jer oni nîzovi (*queue*) koji su brže obradili mrežne pakete moraju čekati na one koji ih još nisu obradili; zbog bilo kojeg razloga (aplikacija/CPU jezgra ili nešto treće). Dodatno za visoko performantne mreže ovdje možemo imati i usporavanja koja smo opisali, kada je *Ethernet flow control* uključen na strani preklopnika, a koristimo transportni protokol poput TCP-a koji već ima ugrađen isti mehanizam koji će dodatno usporiti mrežnu komunikaciju kada se koristi u kombinaciji sa *Ethernet flow control* mehanizmom.



Za detalje kontrole protoka na TCP sloju, detaljnije proučite poglavlje:
**24.2.8. Kontrola protoka (TCP Window i Window scaling).**

**Izvor informacija: (465),(467),(468),(469),(470),(471),(472).**

# 20.6. Mrežna sučelja na OSI sloju 2

U ovoj cjelini govorit ćemo o mrežnim sučeljima koja rade na OSI sloju dva (OSI 2).

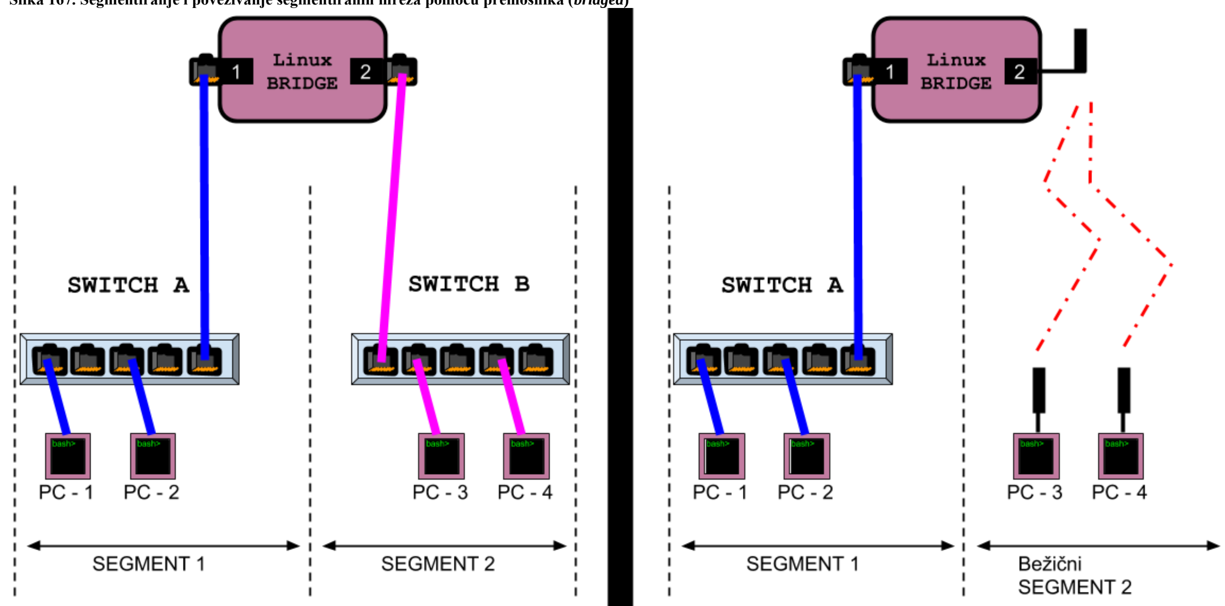
## 20.6.1. Mrežni most (*bridge*) odnosno premosnik

**Slijedi napredno poglavlje!**

Svima je više-manje jasno kako Linux možemo koristiti kao usmjerivač, koji radi na OSI sloju tri (OSI 3). Međutim, osim toga Linux može raditi i kao mrežni most (engl. *Bridge*) na nižem sloju rada odnosno na OSI sloju dva (OSI 2). Upotrebu u kojoj se Linux koristi za premošćivanje odnosno segmentiranje mreža vidljiv je na lijevom djelu donje slike 167 koja predstavlja ovakvu upotrebu Linux računala.

S druge strane imamo i upotrebe u kojoj su dva mrežna sučelja: jedno žičano, a drugo bežično upotrebom Linux mrežnog mosta povezana, kako bi se povezale žičana i bežična mreža (desni dio slike 167).
U oba primjera rada govorimo o takozvanom ***bridge*** načinu rada odnosno načinu rada Linuxa kao mrežnog mosta.

Slika 167. Segmentiranje i povezivanje segmentiranih mreža pomoću premosnika (*bridgea*)




Povezivanje mreža pomoću *ethernet* mrežnog mosta (*bridge-a*) radi se obično kako bi se stvorila veća mreža.
Standard za mrežne mostove je ANSI/IEEE ***802.1d***. Upotrebom mrežnog mosta povezuju se dva odvojena segmenta mreže na način neovisan o mrežnom protokolu. Pri tome se mrežni paketi prosljeđuju na temelju *Ethernet* adrese (***MAC*** adrese), a ne **IP** adrese poput usmjerivača.

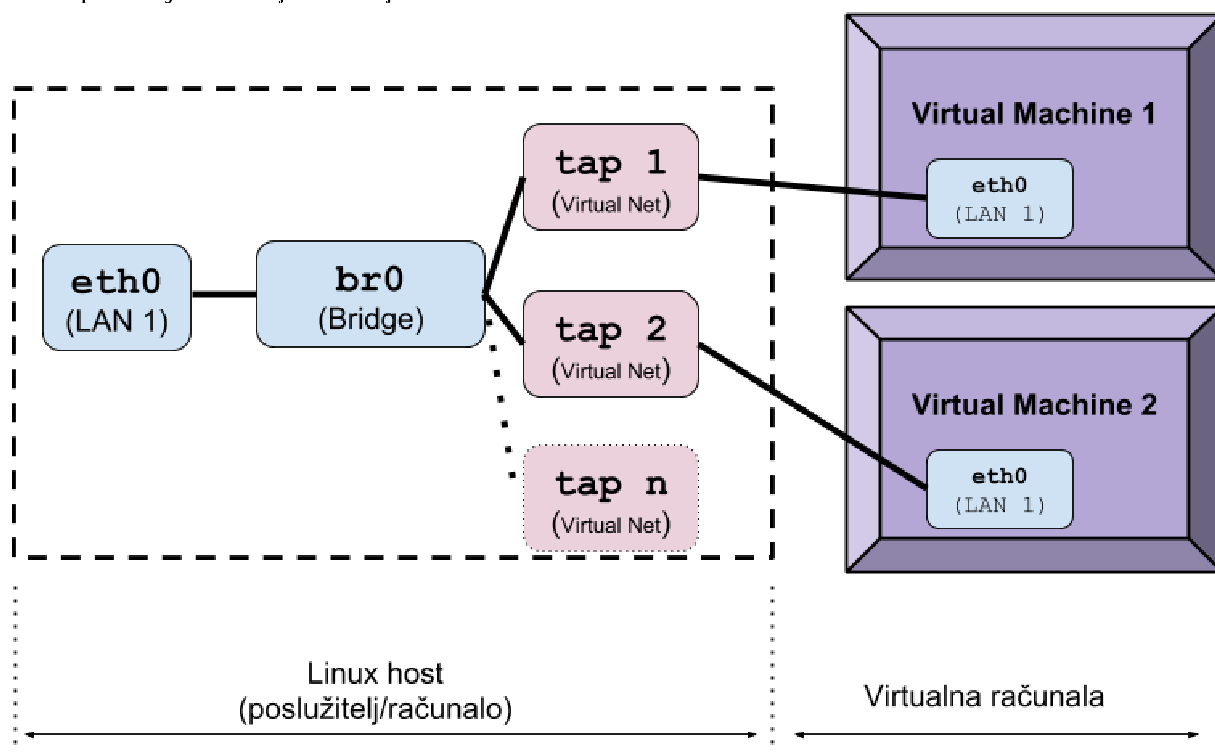
Budući da se prosljeđivanje mrežnih paketa vrši na razini OSI sloja dva (OSI 2) svi viši mrežni protokoli prolaze transparentno kroz mrežni most.



Važno je razumjeti da osim fizičke mrežne kartice, svoju zasebnu i jedinstvenu **MAC** adresu dobiva i logičko mrežno sučelje kao što je **Bridge,** ali i sučelja poput: **Bond**, **VETH**, **TAP** i nekih drugih, jer i ona rade na OSI sloju dva [*Ethernet*].

Nadalje, moguća je i malo drugačija primjena Linux mrežnog mosta (*bridge-a*) koja je najčešća u virtualizaciji, kako je vidljivo na slici 168. dolje.

**Slika 168. Upotreba *bridge* mrežnih sučelja u virtualizaciji**




Kod ovakve primjene koristimo ***bridge*** mrežno sučelje (pr. `br0` ili `vmbr0`) kao vezu s virtualnim ***tap*** mrežnim sučeljem na koje se može spojiti virtualno računalo odnosno virtualna mrežna kartica virtualnog računala.

Naime u ovakvom slučaju imamo fizičku mrežnu karticu na Linux računalu (`eth0`) koja je spojena u *bridge* (`br0`) s `tap` virtualnom mrežnom karticom (ili više njih).



Kod ovakve primjene moguće je imati više `tap` virtualnih mrežnih sučelja spojenih u jedan ***bridge***, konkretno na *bridge* sučelje `br0`.

To znači kako će sav mrežni promet od i prema `tap` virtualnom mrežnom sučelju (ako nismo uključili *vatrozid*) prolaziti kroz *bridge* sučelje `br0`, a preko kojeg će taj mrežni promet u konačnici završiti na fizičkoj mrežnoj kartici `eth0`. Pod „*sav*“ mrežni promet mislimo na sve mrežne protokole koji rade od OSI sloja dva (OSI 2) na više. Dakle za sve mrežne pakete se za njihovo prosljeđivanje pregledava samo zaglavlje na OSI sloju dva (*MAC* adrese), bez obzira što se nalazilo u višim slojevima paketa.

## Konfiguracija mrežnog mosta (*bridge-a*).

Za konfiguraciju mrežnog mosta (*bridge*) obično imamo dostupna dva alata odnosno programa: `brctl` i `ip`.

Pri tome naredba `brctl` dolazi u paketu `bridge-utils` dok naredba `ip` dolazi u paketu `iproute` i obično je već instalirana na sustav. Naredba `brctl` zamjenjuje stariju naredbu `brcfg` koja je imala sâmo osnovne mogućnosti konfiguriranja mrežnog mosta. U svakom slučaju i `brctl` i `ip` koriste se za administraciju mrežnog mosta s time da je naredba `ip` novija i koristit ćemo ju češće. U istom paketu s naredbom `ip` dobivamo i naredbu `bridge`.
Instalirajmo oba softverska paketa (za ***RedHat/CentOS 6.x – 7.7).*** [Od ***v.7.7.*** `bridge-utils` je izbačen iz uporabe]:

```
yum -y install iproute bridge-utils
```

Međutim za ispravno funkcioniranje mrežnog mosta (***bridge***) potreban nam je i kernel modul za tu namjenu; ponekad je već učitan, ali mi ćemo ga ipak učitati ručno, s naredbom `modprobe` na sljedeći način:

```
modprobe bridge
```

Provjerimo je li se sada učitao navedeni kernel modul:
```
lsmod | grep bridge

bridge                 151336  0
stp                     12976  1 bridge
llc                     14552  2 stp,bridge
```
Vidimo s lijeve strane (`bridge`) što znači kako se uredno učitao te kako su se uz njega učitala i dva druga kernel modula:

- `stp` – koji je zadužen za ***STP*** (*Spanning Tree*) protokol.
- `llc` – koji je zadužen za ***LLC*** (*Logical link control*) komunikaciju *(na nižoj mrežnoj razini).*



Za trajno učitavanje *kernel modula* proučite poglavlje:
**11.1.1.2. Napredno: Rad s kernel modulima.**

***Važno je znati da dodavati sljedeća mrežna sučelja unutar bridge-a odnosno sučelja mrežnog mosta:***

- *MOŽEMO:*
  - *Fizička mrežna sučelja poput:* **eth0**, **eth1**, **eno0**, **eno1**, **enp0s1**, **enp0s2**, …
  - *Agregirana/team/bond mrežna sučelja poput:* **bond0**, **bond1**, **bond2**, …
  - *VLAN mrežna sučelja poput:* **eth0.40**, **eth1.20**,**bond0.20**,**bond1.40**
  - *Virtualna mreža sučelja poput:* **tun0**, **tun1**, **tap2**, **veth0**, **veth1** …
- *NE MOŽEMO:*
  - *Pôd sučelja fizičkih mrežnih sučelja, primjerice:* **eth0:1**, **eth0:2**, **eno0:1**, **enp0s1:1**, …
  - *Pôd sučelja agregiranih/team/bond mrežnih sučelja poput:* **bond0:1**, **bond0:2**,**bond1:1**, **bond2:1**, …
  - *Pôd sučelja od VLAN sučelja:* **eth0:1.40**, **eth1:1.20**,**eno1:2.20**,**enp0s1:2.40**, …

---

## 20.6.1.1. Privremena konfiguracija *bridge* mrežnih sučelja

U ovoj cjelini proći ćemo kroz nekoliko primjera; od kreiranja i rada do upotrebe mrežnih mostova (*bridge-va*) za različite namjene.

**1. U ovom primjeru kreirat ćemo *bridge* sučelje `br0` na koje ćemo spojiti fizička mrežna sučelja `eth0` i `eth1`.**
Kreirajmo *bridge* mrežno sučelje (`br0`) upotrebom naredbe `brctl` na sljedeći način:
```
brctl addbr br0
```

Isto možemo napraviti i s novijom `ip` naredbom na sljedeći način (za ***RedHat/CentOS* 7**.x ili novije):
```
ip link add name br0 type bridge
```

*Bridge sučelje `br0` bi mogli obrisati sa sljedećom naredbom (**nemojte ovo napraviti**):*
```
ip link del br0
```

Sada aktivirajmo ovo novo `br0` mrežno sučelje upotrebom novije naredbe `ip` na sljedeći način:
```
ip link set dev br0 up
```

Pogledajmo naše novo mrežno sučelje (`br0`) s naredbom `ifconfig`:
```
ifconfig

br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::ace3:8fff:fee5:27e7  prefixlen 64  scopeid 0x20<link>
        ether ae:e3:8f:e5:27:e7  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 656 (656.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```
Isto smo mogli vidjeti i s naredbom: `ip link show`. Ili još detaljnije (za sva mrežna sučelja): `ip -d a`.
Na ispisu vidimo naše novo `br0` mrežno sučelje aktivno, ali još uvijek nije povezano s drugim mrežnim sučeljima.

Ukoliko koristimo mrežni most (*bridge*) u pravilu se IP parametri definiraju na razini sučelja mrežnog mosta; u našem slučaju je to mrežno sučelje `br0`, a ne na razini fizičkih mrežnih sučelja (pr. `eth0` ili `eth1`). Stoga moramo obrisati IP parametre na mrežnim sučeljima `eth0` i `eth1`, a tek onda konfigurirati IP parametre za `br0` sučelje.

Za brzinsko brisanje svih IP parametara mrežnih sučelja eth0 i eth1 pokrenimo sljedeće naredbe:

```
ip address flush dev eth0
ip address flush dev eth1
```

Sada ćemo dodijeliti IP adresu za mrežno sučelje br0.
Ako želimo konfigurirati IP: 192.168.1.10 uz masku mreže (*netmask*): 255.255.255.0), tada ćemo napraviti:

```
ip address add 192.168.1.10/24 dev br0
```

Sada pogledajmo novu konfiguraciju (skratili smo ispis samo za br0 sučelje):

```
ifconfig
```

```
br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.20.1.185  netmask 255.255.255.0  broadcast 0.0.0.0
```

Kako bi se mrežni paketi prosljeđivali između mrežnih sučelja moramo uključiti prosljeđivanje odnosno *forwarding* paketa:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Sada dodajmo fizička mrežna sučelja eth0 i eth1 u *bridge* sa br0 mrežnim sučeljem. To ćemo postići s naredbom ip:

```
ip link set dev eth0 master br0
ip link set dev eth1 master br0
```

Pogledajmo koja mrežna sučelja imamo unutar *bridge* sučelja br0  (skratili smo ispis):

```
ip link show master br0
```

```
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state FORWARDING
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state FORWARDING
```

Vidimo kako je napravljeno upravo ono što smo htjeli: eth0 i eth1 su u *bridgeu* sa br0 mrežnim sučeljem (master br0).

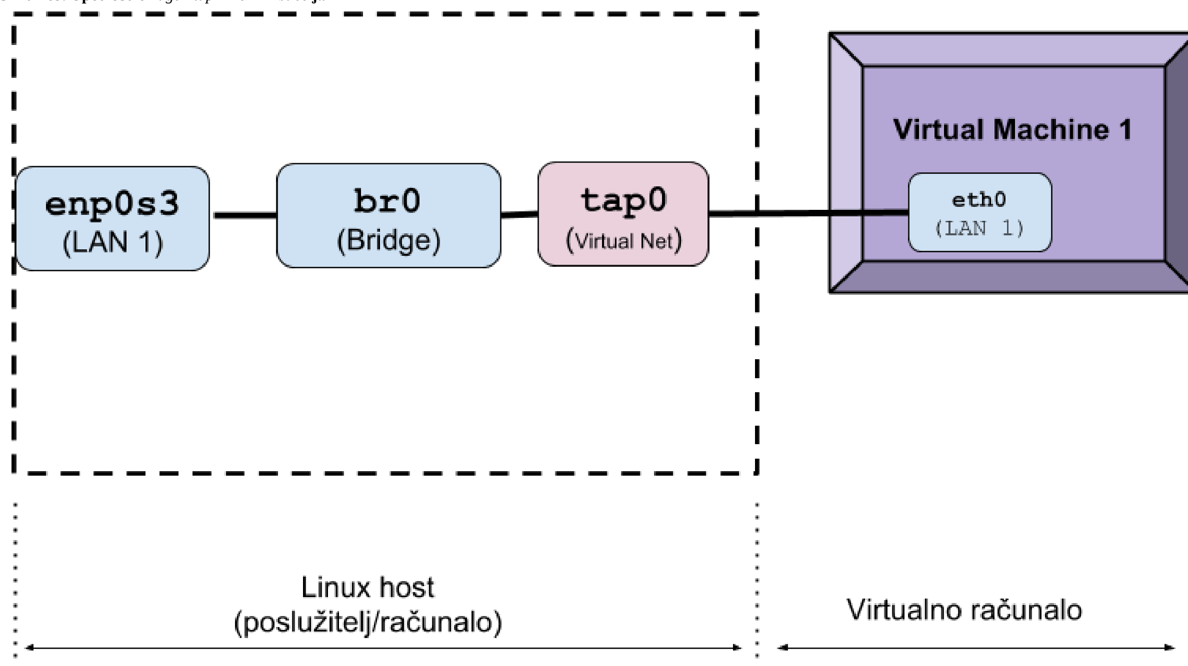
Za brisanje/micanje mrežnih sučelja unutar *bridge*-a, koristimo naredbu: ***brctl delif*** *BR-PORT INTERFACE.*
Ako bi pak s novijom naredbom ip htjeli maknuti sučelje eth1 iz br0 *bridgea*, tada bi trebali pokrenuti naredbu:

```
ip link set dev eth1 nomaster
```

**2. Upotreba mrežnih mostova (*bridgeva*) u virtualizaciji.**

**U ovom primjeru kreirat ćemo *bridge* sučelje br0 na koje ćemo spojiti fizičko mrežna sučelja enp0s3 i virtualno mrežno sučelje tap0**. Na taj način će virtualno računalo biti direktno vidljivo na mreži zajedno s fizičkim računalom koje ima fizičku mrežnu karticu (enp0s3). Dakle unutar iste IP mreže. Ovakav način rada se zove ***bridge network***.
Stoga želimo slijedeći scenarij, kako je vidljivo na slici **169**.

**Slika 169. Upotreba** ***bridge*** **i** ***tap*** **mrežnih sučelja**




Ovdje vidimo fizičko mrežno sučelje enp0s3 fizičkog računala, koje se preko br0 ***bridge*** mrežnog sučelja povezuje sa tap0 virtualnim mrežnim sučeljem. Na tap0 virtualno mrežno sučelje možemo potom spojiti virtualno računalo (*virtualku*) odnosno virtualnu mrežnu karticu virtualnog računala (*virtualke)* preko ***QEMU*** emulatora ***i KVM*** hipervizora.

Bridge mrežno sučelje je posebno mrežno sučelje, koje radi na OSI sloju dva (OSI 2). To znači da ono pri dohvaćanju i prosljeđivanju (preklapanju) mrežnih paketa od sebe i prema sebi, za svaki mrežni paket pregledava samo njegovo zaglavlje na OSI sloju dva odnosno **MAC** adrese. To isto tako znači da *bridge* mrežno sučelje ne zanima što se nalazi na višim slojevima mrežnih paketa (pr. ***IP***, ***TCP***, ***UDP***, …) jer će ih on proslijediti dalje samo na osnovi **MAC** adrese, bez obzira što pojedini paket sadržavao na višim slojevima mreže. Stoga ovakav način rada ima vrlo široku primjenu jer se sa zaglavljem i podacima koje mrežni paketi nose na višim slojevima mreže može pozabaviti neka druga komponenta (pr. virtualizacija/hipervizor). Važno je razumjeti da nakon što *bridge* sučelje odluči kamo sa svakim pojedinim mrežnim paketom, šalje ga prema ***Netfilter*** mrežnom podsustavu Linuxa, koji ovisno o postavkama vatrozida, paket može proslijediti dalje ili odbaciti.



Za detalje o ***Netfilter*** podsustavu i *vatrozidu* (*firewallu*) pogledajte poglavlje:
**26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije.**

Vratimo se na kreiranje *bridge* mrežnih sučelja. Kreirajmo jedno *bridge* mrežno sučelje imena: `br0`:
```
ip link add name br0 type bridge
```
Sada aktivirajmo novo kreirano `br0` mrežno sučelje:
```
ip link set dev br0 up
```
I ovdje pošto koristimo mrežni most (*bridge*), IP parametri se definiraju na razini sučelja mrežnog mosta; u našem slučaju je to mrežno sučelje `br0`, a ne na razini fizičkih mrežnih sučelja (pr. `enp0s3`, `eth0` ili `eth1` i slično).

Sada kada je naše *bridge* sučelje aktivirano pogledajmo što možemo doznati i o njemu (skratili smo ispis):
```
ethtool -i br0
driver: bridge
version: 2.3
```
Vidimo da je vrsta sučelja ***bridge*** te vidimo njegovu inačicu. Prema potrebi, moguće je raditi razne optimizacije ili prilagodbe načina rada *bridge* sučelja (pr. `br0`), preko *sysctl* sistemskih varijabli. Na sljedeći način možemo izlistati sve *sysctl* varijable koje se tiču ovog sučelja.

Za detalje o njima pogledajte izvor informacija (**642**).
```
sysctl -a | grep -i br0
```

Odnosno sve opcije su vidljive preko *sysfs* pseudo datotečnog sustava, kao posebne datoteke unutar direktorija (mape):
```
ls -al /sys/class/net/br0/bridge/
```

U ovom koraku ćemo s našeg fizičkog mrežnog sučelja `enp0s3` obrisati IP parametre, a onda ćemo konfigurirati iste IP parametre za `br0` sučelje. Brzinsko brisanje svih IP parametara mrežnog sučelja `enp0s3` možemo napraviti ovako:
```
ip address flush dev enp0s3
```
Sada ćemo dodijeliti IP adresu za mrežno sučelje `br0`.
Ako želimo konfigurirati IP: 192.168.1.10 uz masku mreže (*netmask*): 255.255.255.0, tada ćemo morati napraviti sljedeće:
```
ip address add 192.168.1.10/24 dev br0
```



*Kasnije moramo riješiti i podrazumijevani usmjerivač (**default gateway**) na ovom mrežnom sučelju, ali pošto ovdje sve testiramo, lokalno (u lokalnoj mreži) će sve raditi i bez njega.*

I u ovom primjeru, kako bi se mrežni paketi prosljeđivali između mrežnih sučelja, moramo uključiti ***forwarding***:
```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Sada dodajmo fizičko mrežno sučelje `enp0s3` u *bridge* sa `br0` mrežnim sučeljem:
```
ip link set dev enp0s3 master br0
```

I konačno kreirajmo posebno virtualno ***tap*** mrežno sučelje imena `tap0` te ga aktivirajmo:
```
ip tuntap add tap0 mode tap
ip link set dev tap0 up
```
Dodajmo i `tap0` sučelje u naš *bridge* `br0`:
```
ip link set dev tap0 master br0
```

Pogledajmo što sada imamo u *bridge-u* odnosno povezano sa *bridge* sučeljem `br0`: (skratili smo ispis)
```
ip link show master br0
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP
4: tap0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP
```

Vidimo kako su u *bridge-u* `br0` povezana: fizičko mrežno sučelje `enp0s3` i virtualno mrežno sučelje `tap0`.

Za ispis statusa svih mrežnih sučelja s opisom o vrsti sučelja (***bridge***, ***TUN***, ***TAP***, …), možemo koristiti malo poznati prekidač:
```
ip -d a
```

***Slijedi virtualizacija i povezivanje s bridge mrežnim sučeljem.***



Pretpostavka je da sve ovo radimo na fizičkom računalu koje ima uključenu virtualizaciju u ***BIOS***-u računala.

Provjerimo imamo li podršku za virtualizaciju:

```
cat /proc/cpuinfo | grep -E "vmx|svm"
```

Ako imamo `vmx` (*Intel*) ili `svm` (*AMD*) zastavice na procesoru, sve je u redu i možemo nastaviti.

Sada moramo instalirati dodatne softverske pakete vezane za virtualizaciju:

```
yum install -y qemu kvm kmod-kvm qemu-kvm qemu-img
```

Za ***KVM*** (Virtualizaciju na razini kernela), potrebno je učitati i kernel module za tu namjenu:

```
modprobe kvm
```

Ovaj kernel modul daje nam pristup *KVM* funkcionalnostima, ali se oslanja na hardversku podršku, koja ovisi o tome koji procesor (CPU) imamo, pa tako za *Intel CPU* moramo koristiti `kvm-intel` dok za *AMD CPU imamo* `kvm-amd`.
Mi imao *Intel CPU*, pa ćemo učitati sljedeći kernel modul:

```
modprobe kvm-intel
```

Pogledajmo sada vezu između KVM kernel modula koje smo učitali:

```
lsmod | grep kvm

kvm_intel              183720  0
kvm                    578558  1 kvm_intel
irqbypass               13503  1 kvm
```

Dakle `kvm` kernel modul se oslanja na `kvm-intel` kernel modul, preko kojega se koristi virtualizacija unutar CPU-a.

Unutar direktorija `/VIRTUALKE` ćemo kreirati disk za virtualno računalo, veličine 10GB:

```
cd /VIRTUALKE
qemu-img create -f qcow2 vm-harddisk.qcow2 10G
```

Sada kada imamo disk za virtualno računalo idemo dalje i možemo pokrenuti našu *virtualku* sa sljedećom naredbom:

```
qemu-system-x86_64 -enable-kvm -cpu host -smp 1 -runas qemu -daemonize -vnc 127.0.0.1:10111 \
-drive file=/VIRTUALKE/qcow2 vm-harddisk.qcow2,index=0,cache=none,aio=threads,if=virtio \
-boot d -net nic,model=virtio,macaddr=00:00:00:01:02:03 -net tap,ifname=tap0 \
-balloon virtio -m 512 -monitor telnet:127.0.0.1:5801,server,nowait
```

Navedeni način pokretanja virtualnog računala ovisi o inačici *Qemu/KVM*, jer se ponekada neke od opcija dodaju, a neke izbace.

U svakom slučaju, opisat ćemo samo neke od njih:

- `-enable-kvm` - uključujemo podršku za virtualizaciju na razini kernela i hardvera (podrška unutar CPU-a).
- `-drive file=/VIRTUALKE/qcow2 vm-harddisk.qcow2,index=0,cache=none,aio=threads,if=virtio` - označava disk virtualnog računala, njegovu vrstu (*virtio*) i način rada (*cache=none,aio=threads*).
- `-net nic,model=virtio,macaddr=00:00:00:01:02:03` - označava virtualnu mrežnu karticu koju će dobiti virtualno računalo, njenu vrstu (*virtio*) i ***MAC*** adresu koju će dobiti (*00:00:00:01:02:03*) zatim slijedi
- `-net tap,ifname=tap0` - označava kako će se virtualna mrežna kartica virtualnog računala spajati na TAP mrežno sučelje i to na naše `tap0` mrežno sučelje (koje smo malo prije kreirali).



Za više detalja o ***TUN/TAP*** mrežnim sučeljima pogledajte poglavlje:
**20.6.3. TUN i TAP - posebna mrežna sučelja.**

Za detalje o virtualizaciji pogledajte i poglavlje:
**27.1. Virtualizacija.**

Umjesto klasičnog *Linux bridge* sučelja, moguće je koristiti i ***Open vSwitch bridge*** sučelje.
Pogledajte poglavlje:
**26.8.5. Open vSwitch servis.**

**Izvor informacija: (616),(617),(618),(1016),(1019),(1020), (1308),(1309),(1310),(1342),(K-6)** `man brctl`, `man ip`, `man modprobe`, `man lsmod`, `man bridge`, `ip link help bridge`.

### 20.6.1.2. Trajna (permanentna) konfiguracija *bridge* mrežnih sučelja

U slučajevima kada je potrebno napraviti trajnu konfiguraciju mrežnih mostova (*bridge-va*), potrebno je napraviti sljedeće.



Pogledajte prvo poglavlje za trajnu konfiguraciju mrežnih sučelja:
**25.6.5.1.2. Trajna (permanentna) ručna konfiguracija mreže.**

Pogledat ćemo konfiguraciju koju smo opisali u primjeru broj **1**. iz prijašnjeg poglavlja. Dakle slučaj u kojem imamo fizička mrežna sučelja `eth0` i `eth1` koja dodajemo u mrežni most (*bridge*) odnosno u njegovo mrežno sučelje imena `br0`.
Kako bi naša konfiguracija bila trajna (i nakon restarta računala) potrebno je za mrežno sučelje `br0` kreirati datoteku imena: `/etc/sysconfig/network-scripts/ifcfg-br0` koja mora sadržavati minimalno sljedeće redove konfiguracije:

```
DEVICE=br0
TYPE=Bridge
STP=off
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.1.10
NETMASK=255.255.255.0
NM_CONTROLLED=no
```

U ovoj konfiguraciji definirali smo parametre rada mrežnog mosta odnosno njegovog sučelja `br0`:

- `DEVICE=br0` – definira ime mrežnog sučelja `br0`.
- `TYPE=Bridge` – označava da se radi o ***bridge*** uređaju odnosno posebnom mrežnom sučelju koje radi na OSI 2 sloju.
- `STP=off` – označava kako isključujemo ***STP*** (*Spanning Tree* protokol).
- `ONBOOT=yes` – označava kako se ovo mrežno sučelje treba podiči tijekom pokretanja računala.
- `BOOTPROTO=static` – označava kako se konfiguracija IP parametara neće raditi preko ***DHCP***/***BOOTP*** protokola već ručno odnosno statički. Njena (statička) konfiguracija slijedi ispod.
- `IPADDR=192.168.1.10` – ovdje definiramo IP adresu ovog mrežnog sučelja.
- `NETMASK=255.255.255.0` – ovdje definiramo masku mreže (*netmask*) ovog mrežnog sučelja.
- `NM_CONTROLLED=no` – označava kako ovim uređajem ne može upravljati tzv. ***NetworkManager***.
  Za **RedHat/CentOS 8.x+** ovdje mora biti definirano `=yes` jer se **NetworkManager** mora koristiti!



Pogledajte poglavlje*:*
**25.6.5.1.2. Trajna (permanentna) ručna konfiguracija mreže.**

A sada pogledajmo konfiguraciju `eth0` mrežnog sučelja u: `/etc/sysconfig/network-scripts/ifcfg-eth0`:

```
DEVICE=eth0
ONBOOT=yes
BRIDGE=br0
BOOTPROTO=none
NM_CONTROLLED=no
```

Ovdje vidimo da ne definirano IP adresu na mrežnom sučelju `eth0` jer se ona (**IP**) definira u *bridge* sučelju.
Nove opcije vidljive ovdje su sljedeće:

- `DEVICE=eth0` – definira ime mrežnog sučelja.
- `BRIDGE=br0` – označava kako je ovo mrežno sučelje (`eth0`) unutar *bridge* mrežnog sučelja `br0`.
- `BOOTPROTO=none` – označava kako se za ovo mrežno sučelje (`eth0`) neće konfigurirati IP parametri statički niti preko ***DHCP***/***BOOTP*** poslužitelja jer se definiraju za mrežno sučelje `br0`.

I pogledajmo konfiguraciju mrežnog sučelja `eth1` u: `/etc/sysconfig/network-scripts/ifcfg-eth1`:

```
DEVICE=eth1
ONBOOT=yes
BRIDGE=br0
BOOTPROTO=none
NM_CONTROLLED=no
```

Ovdje isto vidimo da ne definirano IP adresu jer se ona definira u *bridge* sučelju, a nećemo objašnjavati već objašnjene opcije:

- `DEVICE=eth1` - definira ime mrežnog sučelja.
- `BRIDGE=br0` - označava kako je ovo mrežno sučelje (`eth1`) unutar *bridge* mrežnog sučelja `br0`.
- `BOOTPROTO=none` – označava kako se niti za ovo mrežno sučelje (`eth1`) neće konfigurirati IP parametri statički niti preko ***DHCP***/***BOOTP*** poslužitelja jer se definiraju za mrežno sučelje `br0`.



Pogledajte i primjere konfiguracije ***bridge*** mrežnog sučelja u virtualizaciji:
**27.1.2.1. Konfiguracija mreže.**

Pogledajte i primjere konfiguracije ***bridge*** mrežnog sučelja kod Linux kontejnera:
**27.2.1. Konfiguracija i upotreba Linux kontejnera.**

Pogledajte i primjere povezivanje ***VETH*** mrežnih sučelja i ***bridge*** sučelja:
**20.6.4. VETH - posebno mrežno sučelje.**



*Od linux kernela 3.8. moguće je na **Bridge** mrežnom sučelju definirati i **VLAN** filtriranje.*
*Pogledajte primjere u sljedećoj cjelini i u izvoru informacija (**1310**). O **VLAN**ovima pročitajte više u sljedećem poglavlju.*

**Izvori informacija: (615),(616),(617),(618),(619),(620),(1019),(1020),(1308),(1309),(1310),(1342),(K-6), te plakat P5.**

## 20.6.1.3. Bridge sučelje i VLAN filtriranje

**Slijedi napredna cjelina!**
Bridge sučelje povezuje dva *Ethernet* segmenta zajedno na način neovisan o višem protokolu poput TCP/IP.
To je tako jer se paketi prosljeđuju na temelju *Ethernet* adresa (***MAC*** adresa), a ne IP adresa, kao što to rade usmjerivači.
Budući da se prosljeđivanje vrši na razini OSI sloja dva (OSI 2,) svi viši protokoli (TCP/IP) mogu proći transparentno kroz mrežni most (*bridge*). Linux *bridge* je zapravo virtualni preklopnik (*switch*), a često se koristi u virtualizaciji s KVM/QEMU *hipervizorom*, te mrežnim imenskim prostorima (*network namespaces*) i slično.

S druge strane VLAN (*Virtual LAN*) tehnologija također igra važna ulogu u virtualizaciji. Ona omogućuje grupiranje virtualnih računala u jednu mrežnu cjelinu čak i ako fizička računala nisu na istom fizičkom preklopniku.
To znači da je upotrebom VLANova moguće razdvojiti virtualna računala u izolirane mreže.



Za više detalja o **VLAN** mrežama i **802.1Q** protokolu pogledajte sljedeće poglavlje.

Međutim da bi *bridge* sučelje povezali sa željenom **VLAN** mrežom, bilo je potrebno ili na razini ***bond*** sučelja kreirati VLAN sučelje: primjerice **bond0.20** ili na razini fizičkog sučelja (pr. **eth2**) kreirati VLAN sučelje: primjerice: eth2.20.

Zatim je takva sučelja potrebno dodati u *bridge* sučelje, primjerice u br1 te na kraju isto *bridge* sučelje povezati s **TAP** sučeljem koje se dodjeljuje virtualnom računalu.

Ovakav način rada je očito kompleksan, naročito ako imate više VLAN mreža i veći broj virtualnih računala na poslužitelju.
Ovakav način rada je vidljiv na gornjem djelu slike **169.1**. Srećom u kernelima novijim od **3.8.** uvedena je značajka korištenja VLAN filtera (***VLAN filter***), kako je vidljivo u donjem djelu iste slike.



Sa značajkom filtriranja VLAN-ova na *bridge* sučelju, moguće je na *bridge* sučelju direktno koristiti VLAN mreže bez posebnih VLAN sučelja.

Ovakav način rada je praktično tzv. ***trunk VLAN*** način rada (***tagged***) u kojem ***bridge*** sučelje ne dira VLAN oznake (802.1Q oznake) već ih prema potrebi može filtrirati na TAP sučeljima, skidajući VLAN oznake pripadnosti željenoj VLAN mreži tek na TAP sučelju. Tada TAP sučelje radi u tzv. ***access VLAN*** načinu rada (znano i kao ***untagged***).

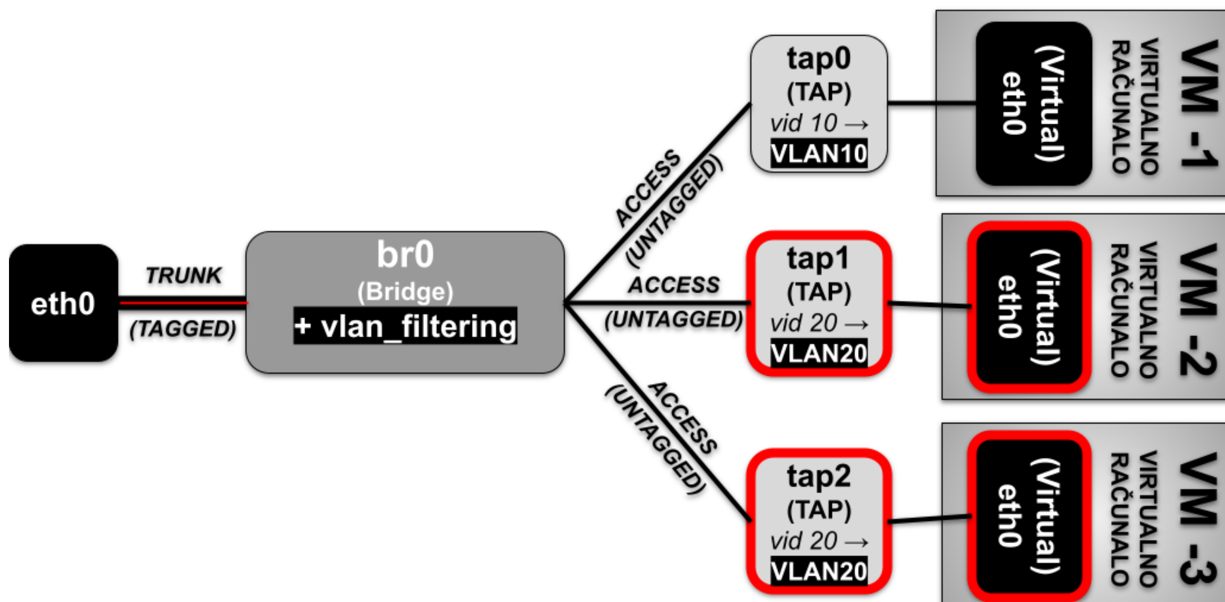
**Slika. 169.1. Bond, bridge i VLAN sučelja te upotreba VLAN mreža.**

eth0
eth1
bond0 (Bond)
bond0.10 (VLAN10 sučelje)
br0 (Bridge)
tap0 (TAP)
VM -1 VIRTUALNO RAČUNALO (Virtual) eth0
bond0.20 (VLAN20 sučelje)
br1 (Bridge)
tap1 (TAP)
VM -2 VIRTUALNO RAČUNALO (Virtual) eth0
Bond ili eth VLAN
eth2
eth2.20 (VLAN20 sučelje)
br2 (Bridge)
tap2 (TAP)
VM -3 VIRTUALNO RAČUNALO (Virtual) eth0
KERNEL <3.8

eth0
eth1
bond0 (Bond)
br0 (Bridge) + vlan_filtering
tap0 (TAP) vid 10 → VLAN10
tap1 (TAP) vid 20 → VLAN20
tap2 (TAP) vid 20 → VLAN20
VM -1 VIRTUALNO RAČUNALO (Virtual) eth0
VM -2 VIRTUALNO RAČUNALO (Virtual) eth0
VM -3 VIRTUALNO RAČUNALO (Virtual) eth0
KERNEL >3.8

Krenimo s primjerima upotrebe *bridge* sučelja s novijom opcijom ***VLAN filter*** (kernel >3.8) kao na slici 169.2.

**Slika. 169.2. bridge sučelje s VLAN filterom.**



Kreirajmo jedno *bridge* mrežno sučelje imena: `br0`:

```
ip link add name br0 type bridge vlan_filtering 1
```

Aktivirajmo novo kreirano `br0` mrežno sučelje:

```
ip link set dev br0 up
```

Za detalje odnosno dodatne informacije o ovom sučelju, možemo pokrenuti:

```
ip -d link show br0
```

Sada ćemo dodati našu fizičku mrežnu karticu (`eth0`) u *bridge* (`br0`):

```
ip link set eth0 master br0
```

A sada kreiramo i dodajemo TAP sučelja na koja će se spajati virtualna računala (virtualna računala nećemo konfigurirati)

```
ip tuntap add tap0 mode tap
ip tuntap add tap1 mode tap
ip tuntap add tap2 mode tap
```

Sada ćemo sva TAP sučelja dodati u *bridge:*

```
ip link set tap0 master br0
ip link set tap1 master br0
ip link set tap2 master br0
```

U istom softverskom paketu (`iproute`) s naredbom `ip` dobivamo i naredbu `bridge`, koju ćemo sada i koristiti.
Tek sada ćemo za TAP sučelja kreirati pripadnosti VLAN-ovima, kao na slici:

```
bridge vlan add dev tap0 vid 10 pvid untagged master
bridge vlan add dev tap1 vid 20 pvid untagged master
bridge vlan add dev tap2 vid 20 pvid untagged master
```

I sada pogledajmo što smo napravili, prema svim sučeljima koja smo konfigurirali (uz naše komentare nakon znaka #):

```
bridge vlan show
port              vlan-id
eth0                1 PVID Egress Untagged #--> PVID 1 (VLAN 1)  [NATIVE VLAN]
br0                 1 PVID Egress Untagged #--> PVID 1 (VLAN 1)  [NATIVE VLAN]
tap0                1 Egress Untagged
                   10 PVID Egress Untagged  #--> PVID 10  (pripadnost VLAN 10)
tap1                1 Egress Untagged
                   20 PVID Egress Untagged #--> PVID 20  (pripadnost VLAN 20)
tap2                1 Egress Untagged
                   20 PVID Egress Untagged #--> PVID 20  (pripadnost VLAN 20)
```

**VLAN 1** (*vlan-id 1* [PVID]) je standardno u upotrebi kao *untagged* jer je on uobičajen kao tzv. **native** VLAN.
To sve zajedno znači kako kroz sučelja: `eth0` i `br0` prolaze svi VLAN paketi (dakle oni su u ***Trunk*** načinu rada), a jedino se za VLAN 1 skida VLAN oznaka. Dok je za ***TAP*** sučelja važno primijetiti da `tap0` koristi VLAN 10 mrežu (*Access/Untagged*), dok `tap1` i `tap2` pripadaju VLAN 20 mreži (*Access/Untagged*), kako smo i htjeli.

***VLAN** oznake se mogu na ovaj način, osim **TAP** sučeljima, dodavati i za druga logička sučelja poput **VETH** i sličnih.*

**Bridge i TAP sučelja i MAC adrese**

Na kraju pogledajmo sva navedena sučelja po pitanju MAC adresa:

```
ip -brief link show
eth0             UP             08:00:27:ba:82:09 <BROADCAST,MULTICAST,UP,LOWER_UP>
br0              UP             08:00:27:6c:a4:71 <BROADCAST,MULTICAST,UP,LOWER_UP>
tap0             UP             c2:16:94:f8:70:3d <BROADCAST,MULTICAST>
tap1             UP             9e:26:74:f3:c8:79 <BROADCAST,MULTICAST>
tap2             UP             ce:cf:f5:a4:03:da <BROADCAST,MULTICAST>
```

Ovdje vidimo sljedeće:

- `eth0` - ovo je fizička mrežna kartica koja ima svoju MAC adresu ugrađenu od strane proizvođača (njena MAC adresa je: `08:00:27:ba:82:09`).
- `br0` - ovo je ***Bridge*** mrežno sučelje koje ima svoju jedinstvenu MAC adresu (`08:00:27:6c:a4:71`).
- `tap0` - ovo je ***TAP*** mrežno sučelje koje također ima svoju jedinstvenu MAC adresu (`c2:16:94:f8:70:3d`).
- `tap1` - ovo je ***TAP*** mrežno sučelje koje također ima svoju jedinstvenu MAC adresu (`9e:26:74:f3:c8:79`).
- `tap2` - ovo je ***TAP*** mrežno sučelje koje također ima svoju jedinstvenu MAC adresu (`ce:cf:f5:a4:03:da`).

**Bridge** sučelje (`br0`) je sučelje koje radi na OSI sloju dva (OSI 2) pa i ono mora imati svoju jedinstvenu MAC adresu, kako bi moglo raditi preklapanje mrežnih paketa na osnovu njihovih MAC adresa (pošiljatelja i odredišta). Nadalje, **Bridge** sučelje ima vezu prema fizičkoj mrežnoj kartici (`eth0`) s jedne strane, a s druge prema **TAP** sučeljima, koja su također sučelja koja rade na OSI sloju dva (OSI 2) pa i ona moraju imati svoje jedinstvene MAC adrese. U konačnici komunikacija od fizičke mrežne kartice do **Bridge** sučelja, pa preko **TAP** sučelja sve do virtualnih mrežnih kartica virtualnih računala prolazi kroz **Bridge** sučelje. Ono se ovdje ponaša kao klasičan mrežni most (***Bridge***) odnosno kao virtualni preklopnik, koji mrežne pakete prosljeđuje prema odredištu, provjeravajući njihove izvorišne i odredišne MAC adrese, a upravo stoga sva ova mrežna sučelja i moraju imati svoje jedinstvene MAC adrese.

**Izvori informacija: (1393),(1394),(1395),** `man ip`,`man ip link help`,`man bridge`,`ip link help bridge`.

## 20.6.2. VLANovi odnosno virtualne lokalne mreže

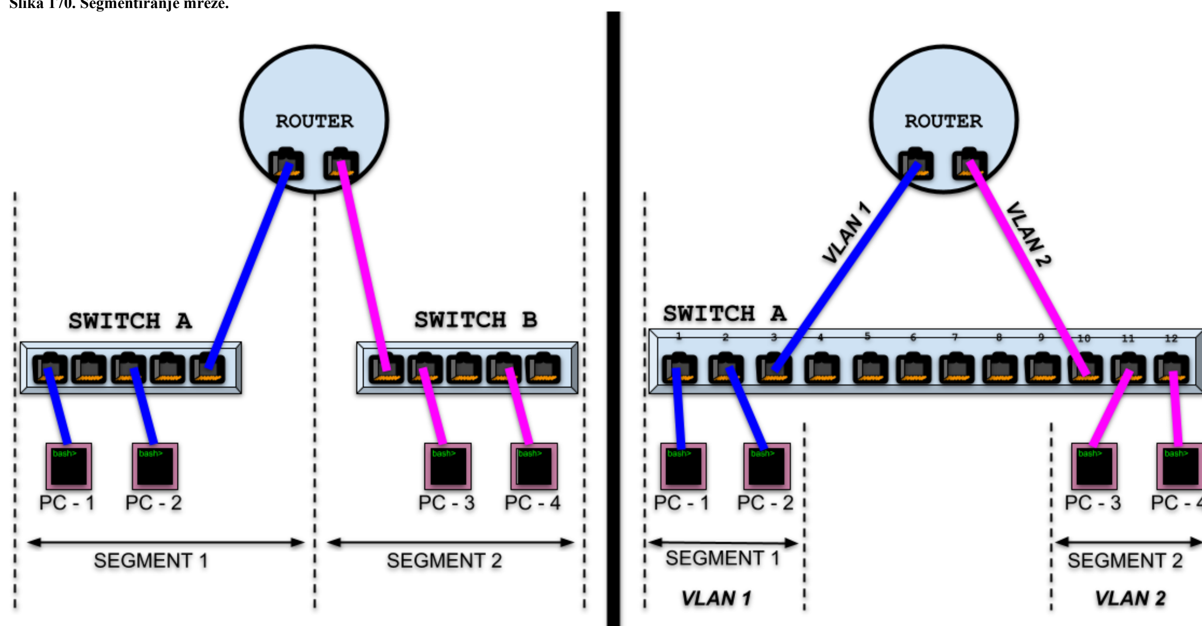
**Slijedi napredno poglavlje!**

Proces u kojemu razdvajamo veću ili kompleksniju mrežu na više odvojenih (izoliranih) dijelova odnosno segmenata mreže, naziva se **segmentacija**. Segmentiranje mreže se može napraviti na dva načina:

- ***Fizički***: jedan segment mreže - jedan preklopnik (engl. *switch*), ili više povezanih preklopnika u istoj mreži.
- ***Logički***: pomoću **VLAN** (engl. *Virtual Local Area Network*) mreža i to na jednom ili više preklopnika koji ih koriste.

Prema standardu **802.1Q**, definirana je podrška za virtualne lokalne mreže tj. **VLAN** (*Virtual LAN*) na ***ethernet*** mrežama pomoću kojih možemo raditi segmentaciju (izolaciju) mreža. Važno je znati kako je i *Broadcast* domena ograničena na **VLAN** mrežu, odnosno svaka **VLAN** mreža ima svoju Broadcast domenu i potpunu izolaciju od drugih mreža.
Pogledajmo sliku 170 na kojoj su vidljiva oba načina segmentacije mreže.

**Slika 170. Segmentiranje mreže.**




Na slici 170 je vidljiva fizička segmentacija (**lijevi dio slike**) te kako je kod ove metode segmentiranja svaki pojedini preklopnik unutar jednog segmenta mreže. Na **desnom** dijelu slike vidljivo je kako smo logičkom segmentacijom i to upotrebom **VLAN** mreža unutar jednog preklopnika dobili istu funkcionalnost kao kod fizičke segmentacije, ali smo dobili i puno više mogućnosti. Ovdje je također vidljivo kako smo sučelja (*portove)* (**1**,**2** i **3**) na preklopniku stavili u jedan segment odnosno ***VLAN*** *(VLAN 1)*, a sučelja (**10**,**11** i **12**) u drugi segment odnosno u drugi ***VLAN*** *(VLAN 2)*. Kako bi to sve uopće bilo moguće, na preklopniku smo prethodno kreirali dva navedena ***VLAN-a*** za navedena sučelja te sučelja stavili, svako u svoj pripadajući ***VLAN***.
U oba slučaja, direktna komunikacija je moguća sâmo i isključivo između računala u istim segmentima odnosno **VLAN**-ovima:

- **Segment 1**: PC-1 i PC-2.
- **Segment 2**: PC-3 i PC-4.

Mreža se obično segmentira zbog smanjenja *broadcast* domene, odnosno izolaciju segmenta mreže u kojoj se nalaze neke logičke cjeline mreže, pa tako u zasebne segmente mreže možemo stavljati:

- Poslužitelje.
- Mrežne pisače, skenere i druge slične uređaje.
- Računala iz zasebnih logičkih cjelina, odnosno različitih odjela tvrtke, koji ne zahtijevaju veliku komunikaciju, a uopće ne smiju imati pristup jedni s drugima, ili smiju imati samo ograničeni pristup, poput:
    - Razvojnih odjela i/ili testnih laboratorija.
    - Menadžmenta i marketinga.
    - Prodaje ili podrške kupcima i slično.

Samo usmjerivač (*router)* ili neki drugi mrežni uređaj koji radi na OSI sloju tri (OSI 3) može povezati dva ovakva segmenta mreže. Dakle međuveza između **VLAN** segmenata mreže se ne može napraviti na OSI sloju dva već na OSI sloju tri (**IP** *sloj*). Stoga se na usmjerivačima (**Router** na slici) i definiraju prava pristupa između raznih segmenata odnosno *VLAN*-ova mreže.

 Uočite kako se u oba modela primjene ***VLAN****ova,* svaki zasebni segment mreže mora povezati sa zasebnom vezom prema usmjerivaču, odnosno njegovom mrežnom sučelju (mrežnom karticom) koje je dedicirano zaduženo za točno definirani segment mreže. Rješenje ovog problema vidjet ćemo kasnije.

Standard **802.1Q** definira način označavanja *VLAN-ova* unutar mrežnih okvira (engl. *VLAN Tagging*) te naravno sposobnosti mrežnih uređaja da ih prepoznaju, znaju obrađivati i koristiti.
Dodatno, ***802.1Q*** standard nudi i prioritizaciju mrežnih okvira, prema standardu ***IEEE 802.1p***, za ***QoS*** sustav.

Za početak pogledajmo kako izgleda mrežni okvir na OSI sloju dva (***OSI 2***), koji ima podršku za *VLAN-ove*:

| DMAC | SMAC | 802.1Q Tag | Ether TYPE | DATA/PAYLOAD | FCS |
|---|---|---|---|---|---|
| Odredišna MAC | Izvorišna MAC | **802.1Q** oznaka | 0x8100 | DATA/PAYLOAD (Sve s gornjih slojeva) | ***CRC*** provjerni zbroj *(Checksum)* |
| 6 *bajta* | 6 *bajta* | 4 *bajta* | 2 *bajta* | od 46 do 1500 *bajta* | 4 *bajta* |

Identifikator koji govori sustavu da se radi o *VLAN* mrežnom paketu (**802.1Q**) zapisan je u ***Ether Type*** polje (konkretno je to heksadecimalna vrijednost ***0x8100***). Iako mrežni okvir izgleda slično klasičnom, on ima dodano polje u koje se umeću informacije o *VLAN* mrežama (**802.1Q *Tag*** *[VLAN]*), a koje je veličine 4 *bajta*. S tim poljem mrežni okvir u odnosu na klasični raste za ta 4 *bajta*. Standard kojim je definirana ova nova veličina mrežnih okvira je **IEEE 802.3ac**. Dakle prema ovom standardu maksimalna veličina (standardnog) mrežnog okvira prema tome raste s 1518 na 1522 *bajta*. Mrežni uređaji koji ne podržavaju ovaj standard najčešće će procesirati (obraditi) ove mrežne okvire, ali bez *VLANova*, i oni (ovakvi mrežni paketi) će biti vidljivi u statistikama kao ***baby giant*** anomalije.

**Kako sada izgledaju najmanji mogući Ethernet mrežni okviri, koji sadrže samo 46 bajta podataka te koriste VLAN-ove?**
Pogledajmo izgled najmanjih mrežnih okvira na drugom sloju mreže (***Layer 2 tj. OSI 2***) s podrškom za **VLAN**:

| DMAC | SMAC | 802.1Q Tag | ETHER TYPE | DATA/PAYLOAD | FCS |
|---|---|---|---|---|---|
| *Odredišna* MAC | *Izvorišna* MAC | **802.1Q** oznaka | 0x8100 | DATA/PAYLOAD (Sve s gornjih slojeva) | ***CRC*** provjerni zbroj *(Checksum)* |
| 6 *bajta* | 6 *bajta* | 4 *bajta* | 2 *bajta* | 46 *bajta* | 4 *bajta* |
| ← U k u p n o 68 b a j t a → | | | | | |

Sve zajedno, kada se spušta na OSI sloj jedan (***Layer 1*** *tj.* ***OSI 1***) izgleda ovako:

| Preambula | SFD | Sve s gornjeg sloja | Interframe gap (razmak) |
|---|---|---|---|
| Označava početak okvira | ***Start Frame Delimiter*** naznačava da nakon njega dolaze podaci → | Sadrži sve s gornjeg sloja mreže (***Layer*** **2** tj. **OSI 2**) | Označava razmak između mrežnih paketa/okvira |
| 7 *bajta* | 1 *bajt* | 68 *bajta* | 12 *bajta* |
| ← U k u p n o 88 b a j t a → | | | |



*U posebnim slučajevima, moguće je koristiti i tehnologiju dvostrukog ugnježđivanja (enkapsulacije) tzv.* ***QinQ****, kako je opisano u* ***802.1ad.*** *Za detalje pogledajte izvor informacija:* **(1066)** *ili sljedeća poglavlja***.**

**Izvori informacija: (60),(386),(1066),(1194),(1195),(K-6),(K-10)**

## 20.6.2.1. Što se događa kod upotrebe VLANova

Upotrebom *VLAN-ova* razdvajamo lokalnu mrežu na više potpuno odvojenih odnosno izoliranih mreža, ako to naši mrežni uređaji omogućuju. Kako je vidljivo, pripadnost mrežnih okvira (paketa) pojedinim *VLAN-ovima* se upisuje unutar svakog mrežnog okvira na OSI sloju dva (OSI 2). Na OSI sloju dva rade i preklopnici, što je logično jer su upravo preklopnici uređaji koji se brinu o *VLAN* mrežama, iako i usmjerivači i drugi mrežni uređaji također prepoznaju *VLAN* mreže. Na preklopniku koji podržava *VLAN-ove* moguće je za svako pojedino mrežno sučelje (*interface/port*) na preklopniku, definirati njegovu pripadnost:

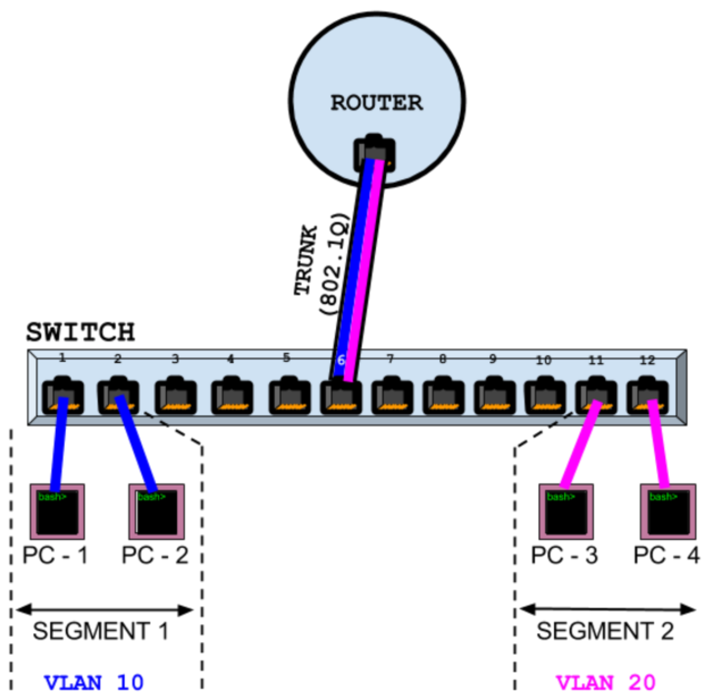
- Sâmo i isključivo jednom (pojedinom) *VLAN-u*, ako radi u takozvanom **ACCESS** način rada.
- Svim *VLAN-ovima*, ako radi u takozvanom **TRUNK** načinu rada.

Zamislimo da pojedino mrežno sučelje na preklopniku pripada točno definiranom *VLAN-u* odnosno ono se nalazi u tzv. ***ACCESS*** načinu rada, kao što je primjerice pripadnost *VLAN-u* broj 15. Tada na tom sučelju sa svakog mrežnog okvira koji je namijenjen za *VLAN* mrežu broj 15, preklopnik **MORA** iz mrežnog okvira (*paketa*) ukloniti *VLAN* polje (ono polje od 4 *bajta*) te ponovno izračunati provjerni zbroj pomoću **CRC** algoritma za cijeli mrežni okvir te novi provjerni zbroj ponovno upisati u **FCS** polje. Potom preklopnik kreira potpuno novi mrežni okvir bez *VLAN* polja odnosno potpuno normalni (obični) mrežni okvir i prebacuje ga na svoje odredišno mrežno sučelje koje pripada *VLAN*-u broj 15 odnosno prema računalu spojenom na njega. Dakle računalo spojeno na mrežno sučelje preklopnika koje pripada samo VLAN-u 15 (***ACCESS*** način rada sučelja) će zaprimiti mrežni paket bez VLAN oznake odnosno neće niti biti svjesno upotrebe VLAN-ova. Ovakav rad mrežnog sučelja preklopnika nazivamo i **untagged** (ne tagiran) odnosno neoznačen način rada, što znači bez VLAN oznake.

Svi drugi mrežni okviri koji sadrže bilo koji drugi *VLAN* broj osim broja 15 neće nikada (niti smiju) biti proslijeđeni na mrežno sučelje koje pripada *VLAN-u* broj 15. Sâmo mrežna sučelja koja pripadaju istom *VLAN-u* mogu prosljeđivati mrežne okvire među sobom. Moguće je i da taj mrežni okvir izlazi na posebno konfigurirano mrežno sučelje na preklopniku koje je u mogućnosti propuštati sve mrežne okvire sa svih *VLAN-ova*, a to su mrežna sučelja (portovi) koji rade u tzv. ***Trunk*** načinu rada (obično prema drugom preklopniku). Tada preklopnik za svaki pojedini mrežni okvir mora nadodati *VLAN* oznaku (polje) u mrežni okvir, te ponovno izračunati CRC provjerni zbroj i upisati ga u FCS polje mrežnog okvira. Tek potom preklopnik može (i mora) kreirati takav novi mrežni okvir i poslati ga na mrežno sučelje u ***TRUNK*** načinu rada.
Ova mrežna sučelja preklopnika koja propuštaju mreže pakete koji nose *VLAN* oznake (*tag*) se nazivaju i **tagged** sučelja odnosno sučelja koja propuštaju pakete s VLAN oznakama.

Pogledajmo kako to sve izgleda na slici 171. Naime na slici je vidljivo kako smo u odnosu na stariji model spajanja sada sve međuveze između preklopnika (*switch*) i usmjerivača (*router*), od kojih je svaka pojedina veza pripadala sâmo i isključivo jednom *VLAN-u*, zamijenili sa sâmo jednom međuvezom, koja je u mogućnosti propuštati sve *VLAN-ove* jer je ona u ***TRUNK*** načinu rada. Prema navedenoj terminologiji, na slici 171 imamo sljedeće konfigurirana mrežna sučelja (*portove)* na preklopniku:

**Slika 171. Upotreba *Access* i *Trunk* načina rada mrežnih sučelja (*portova*).**




- **ACCESS** način rada u kojem su trenutno konfigurirana mrežna sučelja: **1**, **2**, **11** i **12**, a pri tome je pripadnost tih sučelja različita:
  - **VLAN 10** mreži pripadaju sučelja: **1** i **2**.
  - **VLAN 20** mreži pripadaju sučelja: **11** i **12**.

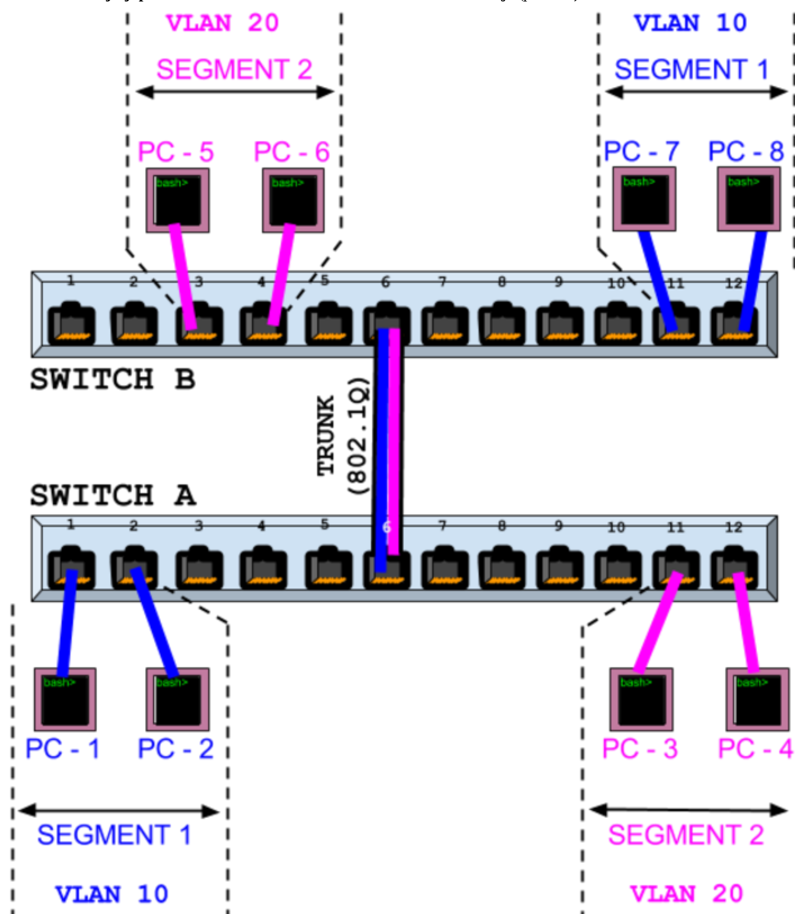
- **TRUNK** način rada u kojem je trenutno konfigurirano da njemu pripada mrežno sučelje **6**. Pri tome, pošto je ovo sučelje u *TRUNK* načinu rada, ono propušta sve *VLANove* odnosno mrežne pakete koji pripadaju svim VLANovima.

To znači kako kroz njega prolaze mrežni paketi koji nose VLAN oznake (*tag*) pripadnosti svih VLANova. Dakle kroz ovo mrežno sučelje (port) broj **6**, prolaze mrežni paketi sa **802.1Q** oznakama pripadnosti VLANovima kojima već pripadaju, te ih preklopnik ne mijenja (VLAN oznake) već ih samo gleda/čita i prosljeđuje dalje.

Dakle on svaki VLAN mrežni paket šalje ovisno o VLAN broju koji nosi. Tako će primjerice sve mrežne pakete s VLAN brojem **20** proslijediti samo na mrežna sučelja **11** i **12**, a one s VLAN brojem **10**, na mreža sučelja **1** i **2**.

Gledajući na proširenje broja povezanih preklopnika ovaj model je moguće i proširiti povezivanjem više preklopnika. Stoga ćemo izbaciti usmjerivač iz slike kako bi sve bilo razumljivije, pa pogledajmo novu shemu odnosno sliku 172 dolje.

**Slika 172. Detaljniji prikaz rada *Access* i *Trunk* načina rada mrežnih sučelja (*portova*).**




U ovom primjeru povezali smo dva preklopnika s međuvezom koja propušta sve *VLANove*, što znači kako smo na preklopniku (*Switch* ***A***), sučelje broj **6** konfigurirali kao ***TRUNK*** te smo istovremeno i preklopnik (*Switch* ***B***), odnosno njegovo sučelje broj **6** isto konfigurirali kao ***TRUNK*** port.

Ovime smo osigurali da će svi *VLANovi* kreirani na prvom preklopniku biti vidljivi i drugom preklopniku. Nakon toga smo sučelja na preklopnicima, na koja su spojena računala koja moraju komunicirati zajedno, stavili u iste *VLANove*, ali u ***ACCESS*** način rada.

Zbog takve pripadnosti VLAN-ovima je moguća direktna komunikacija sâmo između računala u istom *VLANu*:

- **VLAN 10**: PC1, PC2, PC7 i PC8
- **VLAN 20**: PC3, PC4, PC5 i PC6

Naravno u ovoj konfiguraciji bez usmjerivača, veza između ***VLAN 10*** i ***VLAN 20*** je **NEMOGUĆA**.

Ako bi htjeli komunikaciju između VLANova, naravno možemo ubaciti i usmjerivač, kao u primjeru prije (**Slika 171**), te ga pravilno konfigurirati.

***TRUNK*** način rada određenog mrežnog sučelja na mrežnom uređaju znači kako će kroz to sučelje prolaziti paketi koji će sadržavati *VLAN* oznake (Engl. ***VLAN Tag***) prema 802.1Q protokolu.

***ACCESS*** način rada pojedinog mrežnog sučelja u određenom ***VLANu*** na mrežnom uređaju znači kako će mrežni uređaj skinuti ***802.1Q*** oznaku na svakom pojedinom mrežnom paketu koji treba doći u taj VLAN (primjerice VLAN 10) prije nego ga mrežni uređaj pošalje na to mrežno sučelje i u konačnici na uređaj (pr. računalo) spojen na njega. To znači kako će na mrežnom sučelju u ***ACCESS*** načinu rada, završiti *obični* mrežni paketi koji nemaju nikakvu ***802.1Q*** oznaku i za koje računalo spojeno na to sučelje nije niti svjesno da je prije nego je paket došao na mrežno sučelje odnosno prije nego ga je preklopnik obradio, pripadao ikakvom *VLANu*. Dakle mrežni paketi koji dođu do ovog sučelja nemaju VLAN oznaku pa se nazivaju i ***Untagged***.

Pogledajte animaciju koja prikazuje upotrebu ***ACCESS*** (*Untagged*) i ***TRUNK*** (*Tagged*) sučelja na preklopniku

Možemo reći i ovako:

- ***TRUNK*** sučelje odnosno način rada propušta pakete koji pripadaju svim ***VLANovima*** (s ***VLAN Tag****-om [802.1Q]*).
- ***ACCESS*** sučelje skida ***VLAN*** oznake, te kreira i propušta **VLAN ne**označene pakete (***VLAN Untagged***).

U konačnici *VLANovi* donose mnoge prednosti, ali sa strane performansi mrežnih uređaja, a poglavito preklopnika, unose značajnu potrebu za snažnijim hardverom koji cijelu priču s baratanjem VLAN oznaka i izračunavanjem ***CRC*** provjernog zbroja i ponovnog sastavljanja svakog pojedinog paketa na mreži, mora moći odraditi bez usporavanja mreže. Dakle i ovdje se radi o milijunima ili desecima milijuna mrežnih okvira (paketa); ovisno o broju i brzinama mrežnih sučelja na preklopniku, u svakoj sekundi, a koje je potrebno dodatno procesirati (obraditi). Naravno to sve ima svoju cijenu. Snažniji preklopnici kao i snažnije mrežne kartice bez ikakvih usporavanja rade s *VLANovima*, a najsnažniji modeli mrežnih kartica hardverski mogu ubrzavati (*akcelerirati*) sve funkcije koje se tiču *VLANova*. Dakle korištenjem *VLANova* na pouzdanim i brzim preklopnicima, uz pretpostavku da smo pravilno segmentirali mrežu, mrežna komunikacija se može i ubrzati. To je zato što smo pravilnim segmentiranjem u svakom segmentu mreže ostavili one uređaje/računala koja najčešće i komuniciraju, te ne unose nepotreban mrežni promet za one druge, kojima taj promet nikada neće niti biti namijenjen ili uopće ne treba biti vidljiv.



*Upotrebom **Multilayer** preklopnika koji rade na OSI 2/3/4 slojevima te imaju **ASIC** sklopove, usmjeravanje između **VLAN** mreža se može drastično ubrzati.* *Pogledajte poglavlje: **20.4.2. Mjerenje propusnosti sa ASIC sklopovima***.

Dodatno možemo i znatno povećati razinu sigurnosti jer sada je moguće između segmenata mreže uvesti stroga pravila pristupa mrežnim resursima, poput: dozvoljavanja pristupa samo u određenom smjeru ili određenim smjerovima te dozvoljavanja pristupa samo određenim mrežnim servisima ili protokolima (primjerice HTTP) od ili prema određenom segmentu ili čak pojedinom računalu unutar pojedinog segmenta mreže.



***VLANovi** se uz **VXLANove** koriste i u virtualizaciji na razini Hipervizora, za mrežnu izolaciju virtualnih računala! Od linux kernela 3.8. moguće je na **Bridge** mrežnom sučelju (pogledajte prethodna poglavlja) definirati **VLAN** filtriranje.*

**Izvori informacija: (60),(386),(K-6),(K-10)**

## 20.6.2.2. Format 802.1Q polja

Prema standardu *802.1Q* unutar mrežnog (*Ethernet)* okvira, dodaje se 32 bitno polje koje opisuje i *VLAN*ove.
Tablica prikazuje izgled navedenog *802.1Q* polja:

| 802.1Q | | | |
|---|---|---|---|
| 16 bitova | 3 bita | 1 bit | 12 bitova |
| TPID | TCI | | |
| 0x8100 | PCP | DEI | VID |

Opis ***802.1Q*** polja:

- `Tag protocol identifier (TPID)` je 16 bitno polje koje ima vrijednost ***0x8100***, kako bi se mrežni okvir identificirao kao *IEEE* ***802.1Q***.
- `Tag control information (TCI)` polje koje sadrži:
  - `Priority code point (PCP)` - ovo je 3 bitno polje unutar kojega se prema standardu *IEEE 802.1p*, kao *class of service* definira razina prioriteta mrežnog okvira. Vrijednosti prioriteta, mogu biti od 0 do 7.
  - `Drop eligible indicator (DEI)` – ovo je 1-bitno polje (nekada je imalo oznaku **CFI**). Može se koristiti s *PCP* poljem kojim će se indicirati da se radi o paketu koji se može odbaciti, ako dođe do preopterećenja.
  - `VLAN identifier (VID)` - ovo je 12 bitno polje u kojem se definira kojem VLANu pripada mrežni okvir. Heksadecimalne vrijednosti 0x000 i 0xFFF su rezervirane. Sve druge vrijednosti se mogu koristiti, pa je moguće imati definirano maksimalno do **4096 *VLANova*** ***($2^{12}$)***.

**Izvori informacija: (60),(386),(K-6),(K-10).**

### 20.6.2.3. Rad s VLANovima pod Linuxom

Kako bi uopće koristili VLAN-ove odnosno ***802.1Q*** protokol, potrebno je zadovoljiti dva preduvjeta:

- Moramo imati mrežni preklopnik (*switch*) koji podržava ***802.1Q*** protokol odnosno *VLANove*.
- Moramo imati podršku za ***802.1Q*** protokol od strane Linuxa, ako želimo da je Linux svjestan *VLAN* mreža.

Što se tiče podrške od strane Linuxa, u nekim distribucijama linuxa odnosno u nekim inačicama distribucija linuxa ova podrška već dolazi unutar linux kernela dok je u nekim drugima potrebno učitati određeni kernel modul.
Stoga prvo provjerimo, za naš konkretni kernel, je li ova podrška unutar sâmog kernela ili je dostupna kao kernel modul:

```
grep -i CONFIG_VLAN_8021Q= /boot/config-`uname -r`

CONFIG_VLAN_8021Q=m
```

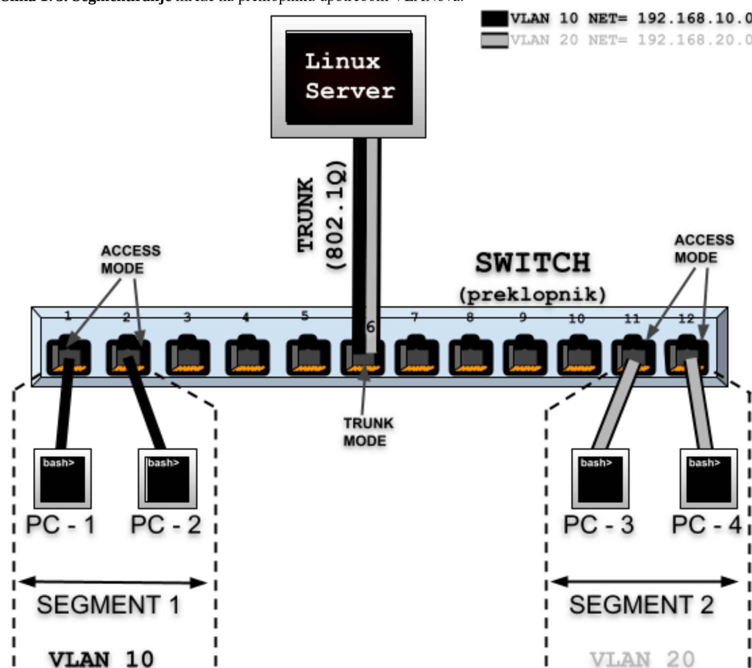
Oznaka koju mi imamo ovdje (`=m`) znači kako je podrška dostupna kao kernel modul koji moramo prvo učitati.
Stoga prvo učitajmo ovaj kernel modul s naredbom `modprobe` na sljedeći način:

```
modprobe 8021q
```

Nazivi fizičkih mrežnih sučelja odnosno mrežnih kartica u linuxu su obično: `eth0`, `eth1`, `eth2` i tako dalje, odnosno u novijim linuxima su to: `eno1`, `eno2`, `eno3` i slično. Nakon što imamo učitanu podršku za VLANove, sve što je potrebno je aktivirati VLAN mrežna sučelja koja se kreiraju tako da na fizičko mrežno sučelje primjerice na `eth0` nadodajemo broj VLAN-a, na način koji ćemo opisati u primjeru na slici 173. Zamislimo scenarij u kojem imamo jedan Linux poslužitelj: ***Linux Server*** vidljiv na slici 173, a koji ima fizičko mrežno sučelje `eth0` koje ćemo podijeliti na dva VLAN-a: ***VLAN10*** i ***VLAN20***.

Fizičko mrežno sučelje `eth0` će praktično raditi u ***TRUNK*** načinu rada (jer ćemo kreirati dva VLAN-a na njemu), a ono će propuštati i sve druge VLAN-ove koje mu preklopnik (*switch*) proslijedi, mada druge VLAN-ove trenutno nećemo koristiti.
Dakle naš Linux će se koristiti kao usmjerivač (OSI 3 uređaj) za povezivanje dva *VLAN*-a: ***VLAN10*** i ***VLAN20***.
Na tom fizičkom mrežnom sučelju Linux računala `eth0` ćemo kreirati dva logička mrežna sučelja; po jedno za svaki *VLAN*. Tako će se *VLAN* sučelja ponašati kao da se radi o zasebnim fizičkim mrežnim sučeljima (mrežnim karticama).

**Slika 173. Segmentiranje** mreže na preklopniku upotrebom VLANova.




Naš Linux poslužitelj je spojen na mrežno sučelje preklopnika broj **6**. To mrežno sučelje preklopnika broj **6** mora biti konfigurirano tako da radi u ***TRUNK*** načinu rada, kako bi uopće moglo označavati sve mreže pakete sa *802.1Q* zaglavljem (*tagirati ih*) te ih potom proslijediti prema Linux poslužitelju.
To u konačnici znači kako preklopnik svaki mrežni paket označava s oznakom pripadnosti konkretnom *VLANu* koja se nalazi unutar *802.1Q* zaglavlja svakog mrežnog paketa koji prolazi kroz ***TRUNK*** vrstu mrežnog sučelja.

Kada ti paketi dođu do Linuxa i njegove `eth0` mrežne kartice ona ih tako označene (s *802.1Q* oznakom) sâmo prosljeđuje do VLAN mrežnih sučelja, ako ona naravno postoje.
Tada `8021q` kernel modul skida *802.1Q* oznaku sa svakog mrežnog paketa i pretvara ga u običan (normalan) mrežni paket te ga šalje ovisno o njegovoj pripadnosti VLAN-u (odnosno VLAN oznaci koju je nosio) na njemu pripadajuće odredišno VLAN mrežno sučelje.

Dakle ovdje, ako je određeni paket nosio *802.1Q* oznaku VLAN-a broj ***10*** tada se on šalje na ***VLAN 10*** mrežno sučelje (`eth0.10` u konkretnom slučaju). U suprotnom smjeru, kada se mrežni paket šalje sa VLAN mrežnog sučelja, ako se recimo sada radi o ***VLAN 20*** mrežnom sučelju (`eth0.20`) mrežni paket dobiva *802.1Q* oznaku VLAN-a ***20*** te se kao takav označen (*tagiran*) šalje kroz fizičko mrežno sučelje `eth0` sve do preklopnika na preklopnikovo mrežno sučelje broj **6** (u ovom slučaju).
Dalje taj paket kroz preklopnik putuje s VLAN oznakom (*802.1Q* zaglavlje). Ako se radi o paketu koji nosi VLAN oznaku **20**, paket se tada šalje na mrežna sučelja preklopnika koja se nalaze u VLAN-u broj **20**, a to su u našem primjeru mrežna sučelja preklopnika **11** i **12**. Ako su ta mrežna sučelja na preklopniku (br. **11** i **12**) u *ACCESS* načinu rada, tada preklopnik prije nego proslijedi pakete na njih, prvo skida VLAN oznaku za VLAN **20** odnosno skida *802.1Q* polje i pretvara ih u normalan mrežni paket i tek tada ih prosljeđuje na mrežna sučelja **11** i **12** kao obične normalne pakete bez ikakvih VLAN (*802.1Q*) oznaka, iako oni pripadaju tom VLAN-u (broj ***20***). Tada ih računala spojena na ta mrežna sučelja preklopnika (**11** i **12**) vide kao obične normalne mrežne pakete i primaju ih, bez ikakvog znanja o VLAN-ovima i *802.1Q* protokolu.



Linux podržava upotrebu ***VLAN***ova na raznim mrežnim sučeljima:

- Od fizičkih mrežnih sučelja (**`eth0`**, **`eno0`**, **`enp`**, …), pa imamo primjerice ***VLAN*** sučelja*: **`eth0.10`**, **`eno0.20,`** ….
- Preko **bond** sučelja (**`bond0`**, **`team0`**, …), pa tako imamo primjerice ***VLAN*** sučelja*: **`bond0.10`**, **`bond0.20`**, …
- Upotrebom ***bridge*** sučelja (**`br0`**, **`vmbr0`**, …) te drugih logičkih sučelja (pr. **TAP, VETH, …**) ili ***Open vSwitch*** sustava.

## Konfiguracija Linux poslužitelja

Nazivi standardnih *VLAN* mrežnih sučelja pod Linuxom, za *VLAN 10* bi bilo: `eth0.10`, za *VLAN 20*: `eth0.20` i tako dalje.
***VLAN sučelja nasljeđuju MAC adresu od primarnog mrežnog sučelja (ili mrežne kartice), na kojoj se kreiraju!***

**VLAN 10 konfiguracija**

Konfigurirat ćemo novo mrežo sučelje `eth0.10` s IP adresom: 192.168.10.1 i maskom mreže 255.255.255.0 odnosno maskom /24. To najjednostavnije možemo ručno napraviti s novijom naredbom `ip` koja dolazi u paketu `iproute2` jer tada sve možemo napraviti s jednom naredbom (sve u jednom redu):

```
ip link add link eth0 name eth0.10 type vlan id 10
```

Tada će biti kreirano mrežno sučelje imena: `eth0.10` koje pripada *VLAN-u* broj **10**.
Međutim, mogli smo instalirati i `vconfig` alat jer se pomoću stare naredbe `ifconfig` to ne može napraviti direktno.
Sada je red na dodjeljivanje IP parametara, pa ćemo novom mrežnom sučelju za *VLAN* ***10*** ručno konfigurirati IP adresu:

```
ip address add 192.168.10.1/24 dev eth0.10
```

I aktivirat ćemo naše novo mrežno sučelje `eth0.10` s naredbom:

```
ip link set eth0.10 up
```

Permanentna (trajna) metoda konfiguracije bi bila kreiranje konfiguracijske datoteke (takozvane **ifcfg** datoteke) imena: `/etc/sysconfig/network-scripts/ifcfg-eth0.10` koja će sadržavati konfiguraciju *VLAN* ***10*** sučelja na fizičkoj mrežnoj kartici `eth0`. To znači da ova datoteka mora sadržavati sljedeće retke konfiguracije:

```
VLAN=yes
DEVICE=eth0.10
BOOTPROTO=static
ONBOOT=yes
TYPE=Ethernet
IPADDR=192.168.10.1
NETMASK=255.255.255.0
NETWORK=192.168.10.0
```

U slučaju kada potpuno prelazimo s normalne mreže na VLAN mrežu, odnosno kada ćemo IP adrese definirati za svako VLAN sučelje, na fizičkom mrežnom sučelju `eth0`, tada nam više ne treba IP adresa na fizičkom mrežnom sučelju, pa će naša konfiguracija čistog (fizičkog mrežnog sučelja) `eth0` biti definirana u datoteci:
`/etc/sysconfig/network-scripts/ifcfg-eth0` i izgledat će ovako (jer IP parametre tada definiramo na VLAN sučeljima):

```
DEVICE=eth0
TYPE=Ethernet
BOOTPROTO=none
ONBOOT=yes
```

U slučaju ako ručno želimo obrisati VLAN mrežno sučelje; primjerice `eth0.10` to možemo napraviti sa sljedećom naredbom:

```
ip link delete eth0.10
```

**VLAN 20 konfiguracija**

Konfigurirat ćemo novo mrežo sučelje `eth0.20` s IP adresom: 192.168.20.1 i maskom mreže 255.255.255.0 odnosno /24.

```
ip link add link eth0 name eth0.20 type vlan id 20
```

Sada ćemo dobiti mrežno sučelje imena `eth0.20` koje pripada VLAN-u broj **20**.
Potom ćemo novom mrežnom sučelju za VLAN *20* ručno konfigurirati IP adresu:

```
ip address add 192.168.20.1/24 dev eth0.20
```

I zatim aktivirajmo naše `eth0.20` (VLAN ***20***) mrežno sučelje sa sljedećom naredbom:

```
ip link set eth0.20 up
```

Permanentna (trajna) metoda konfiguracije bi bila kreiranje konfiguracijske datoteke imena:
`/etc/sysconfig/network-scripts/ifcfg-eth0.20` koja će sadržavati konfiguraciju VLAN **20** sučelja na fizičkoj mrežnoj kartici `eth0`:

```
VLAN=yes
DEVICE=eth0.20
BOOTPROTO=static
ONBOOT=yes
TYPE=Ethernet
IPADDR=192.168.20.1
NETMASK=255.255.255.0
NETWORK=192.168.20.0
```



Neke mrežne kartice podržavaju hardversko ubrzanje obrade ***VLAN*** mrežnih paketa (**802.1Q** dio).



Za detalje o načinu rada ***VLAN*** sučelja i njegovu vezu s nižim slojevima mrežnog stôga Linuxa, pogledajte poglavlje:
**26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije.**

Imamo li *VLAN* podršku u mrežnoj kartici, možemo za mrežno sučelje `eth0` to provjeriti sa sljedećom naredbom (skratili smo ispis):

```
ethtool -k eth0 | grep vlan

rx-vlan-offload: on
tx-vlan-offload: on
```

Vidimo da naša mrežna kartica podržava rasterećenje procesora (*Offload*) za rad s *VLAN* paketima i to u dolazu (`rx-vlan`) i odlazu (`tx-vlan`). Vidimo i da je oboje uključeno. Ako primjerice `rx-vlan` baš želimo isključiti, to možemo postići sa:

```
ethtool -K eth0 rxvlan off
```

Odnosno, ako želimo isključiti `tx-vlan` tada trebamo napraviti sljedeće:

```
ethtool -K eth0 txvlan off
```

**Ako već imamo ovu podršku u hardveru, dobro je imati ju uključeno da bi rasteretili CPU od obrade VLAN zaglavlja.**

## Konfiguracija preklopnika

Sada ćemo konfigurirati preklopnik te ćemo dati primjer za ***Cisco IOS*** operativni sustav, za ***Cisco Catalyst*** preklopnike. Konfiguracija ***TRUNK*** sučelja je na mrežnom sučelju broj **6**, odnosno po *Cisco* oznaci: *Gigabit Ethernet port* (***Gi***) *broj* **6**:

```
conf t
interface Gi 0/6
switchport trunk encapsulation dot1q
switchport mode trunk
```

Dat ćemo i primjer konfiguracije mrežnih sučelja u ***ACCESS*** načinu rada i to mrežna sučelja preklopnika broj **11** i **12**, koja pripadaju VLANu broj **20**, a radit će u ***ACCESS*** načinu rada:

```
conf t
interface Gi 0/11
switchport mode access
switchport access vlan 20

interface Gi 0/12
switchport mode access
switchport access vlan 20
```



*Preklopnici proizvođača poput **Juniper** i **Allied Telesys** koriste istu logiku konfiguracije **ACCESS/TRUNK** načina rada kao i **Cisco**. Međutim neki drugi proizvođači poput **HP** i **Brocade** koriste pojmove poput **Untagged** i **Tagged** kod konfiguracije navedenih načina rada sučelja.*

Pogledajmo usporednu tablicu nekoliko proizvođača mrežnih preklopnika, s terminologijama koje koriste:

| Proizvođač | TRUNK (802.1Q trunk) način rada | ACCESS način rada |
|---|---|---|
| Cisco (IOS) | `Trunk` | `Access` |
| Juniper (JunOS) | `Trunk` | `Access` |
| Allied Telesys | `Trunk` | `Access` |
| HP *(određeni modeli uređaja)* | `Tagged` | `Untagged` |
| Brocade | `Tagged` | `Untagged` |
| Ruckus (ICX serija) | `Tagged` | `Untagged` |

## 802.1ad (QinQ)

Osim klasičnih ***VLAN-ova*** definiranih ***802.1Q*** protokolom, koji praktično unose jednu oznaku (engl. *tag*) u *ethernet* mrežni okvir, telekomi i velike korporacije često koriste i drugu tehnologiju takozvanog dvostrukog označavanja (*tagiranja*) mrežnih okvira. Ova tehnologija je definirana u standardu ***802.1ad*** i radi upravo navedeno, uvodi još jednu ***VLAN*** oznaku pripadnosti, a koju takozvani *metro ethernet* (ili napredni) preklopnici prema potrebi mogu ukloniti nakon prijenosa, tako da krajnjem korisniku (pr. tvrtki kojoj se pruža usluga prenošenja klasičnih ***VLAN-ova***), ostaju vidljivi samo klasični ***802.1Q*** označeni ***VLAN-ovi***. Pogledajmo izgled *ethernet* mrežnih okvira to jest dvostruko označenih ***VLAN-ova*** (engl. *double tagged*), koji se nazivaju i ***QinQ*** (***802.1ad***), na drugom sloju mreže (***OSI 2***). Vidljiva je i drugačija ***Ether type*** oznaka (***0x88A8***) od ***802.1Q***.

| DMAC | SMAC | 802.1ad Tag | 802.1Q Tag | ETHER TYPE | DATA/PAYLOAD | FCS |
|---|---|---|---|---|---|---|
| *Odredišna* MAC | *Izvorišna* MAC | **802.1Q** oznaka **QinQ** | **802.1Q** oznaka | `0x88A8` | DATA/PAYLOAD (Sve s gornjih slojeva) | ***CRC*** provjerni zbroj *(Checksum)* |
| 6 *bajta* | 6 *bajta* | 4 *bajta* | 4 *bajta* | 2 *bajta* | minimalno 46 *bajta* | 4 *bajta* |
| ← U k u p n o (minimalno) 74 b a j t a → | | | | | | |

**Izvori informacija: (385),(386),(387),(388),(389),(390),(1066),(1194),(1310),(1342),(K-6),(K-10),** `man ethtool`**,**`man ip`

### 20.6.2.4. Automatska propagacija VLANova

**Slijedi napredna cjelina!** *Multiple Registration Protocol* (**MRP**), koji je zamijenio *Generic Attribute Registration Protocol* (**GARP**), čini generički programski okvir definiran u **IEEE 802.1ak** dodatku na **IEEE 802.1Q** standard. **MRP** omogućuje mrežnim mostovima (engl. *bridge*), preklopnicima ili drugim sličnim uređajima da registriraju i odjave vrijednosti atributa, kao što su VLAN identifikatori ili članstvo to jest pripadnost *multicast* grupama u lokalnim mrežama (**LAN**). **MRP** protokol radi na sloju podatkovne veze odnosno OSI sloju dva [***OSI 2***]. Vezano za VLAN mreže, koristi se ***MVRP*** protokol, koji je zamijenio stariji ***GVRP*** protokol. ***MVRP*** je mrežni protokol koji također radi na sloju podatkovne veze [***OSI 2***]. ***MVRP*** je definiran u **802.1ak** dodatku na ***802.1Q-2005*** standard, koji je proširivan, a trenutno je aktivan ***802.1Q-2018***.
***MVRP*** se koristi za automatsku konfiguraciju VLAN mreža među mrežnim preklopnicima (engl. *switches*). Naime unutar mreže na OSI sloju dva (OSI 2), *MVRP* pruža metodu za dinamičko dijeljenje informacija o VLAN mrežama te konfiguriranje potrebnih VLAN-ova. Na primjer, da biste dodali određeno sučelje na preklopniku (*port)* u određeni VLAN, potrebno je rekonfigurirati samo konkretno mrežno sučelje ili mrežni uređaj koji podržava VLAN mreže, spojen na to mrežno sučelje. Dok se potrebne VLAN *trunk* postavke načina rada sučelja dinamički kreiraju na drugim preklopnicima koji podržavaju MVRP. Dakle MVRP pomaže u održavanju VLAN konfiguracije dinamički na temelju trenutne konfiguracije mrežnih uređaja.
***MVRP*** nam preko ***802.1Q*** protokola omogućava:

- Dinamičku konfiguraciju i distribuciju informacija o pripadnosti VLAN mreža.
- Statičku konfiguraciju informacija o pripadnosti VLAN mrežama putem mehanizama upravljanja, koji omogućuju konfiguraciju statičkih unosa registracije VLAN-ova.
- Kombiniranu statičku i dinamičku konfiguraciju, u kojoj se neki VLAN-ovi mogu konfigurirati putem klasičnih mehanizama upravljanja, a drugi VLAN-ovi se mogu oslanjati na MVRP za konfiguraciju.

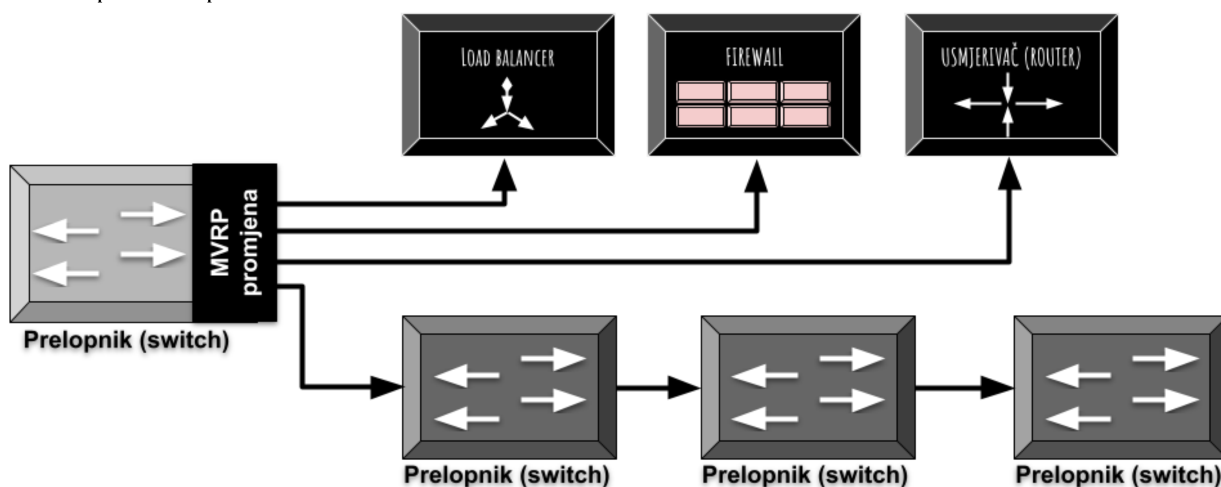
***MVRP*** definira takozvanu MRP aplikaciju koja pruža uslugu registracije VLAN-ova. MVRP koristi deklaraciju atributa MRP [*MRP Attribute Declaration*] (**MAD**) i širenje atributa MRP [*MRP Attribute Propagation*](**MAP**), koji daju opise stanja i zajedničke mehanizme širenja informacija definirane za upotrebu u aplikacijama temeljenim na MRP-u. *MVRP* pruža mehanizam za dinamičko održavanje sadržaja dinamičkih unosa registracije VLAN-ova za svaki VLAN i za širenje informacija koje sadrže, na druge mrežne preklopnike ili druge mrežne uređaje koji imaju tu potrebu odnosno koji podržavaju MVRP protokol.Ove informacije omogućuju uređajima svjesnim MVRP-a da uspostave i dinamički ažuriraju svoje znanje o skupu VLAN-ova koji trenutno imaju aktivne članove te preko kojih mrežnih sučelja (*portova*) se do tih članova može doći. Prema tome glavna svrha MVRP-a je omogućiti mrežnim preklopnicima da automatski otkriju informacije o VLAN-ovima koje bi inače trebale biti ručno konfigurirane na svakom pojedinom mrežnom preklopniku (ili uređaju).
Naime, bez korištenja MVRP-a, potrebna je ili ručna konfiguracija VLAN postavki ili korištenje sličnih protokola za automatsku konfiguraciju VLAN mreža koje su razvili određeni proizvođači mrežne opreme. Primjerice tvrtka Cisco je razvila tzv. VLAN Trunking Protocol (***VTP)*** te *Dynamic Trunking Protocol* (***DTP)***. Mana upotrebe ovakvih protokola koje su razvili te licencirali i zaštitili određeni proizvođači mrežne opreme je u tome što oni ne rade na mrežnoj opremi drugih proizvođača.
Podrška za ***MVRP*** postoji u novijim Linux kernelima, a možemo ju provjeriti sa sljedećim nizom naredbi:

```
grep CONFIG_VLAN_8021Q_MVRP /boot/config-$(uname -r)
CONFIG_VLAN_8021Q_MVRP=y
```

Dakle ako imamo vrijednost **=y** to znači da podršku za ***MVRP*** imamo već ugrađenu u sâm kernel. Zatim je potrebno i aktivirati ju za određeno mrežno sučelje, u *ifcfg* datoteci (`/etc/sysconfig/network-scripts/ifcfg-*`) željenog mrežnog sučelja, sa sljedećim VLAN opcijama:

```
VLAN=yes
MVRP=yes
```

**Slika 173.1. Upotreba MVRP protokola.**




Na slici vidimo upotrebu ***MVRP*** protokola među raznim mrežnim uređajima. Pri tome se na slici u primjeru, na krajnje lijevom preklopniku kreiraju, brišu ili mijenjaju VLAN-ovi, a nakon toga se oni automatizmom, pomoću MVRP protokola propagiraju na sve mrežne uređaje na slici. To primjerice znači da će se kreiranjem novog VLAN-a 40, na krajnje lijevom preklopniku, VLAN 40 u konačnici propagirati (kreirati) i na svim mrežnim uređajima na slici, pomoću ***MVRP*** protokola.

**Izvori informacija: (1225),(1226),(1227),(1228),(1229),(1230),(1231),** `man nm-settings`.

## 20.6.3. TUN i TAP - posebna mrežna sučelja

**Slijedi napredno poglavlje!**
**TUN** i **TAP** su posebna virtualna mrežna sučelja koja se koriste za posebne namjene. **TUN** (Engl *Network TUNnel*) simulira mrežno sučelje koje radi na OSI sloju tri (IP). Dakle *TUN* mrežno sučelje se koristi za namjene usmjeravanja (*routinga*) poput raznih *VPN* tunela (pr. *OpenVPN*) i slično. S druge strane **TAP** (Engl. *Network TAP*) simulira mrežno sučelje na nižem sloju odnosno OSI sloju dva, pa prema tome radi s *ethernet* mrežnim okvirima.
Tako se primjerice ***TAP*** sučelje najčešće koristi povezano s mrežnim mostom (Engl. *bridge*) koji je poveznica između *hipervizora* za virtualizaciju i fizičkog mrežnog sučelja (mrežne kartice) s jedne strane, te poveznice s virtualnom mrežnom karticom unutar virtualnog računala s druge strane.



Za ove obje vrste virtualnih mrežnih sučelja koristi se ista poveznica na upravljački program: `/dev/net/tun`. Odnosno, za njihov rad su potrebni slijedeći kernel moduli: `tun` (za **TUN** sučelje) i `tap` (za **TAP** sučelje).



Važno je razumjeti da osim fizičke mrežne kartice, svoju zasebnu i jedinstvenu MAC adresu dobiva i **TAP** logičko mrežno sučelje (ali i **Bridge**, **Bond**, **VETH** i neka druga ), jer ono radi na OSI sloju dva [*Ethernet*] pa je uključeno u preklapanje mrežnih paketa na osnovi MAC adresa. S druge strane **TUN** sučelje radi na OSI sloju tri te nema MAC adresu.



*Mrežna sučelja **TUN** i **TAP** su virtualna pa se moraju povezati s nekim drugim sučeljem, a obično je to **bridge** sučelje.*

**Kreiranje i brisanje *TUN* i *TAP* mrežnih sučelja**
Pogledajmo kako se kreira i aktivira ***TAP*** mrežno sučelje imena `tap0` upotrebom naredbe `ip`:

```
ip tuntap add tap0 mode tap
ip link set dev tap0 up
```

Za ispis svih mrežnih sučelja s opisom vrste sučelja, možemo koristiti: **`ip -d a`**. Obrisati ga možemo na sljedeći način:

```
ip tuntap del tap0 mode tap
```

Kreirati ***TUN*** mrežno sučelja imena `tun0` upotrebom naredbe `ip` možemo napraviti sa:

```
ip tuntap add tun0 mode tun
```

**Primjer upotrebe iz virtualizacije**
Ako smo pokrenuli **QEMU** (**KVM**) *hipervizor* za virtualizaciju te jedno virtualno računalo na njemu, koje ima konfiguriranu barem jedno mrežno sučelje (mrežnu karticu) to virtualno mrežno sučelje se mora povezati prema hipervizoru, na jedno ***TAP*** mrežno sučelje. Potražimo prema *file deskriptorima* poveznice prema svim `/dev/net/tun` uređajima, a tako ćemo pronaći i naš *hipervizor* (**KVM/QEMU**) koji u konkretnom slučaju ima ***PID*:** `18863`.

To ćemo napraviti pomoću naredbe `lsof`:

```
lsof | grep "/dev/net/tun"

COMMAND    PID    TID    USER    FD       TYPE        DEVICE   SIZE/OFF   NODE   NAME
kvm        18863         root    17u       CHR        10,200     0t70     136    /dev/net/tun
```

Naredba `lsof` nam je našla upravo proces `kvm` koji ima ***PID*** broj koji smo tražili, a koji koristi upravljački program za virtualno mrežno sučelje `/dev/net/tun` prema virtualnom računalu.



Podsjetite se naredbe `lsof` u poglavlju:
**25.7.6. Naredba list open files (*lsof*).**

Dakle sada za ***PID*** `18863` tražimo *file deskriptor* broj **17** te tražimo više podataka o njemu (o *file deskriptoru* br. 17 za ***PID*** `18863`). Stoga pogledajmo što opisnik file deskriptora **17** za ***PID*** `18863` govori:

```
cat /proc/18863/fdinfo/17

pos:     70
flags:   0104002
mnt_id:  20
iff:     tap310i0
```

Ono što smo tražili nalazi se u polju: `iff`. Dakle dobili smo traženo ime virtualnog mrežnog sučelja vrste **TAP** koje je u ovom slučaju: `tap310i0`. To znači da se virtualna mrežna kartica na virtualnom računalu veže za **TAP** sučelje: `tap310i0`.

*Napomena: `iff:` polje je novost koja nije dostupna u starijim (2.6.x) inačicama linux kernela.*

Pogledajmo i detalje o ovom ***TAP*** (`tap310i0`) mrežnom sučelju:

```
ifconfig tap310i0

tap310i0  Link encap:Ethernet  HWaddr 2a:bb:57:83:0b:20
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:625 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1266 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:52163 (50.9 KiB)  TX bytes:108607 (106.0 KiB)
```

Veza navedenih fizičkih i logičkih mrežnih sučelja je sljedeća:

*Virtualno računalo (virtualna mrežna kartica)* **`eth0`** → **`tap310i0`** (***TAP*** *sučelje*) → **`br0`** *(Bridge sučelje)* → **`eth0`** *(fizička mrežna kartica)*



Pogledajte i poglavlje u kojem dajemo primjere za rad s ***TAP*** i ***Bridge*** mrežnim sučeljima:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**
Za detaljan rad s ***TAP*** i ***Bridge*** sučeljima pogledajte i poglavlje: **27.1. Virtualizacija.**

Moguće je optimizirati **TX** međumemoriju i za ***TUN* i *TAP*** sučelja **(*txqueuelen*)**, ali i koristiti ***multiqueue***, kako je upisano u:
**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**
**25.2.2.1. Multiqueue i virtualizacija.**

**Izvori informacija:(702),(703),(704),** **`man lsof`**,**`man ip`**,**`ip link help`**,**`ip tuntap help`**.

## 20.6.4. VETH - posebno mrežno sučelje

**Slijedi napredno poglavlje!**
**VETH** je virtualno mrežno sučelje koje se koristi za posebne namjene. **VETH** (Engl. *Virtual Ethernet*) simulira mrežno sučelje koje radi na OSI sloju dva (*MAC*). ***VETH*** sučelja se kreiraju u pâru odnosno kao pâr povezanih virtualnih *ethernet* sučelja, a njih možemo promatrati kao virtualni *prespojni* kabel: sve ono što uđe na jednom kraju ovog „*kabela*" mora izaći na drugom kraju i obratno.

To čini *veth* pârove idealnim za povezivanje različitih komponenata virtualnog umrežavanja zajedno, kao što su:

- Linux mostovi odnosno Linux *bridge* sučelja.
- ***OVS*** mostovi (***Open vSwitch*** bridge/switch).
- ***LXC*** Linux kontejneri ili izolirani mrežni prostori Linuxa (Linux *network namespaces*),
- Kao i fizička ili druga mrežna sučelja.

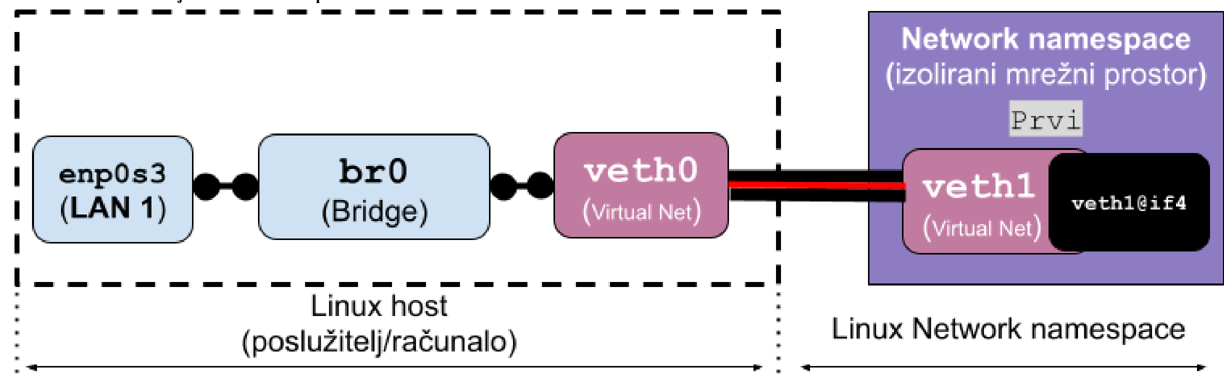


***VETH*** mrežno sučelje se u radu oslanja na kernel modul imena: `veth`.

Mi ćemo se u primjerima koji slijede nasloniti na priču o izoliranim mrežnim linux prostorima odnosno takozvanim *Linux namespaces*, kako smo već objasnili u poglavlju: **26.8.1.Network *Namespaces***.

Stoga zamislimo izolirani mrežni linux prostor imena: `PRVI`, kako je vidljivo na novoj shemi ovakvog spoja, na slici 173.A:

**Slika 173.A. VETH sučelje i izolirani mrežni prostor Linuxa**




Prisjetimo se kako je izolirani mrežni prostor Linuxa potpuno mrežno izoliran od našeg Linuxa u kojem smo ga kreirali.

Stoga prvo kreirajmo ovaj izolirani mrežni Linux prostor imena: `PRVI` pomoću naredbe `ip`, na sljedeći način:

```
ip netns add PRVI
```

Potom kreirajmo jedan ***VETH*** pâr mrežnih sučelja i to sučelja sljedećih imena: `veth0` i `veth1`:

```
ip link add veth0 type veth peer name veth1
```

**VETH** mrežna sučelja možemo provjeriti s naredbama: **`ip link list`** (vidjet ćemo kako su povezani).
Dok za ispis statusa svih mrežnih sučelja s opisom o vrsti sučelja (***bridge***, ***TUN***, ***TAP***, …) možemo koristiti naredbu:

```
ip -d a
```

Ili možemo vidjeti samo osnovne parametre s naredbom:

```
ifconfig -a
```

Sada ćemo ***VETH*** sučelje `veth1` povezati s našim izoliranim mrežnim Linux prostorom (*Linux namespace*) imena: `PRVI`

```
ip link set veth1 netns PRVI
```

Kako bi unutar našeg izoliranog mrežnog Linux prostora imena `PRVI` vidjeli novo sučelje `veth1` moramo isključivo unutar izoliranog mrežnog Linux prostora izlistati sučelja (jer ih nećemo vidjeti iz normalnog mrežnog prostora Linuxa) i to ovako:

```
ip netns exec PRVI ip link list

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen
1000  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3: veth1@if4: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default  qlen 1000 link/ether d2:dd:1b:69:6d:68 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Vidimo da se unutar našeg izoliranog mrežnog Linux prostora `PRVI` pojavilo naše novo mrežno sučelje: `veth1`.
Mogli smo i preimenovati ime mrežnog sučelja `veth1` u `eth0` i to samo unutar izoliranog mrežnog Linux prostora `PRVI`.

```
ip netns exec PRVI ip link set veth1 name eth0
```


Kako bi drugu stranu ***VETH*** tunela (u našem slučaju je to mrežno sučelje `veth0`) povezali s našom fizičkom mrežnom karticom (`enp0s3`) potrebno nam je ***bridge*** sučelje.

Kao ***bridge*** sučelje možemo koristiti ili ***Open vSwitch*** (***OVS***) ili standardni ***Linux bridge***.

Kako ne bi komplicirali s ***OVS (Open vSwitch** bridge/virtual switch)* koristit ćemo klasično Linux *bridge* sučelje.
Na fizičkom računalu (Linuxu) kreirajmo *bridge* mrežno sučelje imena: `br0` pomoću naredbe `ip` na sljedeći način:

```
ip link add name br0 type bridge
```

Potom aktivirajmo naše novo `br0` mrežno sučelje pomoću naredbe `ip` na sljedeći način:

```
ip link set dev br0 up
```

Zatim dodajmo `veth0` sučelje u naš *bridge* `br0` s čime smo ih povezali (dakle povezujemo sučelje `br0` i `veth0`):

```
ip link set dev veth0 master br0
```

I konačno dodajmo i našu fizičku mrežnu karticu `enp0s3` u naš *bridge* `br0`.
Sada imamo vezu: `enp0s3`→`br0`→`veth0`, a u konačnici zapravo imamo vezu: `enp0s3`→`br0`→`veth0` →`veth1`.

```
ip link set dev enp0s3 master br0
```


Važno je razumjeti da osim fizičke mrežne kartice, svoju zasebnu i jedinstvenu MAC adresu dobiva i **VETH** logičko mrežno sučelje (ali i **Bridge**, **Bond**, **TAP** i neka druga sučelja), jer ono radi na OSI sloju dva [*Ethernet*] pa je uključeno u preklapanje mrežnih paketa na osnovi MAC adresa.


Vezano za MAC adrese, pogledajte i poglavlje:
**19.3.1. Što su MAC adrese.**

Pogledajmo ***MAC*** adrese mrežnih sučelja koja smo do sada koristili (unutar primarnog mrežnog linux prostora):

```
ip -brief link show master br0
enp0s3           UP               08:00:27:6c:a4:71 <BROADCAST,MULTICAST,UP,LOWER_UP>
veth0@if3        UP               f6:e5:1d:a7:ca:e3 <BROADCAST,MULTICAST>
```

Vidimo da *bridge* sučelje (`br0`) povezuje našu fizičku mrežnu karticu (`enp0s3`) i ***VETH*** sučelje (`veth0`), a pri tome svatko od njih ima svoju jedinstvenu ***MAC*** adresu.

Pogledajmo i detalje, koje međuveze sada imamo u ***bridge*** sučelju imena: `br0`.
To ćemo vidjeti pomoću naredbe `ip` na sljedeći način:

```
ip link show master br0
2:veth0:<BROADCAST,MULTICAST,UP,LOWER_UP>mtu 1500  qdisc fq_codel master br0 state UP
4:enp0s3:<BROADCAST,MULTICAST,UP,LOWER_UP>mtu 1500 qdisc fq_codel master br0 state UP
```

Vidimo da su i `veth0` i `enp0s3` u istom `br0` Linux *bridge*-u, što potvrđuje njihovu povezanost.



*Prisjetimo se: pošto ovdje koristimo mrežni most (**bridge**): IP parametri se definiraju na razini sučelja mrežnog mosta; u našem slučaju je to mrežno sučelje **`br0`**, a ne na razini fizičkih mrežnih sučelja (pr. `enp0s3`, `eth0` ili `eth1`).*

Ne zaboravimo aktivirati i mrežno sučelje `veth0` i to ovako:

```
ip link set dev veth0 up
```

Važno je razumjeti da je ovdje **`veth0`** primarni dio pâra jer se nalazi u Linuxu (ne u izoliranom mrežnom prostoru linuxa).
Stoga možemo vidjeti njegovu vezu prema ***VETH*** pâru unutar izoliranog mrežnog linux prostora linuxa:

```
cat /sys/class/net/veth0/if{index,link}
4
3
```

Vidimo broj `4` koji označava **@if4** što je veza na `veth1` što je vidljivo kao `veth1@if4`, dok broj `3` označava **@if3** što je veza na `veth0` što je vidljivo kao `veth0@if3`.

Pogledajmo kako smo to saznali, pozivanjem dvije naredbe redom:

```
ip link list veth0
5: veth0@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
ip netns exec PRVI ip link list veth1
4: veth1@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
```

Potom možemo unutar našeg izoliranog mrežnog linux prostora imena `PRVI` dodijeliti IP adresu (`192.168.1.212`) mrežnom sučelju `veth1`.
To ćemo postići na sljedeći način:

```
ip netns exec PRVI ifconfig veth1 192.168.1.212/24 up
```

Sada možemo uči u naš izolirani mrežni linux prostora imena `PRVI` i pokrenuti ***bash*** ljusku te nastaviti s radom/testiranjem:

```
ip netns exec PRVI bash
```

Za ispis detalja oko mrežnih imeničnih prostora, možemo koristiti naredbu `lsns` na sljedeći način:

```
lsns --output-all --type=net
```



Pogledajte i poglavlja:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**
**26.8.1. Network *Namespaces*.**
**27.2. Linux kontejneri.**
**26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije** ← *za vezu prema mrežnom stôgu Linuxa.*

Pogledajte i poglavlja u kojima govorimo o posebnim mrežnim sučeljima:
**26.8.2. Druga mrežna sučelja (*MACVLAN,IPVLAN,VXLAN,MACVTAP/IPVTAP*).**



Sumirajmo: **VETH** sučelja su virtualni *Ethernet* uređaji.
**VETH** sučelja se ponašaju kao tuneli između različitih mrežnih imenskih prostora Linuxa (*network namespaces*) to jest za stvaranje veze među njima. Oni se vežu s fizičkom mrežnom karticom pomoću **bridge** sučelja s jedne strane, odnosno mrežnog sučelja u drugom mrežnom imenskom prostoru (*network namespaces*) s druge strane.

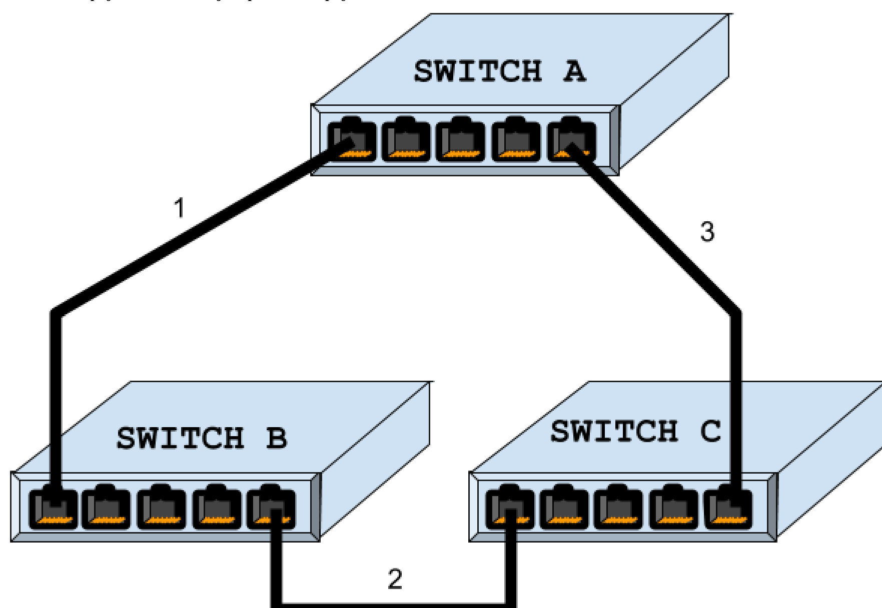
**VETH** sučelje se može koristiti i kao samostalno mrežno sučelje, a dodatno može pripadati i određenom VLAN-u!

**Izvori informacija: (742),(743),(744),(1342), `ip link help`,`ip netns help`,`man lsns`.**

## 20.6.5. Redundancija i *Spanning Tree protokol* (STP)

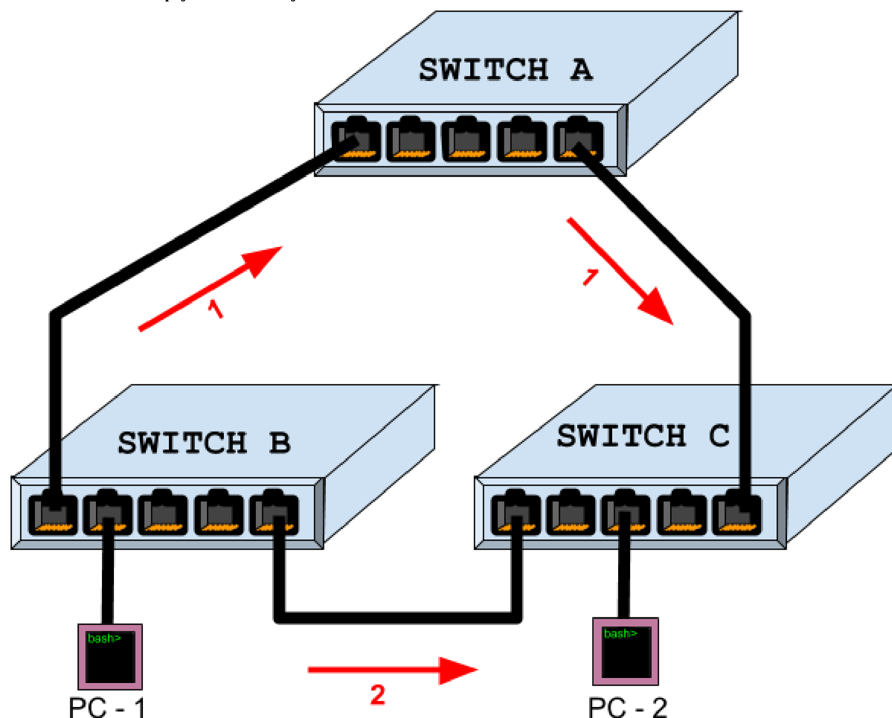
Kada u mreži želimo dobiti zalihost (redundanciju) odnosno povezati preklopnike ili druge mrežne uređaje u logički prstenasti spoj, u svrhu osiguravanja nesmetanog rada mreže ispadom bilo kojeg preklopnika (u ovom primjeru), moramo koristiti neki mrežni protokol koji bi spriječio pojavu petlje u mreži, iako smo ju stvorili ovakvim povezivanjem uređaja. O čemu se ovdje radi?. Naime redundanciju na razini mrežnih uređaja možemo dobiti povezivanjem mrežnih uređaja u prstenasti spoj prema principu: prvi uređaj se spaja na drugi, drugi na treći, a treći na prvi (1-2-3-1). S time smo u slučaju kada se neki od uređaja pokvari ili se neka od međuveza prekine, dobili alternativni put do odredišta (pr. 1-3, 1-2, 3-2). U tom trenutku će komunikacija krenuti drugim (alternativnim) smjerom odnosno putem. Pogledajmo sliku 173.1 prstenastog spoja tri preklopnika (*switcha*).

**Slika 173.1 Spoj mrežnih uređaja u prstenasti spoj.**




U normalnoj komunikaciji između dva računala (slika 173.2) **PC-1** i **PC-2** (*Unicast* komunikacija) dogodit će se sljedeće. Računalo (**PC-1**) šalje paket prema računalu **PC-2**. Paket dolazi na prvi preklopnik (**Switch B)** koji će poslati prvi mrežni paket poslan s računala **PC-1** prema preklopniku **Switch A** jer mu je tako vidljivo u *MAC* tablici odnosno odredište za ovaj paket je prema tom preklopniku. Drugi preklopnik **Switch A** zaprima paket i prema svojoj *MAC* tablici otkriva da paket mora slati dalje na preklopnik **Switch C**, a koji onda šalje paket direktno na odredišno računalo (odlučio je prema svojoj *MAC* tablici).

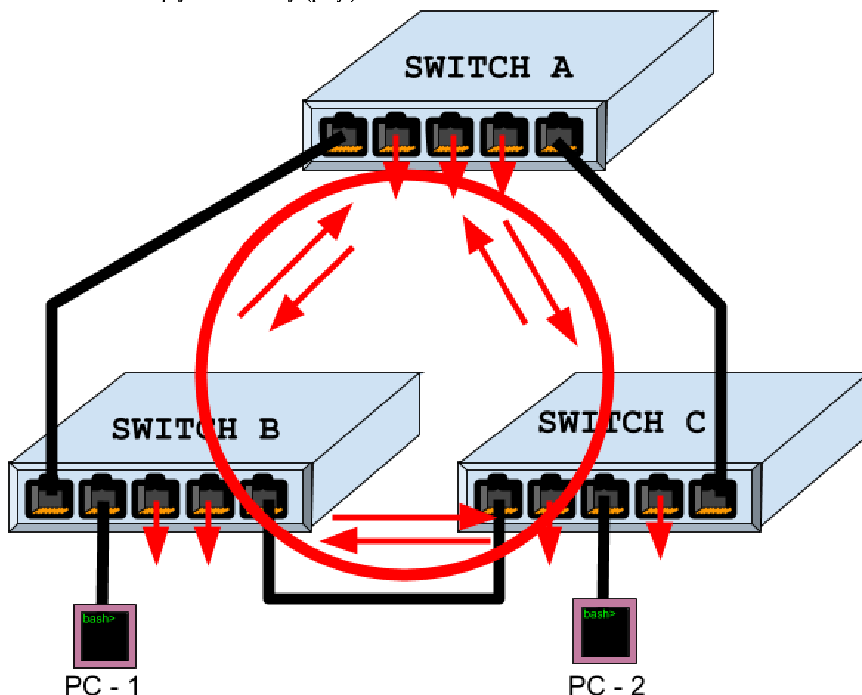
**Slika 173.2 Prstenasti spoj mrežnih uređaja i računala.**




Mrežni paket na povratku može ići istim putem, ali se u nekom trenutku može dogoditi, nakon pražnjenja *MAC* tablice ili drugih promjena, da preklopnik **Switch C** sada prosljeđuje paket preko preklopnika **Switch B**, do odredišnog računala.
U tom trenutku, ako računalo **PC-1** vraća paket na drugo računalo (**PC-2**), može ga vrati istim putem, ali ako on ima u svojoj *MAC* tablici i unos koji mu govori da do odredišta može doći i preko drugog preklopnika (**Switch A**), on ga može poslati i prema

tamo. Tada se može dogoditi da na odredište dođu duplicirani paketi. Još veći problem predstavlja komunikacija u kojoj se šalju *Broadcast* ili *Multicast* paketi koji se šalju na sva mrežna sučelja na svim preklopnicima, jer ih oni tako moraju i prosljeđivati. Ako računalo **PC-1** pošalje *Broadcast* poruku, primjerice ***ARP Request***, ***DHCP Discovery*** ili slično, pošto je odredišna *MAC* adresa *broadcast* adresa, svi preklopnici koji prime ovu poruku MORAJU ju proslijediti na SVA svoja mrežna sučelja (*portove*). Kako poruka bude putovala od preklopnika do preklopnika, svaki od njih će ju proslijediti (prekopirati) na sve svoje portove odnosno sva svoja mrežna sučelja. Kada ova poruka dođe s druge strane sve do preklopnika **Switch C** i on će ju ponovno proslijediti na sva svoja mrežna sučelja pa će poruka ponovno doći do izvorišnog preklopnika **Switch B** koji će ju ponovno proslijediti na sva mrežna sučelja. Tada će doći do toga da će se ova poruka u beskonačnosti kopirati i *vrtjeti* po ovoj zatvorenoj mreži i ona će vrlo brzo preplaviti sve preklopnike i toliko ih zagušiti da će se zablokirati i prestati s radom. Naime unutar mrežnih okvira na OSI sloju dva (OSI 2) ne postoji *TTL* polje (vrijeme života paketa) koje bi pomoglo da mrežni okvir nakon određenog broja prolazaka kroz preklopnike bude odbačen, pa se stoga *Broadcast* poruke u mrežama koje su u prstenastom spoju odnosno imaju zatvorenu petlju; beskonačno kopiraju i prosljeđuju s jednog uređaja na drugi, kao što je vidljivo na slici 173.3.

**Slika 173.3 Prstenasti spoj mrežnih uređaja (petlja).**




Stoga je za ovakve mreže u kojima su mrežni uređaji međusobno spojeni u prsten odnosno u spoju među njima postoji petlja, potrebna dodatna pomoć, koja je implementirana u protokolu znanom kao ***Spanning Tree protokol*** (*STP*).
Upotrebom ***STP*** protokola se razmjenjuju posebni ***BPDU*** mrežni okviri na OSI sloju dva koji koriste **802.2 LLC** (*Logical Link Control*) pod sloj. Prema tome, ako želimo redundanciju (zalihost) u mreži koja ima petlju, moramo imati konfigurabilni preklopnik koji podržava ovaj (ili neki noviji) protokol za ovakvu vrstu redundancije.
Osim toga, ovaj protokol nas štiti i od slučajeva u kojima netko slučajno ili zlonamjerno može napraviti petlju u mreži te s time ugroziti cijelu mrežu.

Važno je razumjeti da ovdje stalno govorimo o redundanciji na OSI sloju dva i petlji koja je napravljena između klasičnih preklopnika koji rade samo na OSI sloju dva. Za postizanje redundancije međuveza te ubrzanje istih, također na OSI sloju dva (OSI 2) postoji druga vrsta protokola poput *LACP* protokola, o kojem ćemo govoriti u sljedećim poglavljima.

***STP*** je bio prvi od protokola koji rješava ovaj problem, a poslije njega su napravljene mnoge njegove varijante koje rade znatno brže od njegove inicijalne inačice, poput:
• ***Rapid Spanning Tree Protocol*** *(RSTP) te* ***Per-VLAN Spanning Tree*** i ***Per-VLAN Spanning Tree Plus***.
• ***Rapid Per-VLAN Spanning Tree*** te ***Multiple Spanning Tree Protocol*** *(MSTP)*.

Izvorni ***STP*** protokol je definiran u standardu 802.1D. Mi ćemo dalje zbog jednostavnosti govoriti o izvornom ***STP*** protokolu. Naime osim u preklopnicima, ovaj protokol se može koristiti i u Linuxu za posebne namjene. Naime važno je razumjeti da se vrlo često Linux koristi kao platforma za nove generacije preklopnika za podatkovne centre u takozvanim ***SDN*** (*Software Defined Network*) uređajima odnosno implementacijama. Standardno u Linuxu se sa **STP** protokolom radi upotrebom naredbe `brctl` koja dolazi u softverskom paketu `bridge-utils`. Mi se nećemo više odnosno detaljnije baviti sa ***STP*** protokolom jer je njegova primjena u Linuxu ograničena za posebne primjene, dok se na klasičnim mrežnim uređajima i dalje koristi.



Pogledajte i poglavlje:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**

Za visoku dostupnost sustava, te što ona znači, pogledajte poglavlje:
**26.8.4. Visoko dostupni sustavi (*High Availability*).**

Ako do mrežne kartice dolaze STP paketi s preklopnika, a kernel modul za STP (`bridge`) nije učitan, takvi paketi će biti odbačeni. To također znači da će takvi odbačeni paketi biti vidljivi u statistikama naredbi poput: `netstat`, `nstat` i `ethtool`!

**Izvor informacija: (793), (K-6),(K-10),** **`man brctl`**,**`man bridge`**.

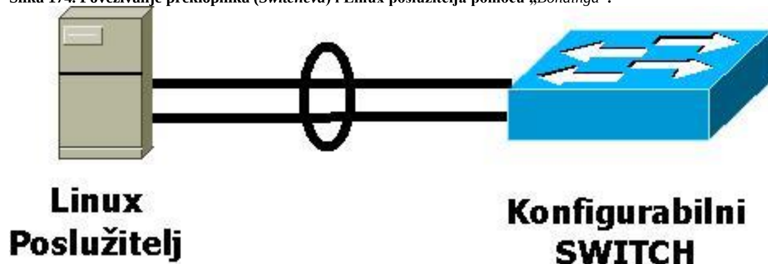
## 20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*)

**Slijedi napredno poglavlje (20.6.6.x)!**
Pojmovi: *Bonding*, *Agregacija*, *Team* ili *Ether Channel* označavaju razne metode logičkog povezivanja više fizičkih mrežnih sučelja (kartica) u jedno logičko mrežno sučelje. To logičko mrežno sučelje ima veću efektivnu ukupnu propusnost od pojedine fizičke mrežne kartice te omogućava redundanciju odnosno ispad jedne (ili više) fizičkih mrežnih kartica bez prekida u radu na razini logičkog mrežnog sučelja, a i sâme mrežne komunikacije. Kako radi Linux ***Bonding*** tehnologija:

- Konfigurira se svaka fizička mrežna kartica koja će biti povezana u *bond*, a potom mrežne kartice postaju *tzv.* robovske odnosno pomoćne (engl. *slave)* kartice.
- Od ovog trenutka apsolutno svi *IP parametri* fizičkih mrežnih kartica (sučelja) postaju nepotrebni jer se sva konfiguracija radi u sljedećim koracima na logičkim mrežnim sučeljima odnosno takozvanim *bond* sučeljima.
- Sada se konfiguriraju inicijalan parametri *bond* logičkog mrežnog sučelja odnosno takozvanog *bond* sučelja.
- Potom se konfiguriraju IP parametri *bonding* mrežnog sučelja: IP adresa i maska mreže, IP adresa mreže, *broadcast* IP adresa, podrazumijevani usmjerivač (*Default Gateway*) i drugo.
- Pokreće se Linux *Bonding* upravljački program odnosno kernel modul imena: `bonding`.
- I konačno se kreira posebno logičko mrežno sučelje to jest *bond* mrežno sučelje (engl. *Channel Bonding Interface*) imena `bond`**`x`**, primjerice: `bond0`, `bond1`, … .

Važno je znati kako postoji nekoliko različitih načina rada *Bonding-a* odnosno *agregacije* od kojih neki zahtijevaju i rekonfiguraciju sa strane preklopnika na koji su spojeni, te naravno i podršku za protokole koji su potrebni za ovakav rad, također na strani preklopnika. Logička shema ovakvog povezivanja je vidljiva na slici **174.**

**Slika 174. Povezivanje preklopnika (*Switcheva*) i Linux poslužitelja pomoću „*Bondinga*“.**






Za visoku dostupnost sustava, te što ona znači, pogledajte poglavlje:
**26.8.4. Visoko dostupni sustavi (*High Availability*).**
Za vezu ***bondinga*** prema mrežnom stôgu Linuxa i detaljima o načinu rada na najnižoj razini, pogledajte poglavlje:
**26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije.**



*U međuvremenu, pojavila se i druga tehnologija pod Linuxom koja se zove **Teaming**, a koja rješava iste probleme kao i **Bonding** i u praksi daje iste ili slične brzine, ali nudi i dodatne mogućnosti, koje su nekima potrebne.*
*Za više informacija o Linux **Teamingu** pogledajte izvore informacija:* (**748**),(**749**),(**750**).

***Od Red Hat 9.x. napušten je Teaming te su sve aktivnosti i razvoj fokusirani na Bonding.***

### 20.6.6.1. Vrste *bondinga* i načini njegovog rada

Na većini Linuxa od kernela 2.6.32 dostupni su sljedeći načini (Engl. *modes*) rada logičkog uvezivanja mrežnih kartica (*Bondinga),* a koji se označavaju prema definiranim brojčanim oznakama u Linuxu:

- **`mode=0`** **(balance-rr)** *Round-robin load balancing* odnosno algoritam kružnog dodjeljivanja/raspodjele prometa:
  - Šalje pakete sekvencijalno od prve fizičke “engl. *slave*” mrežne kartice unutar *bond-a,* do zadnje.
  - Prednosti: *Load Balancing* (raspodjela prometa) i tolerancija na ispad.
  - Nije potrebna konfiguracija preklopnika (*switcha*).

- **`mode=1`** **(active-backup)** *aktivno-pričuvni* odnosno način rada u kojem je sâmo jedan aktivan, a drugi u pripravnosti:
  - Samo prva mrežna kartica (*slave*) je aktivna. Druga (*slave*) postaje aktivna sâmo isključivo, ako je trenutno aktivna kartica neispravna. MAC adresa *Bond-a* je vidljiva sâmo na jednom mrežnom sučelju kako se ne bi stvarali problemi na strani preklopnika.
  - Prednosti: tolerancija na ispade.
  - Nije potrebna konfiguracija preklopnika (*switcha*).

- `mode=2` **(balance-xor)** *XOR* (ekskluzivno ILI – logička operacija). Način rada prema *XOR* algoritmu raspodjele:
    - Šalje mrežne pakete bazirano na sljedećem algoritmu:
      *Raspored* = (*Source MAC adresa* ***XOR*** *Destination MAC adresa*) **mod** (*ukupan broj* "slave" *kartica*)
    - Ova metoda odabire uvijek istu mrežnu karticu (slave) za određenu odredišnu *MAC* adresu.
    - Prednosti: *Load Balancing* (raspodjela prometa) i tolerancija na ispad.
    - Nije potrebna konfiguracija preklopnika (*switcha*).

- `mode=3` **(broadcast)** *Broadcast* način rada:
    - Šalje pakete na sve mrežne kartice (*slave*).
    - Prednosti: tolerancija na ispade.
    - Nije potrebna konfiguracija preklopnika (*switcha*).

- `mode=4` **(802.3ad)** Koristi se ***IEEE 802.3ad*** protokol za dinamičku *agregaciju* (*bonding*) s preklopnikom (*switchem*):
    - Kreira se *agregacijska* (*bonding*) grupa koja dijeli istu brzinu i *duplex* na svim mrežnim karticama. Koriste se sve mrežne kartice istovremeno, za primanje i slanje mrežnih paketa, prema ***802.3ad*** protokolu.
    - Prednosti: *Load Balancing* (raspodjela prometa), iskorištavanje svih fizičkih mrežnih kartica i za slanje i primanje paketa te tolerancija na ispad, što nudi najveću propusnost i pouzdanost.
    - Potrebna je rekonfiguracija preklopnika, a preklopnik mora podržavati protokol ***802.3ad*** i naravno konfiguraciju tog protokola. Više o ovom načinu rada u sljedećem poglavlju.

- `mode=5` **(balance-tlb)** Adaptivni "*Transmit load balancing*" odnosno balansiranje slanja paketa:
    - Šalju se paketi koji su distribuirani prema trenutnom opterećenju na svakoj mrežnoj kartici (*slave*). Dolazni promet se prima preko jedne kartice, a ako je ona neispravna druga kartica preuzima ulogu primatelja.
    - Prednosti: *Load Balancing* (raspodjela prometa) i tolerancija na ispad.
    - Nije potrebna konfiguracija preklopnika (*switcha*).

- `mode=6` **(balance-alb)** Adaptivni "*Load balancing*" odnosno prilagodljivi način balansiranja raspodjele prometa:
    - Uključuje prethodni način rada (***mode*=5**) i dodaje "*receive load balancing*" tj. balansiranje primatelja tj. mrežnih kartica koje primaju promet. Dakle dolazni *load balancing* (*receive load balancing*) se postiže s time da *bonding* upravljački program barata s *ARP* porukama tj. presreće ih u odlasku prema mreži i mijenja njihovu izvorišnu (*source)* MAC adresu. Na taj način svaki paket ima kao *source* MAC adresu, adresu druge fizičke mrežne kartice unutar *bond-a* pa će na taj način i dolazni paketi ići svaki na drugu fizičku mrežnu karticu. Na ovaj način se dobiva i *load balancing* odlaznog prometa, uz nešto veću potrošnju resursa (CPU) na te operacije.
    - Nije potrebna konfiguracija preklopnika (*switcha*).

Naveli smo sâmo osnovne principe i načine rada Linux ***bonding**-a,* koji se ponekad naziva i ***Team***.

---

### 20.6.6.2. Konfiguracija *bondinga* u Linuxu

Povezivanje više fizičkih mrežnih kartica u jedno logičko mrežno sučelje, radi se pomoću tehnologije koja se pod Linuxom zove ***bonding*** ili ***teaming***. Prvo provjerimo je li naš trenutni kernel uopće kompiliran s podrškom za *bonding*:

```
grep BONDING /boot/config-`uname -r`
CONFIG_BONDING=m
```

Vidimo da je *bonding* podržan kao kernel modul jer imamo oznaku `=m`.
Potom moramo učitati ovaj kernel modul, kako bi uopće dobili funkcionalnost *bondinga*, upotrebom naredbe `modprobe`:

```
modprobe bonding
```

Sada je sve spremno. Za sam kernel modul `bonding`, moguće je definirati i mnoge parametre rada koji su vidljivi s naredbom `modinfo`, a mi ćemo filtrirati sâmo navedene parametre odnosno ispisat ćemo koji od njih su dostupni za naš kernel modul.

```
modinfo bonding |  grep -i parm
```

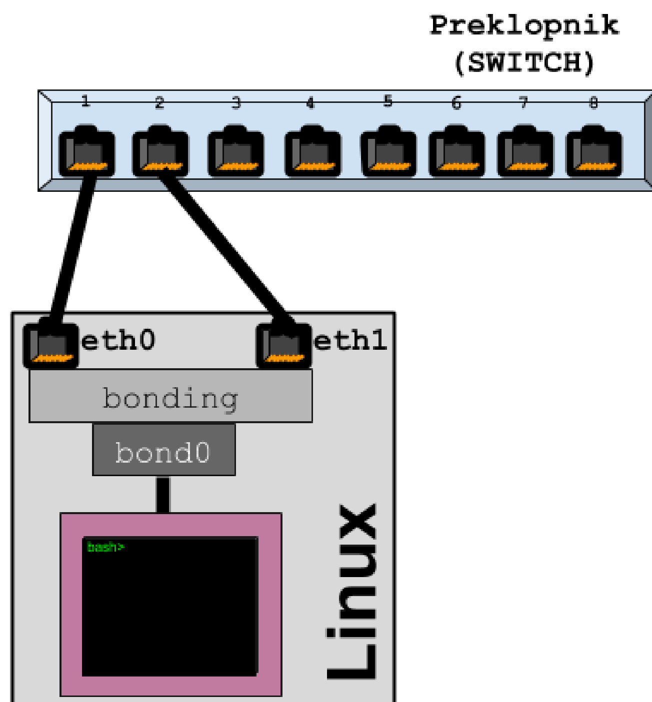
Koristit ćemo naše dvije mrežne kartice: `eth0` i `eth1` kako bi ih dodali u logičko `bond0` mrežno sučelje.
Pri tome smo se odlučili za *bond mode* broj **2** (*aktivno-pričuvni*), što baš i nije najbolje za brzinu rada i iskorištenost mrežnih kartica, ali je najjednostavnije za primjer.



**Bond** sučelja mogu biti povezana i s **Bridge** sučeljem. Nadalje, **Bond** sučelja prema potrebi mogu pripadati i određenom VLAN-u, pa možemo primjerice imati **bond** sučelja: **`bond0.10`**, **`bond0.20`** i slična.

Pogledajmo shemu spajanja našeg Linuxa s preklopnikom na slici 175.

**Slika 175. Povezivanje Linux poslužitelja s preklopnikom.**





Osim takozvane ***ifcfg*** konfiguracije koju ćemo dolje objasniti, moguća je i konfiguracija *bondinga* s ***NetworkManager servisom*** odnosno pomoću naredbe **nmcli,** kako je objašnjeno u poglavlju:

**25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**

Da bismo krenuli dalje, prvo moramo prirediti konfiguraciju naših fizičkih mrežnih kartica `eth0` i `eth1`.
Krenimo od konfiguracije kartice `eth0`, pa otvorimo njenu konfiguracijsku datoteku (takozvanu ***ifcfg***):
`/etc/sysconfig/network-scripts/ifcfg-eth0`, koja će sada sadržavati sâmo sljedeće retke teksta.

Pri tome konfiguracija u datoteci `/etc/sysconfig/network-scripts/ifcfg-eth0` je sljedeća:

```
DEVICE=eth0
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
```

Važno je razumjeti da u konfiguraciji fizičke mrežne kartice više **ne smije** biti IP parametara jer se oni sada konfiguriraju na razini logičkog *bond* mrežnog sučelja.

Objasnit ćemo što smo postavili u konfiguraciji mrežne kartice `eth0` i to sâmo važne dijelove:

- `DEVICE=eth0` - definira se upotreba fizičke mrežne kartice `eth0`.
- `MASTER=bond0` - ovdje se definira *MASTER* (upravitelj) *bond* mrežno sučelje imena `bond0` kao upravitelj gore navedenoj fizičkoj mrežnoj kartici `eth0`.
- `SLAVE=yes` - ovdje se definira kako se radi o fizičkoj mrežnoj kartici `eth0` koju ovdje dodajemo kao pomoćnu/robovsku (Engl. *Slave*). Što znači da se ona dodaje u gore navedenu `bond0` upravljačku grupu.

Sada nastavljamo s konfiguracijom druge kartice odnosno `eth1` stoga otvorimo njenu konfiguracijsku datoteku imena:
`/etc/sysconfig/network-scripts/ifcfg-eth1`, koja će sadržavati sâmo sljedeće dijelove:

```
DEVICE=eth1
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
```

Pogledajmo što smo postavili ovdje (samo važne dijelove):

- `DEVICE=eth1` – radi se o fizičkoj mrežnoj kartici `eth1` koju ovdje definiramo.
- `MASTER=bond0` – ovdje se definira *MASTER* (upravitelj) *bond* mrežno sučelje imena `bond0` kao upravitelj gore navedenoj fizičkoj mrežnoj kartici `eth1`.
- `SLAVE=yes` – ovdje se definira kako se radi o fizičkoj mrežnoj kartici `eth1` koju ovdje dodajemo kao pomoćnu/robovsku (Engl. *Slave*) što znači da se ona dodaje u gore navedenu `bond0` upravljačku grupu.



Kako je vidljivo iz obje konfiguracijske datoteke za fizičke mrežne kartice `eth0` i `eth1` ovdje **NEMA** konfiguracije **IP** parametara jer se oni konfiguriraju sâmo i isključivo na razini logičke ***bond*** kartice (`bond0`, `bond1`, …).

I na kraju moramo kreirati novo mrežno sučelje `bond0` i konfiguracijsku datoteku za njega, pa će to biti datoteka imena: `/etc/sysconfig/network-scripts/ifcfg-bond0`. Pogledajmo što ona mora sadržavati:

```
DEVICE=bond0
NAME=bond0
TYPE=Bond
BONDING_MASTER=yes
BONDING_OPTS="miimon=100 mode=active-backup"
```

```
IPADDR=192.168.100.20
NETWORK=192.168.100.0
NETMASK=255.255.255.0
USERCTL=no
BOOTPROTO=none
ONBOOT=yes
```

U prvoj cjelini su opcije i parametri vezani za *bonding*:

- `DEVICE=bond0` – radi se o *bond* mrežnom sučelju imena `bond0` koje ovdje definiramo.
- `NAME=bond0` – ovdje definiramo ime logičkog mrežnog sučelja: `bond0`.
- `TYPE=Bond` – pošto se radi o *bond* vrsti mrežnog sučelja to moramo definirati ovdje.
- `BONDING_MASTER=yes` - ovdje definiramo kako je ovo mrežno sučelje (`bond0`) „*MASTER*" odnosno odgovorno je za „*bond*" funkcionalnost.
- `BONDING_OPTS="miimon=100 mode=active-backup"` – ovdje definiramo vrstu ***bond*** mrežnog sučelja i druge parametre rada. Ova opcija s identičnim parametrima se može definirati i na sljedeći način. Dakle upotrebom ***bond*** brojčanog načina [***moda***] rada, poput: `BONDING_OPTS="mode=1 miimon=100"` što znači kako ovo `bond0` mrežno sučelje stavljamo u „*aktivno/pričuvni*" odnosno (***bond mode=1***) način rada. I dodatno kako će se stanje fizičkih mrežnih sučelja (mrežnih kartica) `eth0` i `eth1` pratiti preko ***MII*** sučelja (*Media Independent Interface*) i to svakih 100 milisekundi.

A u drugoj cjelini definiramo IP parametre koje će dobiti mrežno sučelje `bond0`:

- `IPADDR=192.168.100.20` – postavili smo mu (`bond0`) IP adresu: 192.168.100.20.
- `NETWORK=192.168.100.0` – postavili smo mu (`bond0`) IP adresu mreže na: 192.168.100.0.
- `NETMASK=255.255.255.0` – postavili smo mu (`bond0`) masku mreže (*netmask*) na: 255.255.255.0.

I na kraju kako bismo osigurali da se *bonding* kernel modul učita tijekom svakog pokretanja sustava dodajmo ga u konfiguraciju. Stoga kreirajmo datoteku imena: `/etc/modprobe.d/bond.conf` i u nju dodajmo sljedeći redak, što se često odradi automatski za nas, od strane sustava. Mi ćemo to ipak napraviti ručno (što za ***RedHat/CentOS 7+*** nije potrebno):

```
alias bond0 bonding
```



Preporuka je **<u>NE</u>** konfigurirati opcije navedene pod (`BONDING_OPTS`) u datoteci: `/etc/modprobe.d/bond.conf` jer se kod restartanja *bond* mrežnog sučelja ove opcije ponovno ne čitaju. Odnosno ne postavljaju se čitajući iz ove datoteke, već iz datoteke: `/etc/sysconfig/network-scripts/ifcfg-bond0`.

Dakle nakon prvog restarta računala naše novo mrežno sučelje `bond0` će postati aktivno s dodijeljenom IP adresom. Pogledajmo što smo sada dobili (obratite pažnju i na ***MAC*** adresu ovog i `eth0` mrežnog sučelja vidljivu pod `HWaddr`):

```
ifconfig bond0

bond0     Link encap:Ethernet  HWaddr 6c:3b:e5:bd:f5:78
          inet addr:192.168.100.20  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe83::22c:22ff:fe9a:fs2b/64 Scope:Link
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
          RX packets:75 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:50875 (50.8 KiB)  TX bytes:2446 (2.4 KiB)
```

Aktivna ***MASTER*** *bond* mrežna sučelja možemo vidjeti sa sljedećom naredbom:

```
cat /sys/class/net/bonding_masters

bond0
```

Međutim za više detalja sâmo o našem bond0 sučelju pokrenite sljedeću naredbu (važnije statuse smo označili zatamnjeno):

```
cat /proc/net/bonding/bond0

Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)
Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: eth0
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth0
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 6c:3b:e5:bd:f5:78
Slave queue ID: 0

Slave Interface: eth1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 6c:3b:e5:bd:f5:7c
Slave queue ID: 0
```

Sve opcije za naše mrežno sučelje bond0 su vidljive i u direktoriju /sys/class/net/bond0/bonding. Sve ove (ili većina opcija može se mijenjati u radu) promjenom vrijednost odnosno sadržaja datoteka, od kojih ćemo spomenuti njih sâmo nekoliko:

| Datoteka | Značenje |
|---|---|
| /sys/class/net/bond0/bonding/miimon | Ovo je vrijednost miimon vremena, svakih koliko milisekundi će se provjeravati stanje (ispravnost) fizičkih mrežnih kartica unutar *bond-a*. Ako je neka mrežna kartica neispravna, izbacuje se iz *bond-a. Mi ovdje imamo vrijednost 100* (što znači *100 ms).* |
| /sys/class/net/bond0/bonding/mode | Ovdje je postavljen *bond* način rada (*mode*).<br>Mi ovdje imamo vrijednost: *active-backup* ***(1)***. |

Ove vrijednost možemo mijenjati „*u letu*“. Primjerice privremeno  povećajmo miimon vrijeme na 250 milisekundi:

```
echo 250 > /sys/class/net/bond0/bonding/miimon
```

Ovisno o *bonding modu* rada, dostupne su i mnoge druge opcije i parametri. U sljedećem poglavlju ćemo napraviti drugu konfiguraciju *bondinga*, u kojemu ćemo promijeniti način *bondanja* odnosno koristit ćemo napredniji *bonding mod* broj četiri (**4**) koji koristi protokol **802.3ad** odnosno *LACP* kod kojega moramo imati konfigurabilni preklopnik, koji mora podržavati ***LACP*** protokol.

**Ručna konfiguracija Linux bond sučelja (Red Hat 6.x – 8.x)**

Vidjeli smo kako trajno konfigurirati sustav da koristi Linux ***bond***, pomoću ***ifcfg*** datoteka. Međutim moguće je i ručno kreirati *bond* sučelje i u njega dodati željene fizičke mrežne kartice.

Prvo moramo imati instaliran softverski paket iputils, koji je obično već instaliran, ali ako nije instalirajmo ga ovako:

```
yum -y install iputils
```

Sada ćemo kreirati *bond* sučelje imena bond0 kojem će biti dodijeljene fizičke mrežne kartice eth0 i eth1.

Koristit ćemo novu naredbu ifenslave koja se koristi za rad s ***bond*** sučeljima.

```
ifenslave bond0 eth0 eth1
```

Sada novom *bond* sučelju trebamo dodijeliti IP adresu i aktivirati ga:

```
ip addr add 192.168.100.120/24 dev bond0
ip link set bond0 up
```

U svakom trenutku možemo i izbaciti pojedinu mrežnu karticu iz *bonda*; primjerice eth1:

```
ifenslave -d bond0 eth1
```

Ili ju ponovno vratiti u isti *bond* imena bond0:

```
ifenslave bond0 eth1
```

**Za Red Hat 9.x** to postižemo s naredbom nmcli (dostupnom i od ***RedHat 8.x)***. Pogledajmo kreiranje ***bond*** sučelja (bond0):

```
nmcli connection add type bond con-name "Bondconn1" ifname bond0
nmcli connection add type ethernet slave-type bond con-name bond0-if1 ifname eth0
master bond0
nmcli connection add type ethernet slave-type bond con-name bond0-if2 ifname eth1
master bond0
```

Možemo i mijenjati vrstu *bond* sučelja, primjerice u „*active-backup*:

```
nmcli connection modify Bondconn1 bond.options "mode=active-backup"
```

**Izvori informacija: (413),(415),(417),(749),(750),(1340),(1341),(1342),** **man ifenslave**,**man nmcli**, **plakat P5**

### 20.6.6.3. *IEEE 802.3ad* (*802.1AX*-2008) odnosno LACP protokol

***LACP*** (*Link Aggregation Control Protocol*) protokol omogućava aktivni *bonding/agregaciu/teaming* između različitih uređaja te uređaja različitih proizvođača. Unutar specifikacije *Link Aggregation Control Protocol* (***LACP***) pruža metodu za kontrolu povezivanja nekoliko fizičkih mrežnih sučelja (mrežnih kartica) zajedno u jedno logičko mrežno sučelje, odnosno u jedan logički komunikacijski kanal. *LACP* protokol omogućuje mrežnom uređaju da pregovara automatsko povezivanje fizičkih mrežnih sučelja slanjem *LACP* paketa direktno na povezani uređaj koji također podržava *LACP* protokol.
Dakle pomoću *LACP* protokola se dvije strane; primjerice Linux poslužitelj i konfigurabilni preklopnik koji podržava *LACP*, dogovaraju kako će se kreirati logičko *LACP* mrežno sučelje koje se sastoji od fizičkih mrežnih sučelja unutar njega.
Oni se osim toga dogovaraju odnosno komuniciraju o stanju fizičkih sučelja unutar *LACP* komunikacijskog kanala odnosno *LACP* logičkog sučelja, a koje se često naziva ***port channel***.
Za ***LACP*** protokol je važno znati:

- Da je maksimalan broj fizičkih mrežnih sučelja unutar jednog LACP kanala (*bonda*) obično osam (8).
- Da se LACP mrežni paketi šalju na *multicast* (grupnu) adresu `01:80:c2:00:00:02`.
- Da se u trenutku inicijalnog povezivanja odnosno detekcije LACP uređaja LACP paketi šalju svake sekunde, a potom se šalju tzv. ***keepalive*** poruke (poruke da je druga strana „živa“) na jedan od dva moguća načina:
    - Standardno: `slow` - svakih 30 sekundi.
    - Ili ubrzano: `fast` - svaku jednu (1) sekundu.

    Naime u radu se svakih (`slow` ili `fast`) sekundi šalju ***LACP PDU*** poruke između uređaja koji su povezani i koriste LACP protokol.
- Da svaki LACP kanal (*bond*) može imati svoj (*per port-channel*) *load balancing* mehanizam raspodjeljivanja mrežnih paketa. Dakle svako logičko *bond* sučelje može koristiti drugačiji algoritam (ako je to potrebno).
- Postoje dva načina rada LACP protokola:
    - `Active` - ovo je način u kojemu se LACP aktivira tako da ova strana inicira otvaranje *LACP* kanala, slanjem *LACPDU* poruka drugoj strani, očekujući dalju razmjenu *LACPDU* poruka (*LACP* uloga: ***Actor***).
    - `Passive` - ovo je način u kojemu se LACP aktivira sâmo, ako ga podržava druga strana, odnosno ako je ispravno detektiran LACP uređaj s druge strane, odnosno slušaju se *LACPDU* poruke (*LACP* uloga: ***Partner***).

LACP funkcionira slanjem mrežnih okvira ***LACPDU (****Link Aggregation Control Protocol Data Units****)*** na sva mrežna sučelja (mrežne kartice) koja imaju omogućen LACP protokol. Ako je pronađen uređaj na drugom kraju veze koji također ima omogućen LACP on će također samostalno slati *LACPDU* mrežne okvire preko svih mrežnih sučelja između kojih ostvarujemo *bond/LACP* kanal. Kako smo vidjeli LACP se može konfigurirati u jednom od dva načina: ***aktivni*** ili ***pasivni***. U *aktivnom* načinu uvijek će se slati *LACPDU* mrežni okviri kroz aktivna mrežna sučelja unutar *bonda/LACP* kanala. Međutim u *pasivnom* načinu rada, *LACPDU* poruke na strani koja je konfigurirana kao pasivna se ne šalju, već se na njih čeka od aktivne strane i stoga se može koristiti kao način nadziranja slučajnih petlji i to sve dok je drugi uređaj odnosno druga strana u aktivnom načinu rada.

## Oblik LACPDU poruke (paketa)

U ovoj naprednoj cjelini obratit ćemo pažnju na oblik *LACPDU* poruke odnosno mrežnog paketa. Naime *LACPDU* poruke šalju se bez korištenja IP sloja (OSI 3) niti TCP/UDP kao transportnih slojeva (OSI 4). Pogledajmo jedan *LACPDU* paket koji šalje aktivni preklopnik na drugi preklopnik. Gledamo sâmo vezu: ***Preklopnik 1*** *(sučelje 22)* → ***Preklopnik 2*** *(sučelje 25):*

```
(1) Frame 1: 124 bytes on wire (992 bits), 124 bytes captured (992 bits)
(2) Ethernet II, Src:(00:13:c4:12:0f:0d),Dst:Slow-Protocols [0x8809]
(01:80:c2:00:00:02)
(3) Slow Protocols
     Slow Protocols subtype: LACP (0x01)
(4) Link Aggregation Control Protocol
      LACP Version: 0x01
       TLV Type: Actor Information (0x01)
        TLV Length: 0x14
        Actor System Priority: 32768
        Actor System ID: (00:13:c4:12:0f:0d)
        Actor Key: 13
        Actor Port Priority: 32768
        Actor Port: 22
        Actor State: 0x85, LACP Activity, Aggregation, Expired
        [Actor State Flags: E****G*A]
        Reserved: 000000
       TLV Type: Partner Information (0x02)
        TLV Length: 0x14
        Partner System Priority: 32768
        Partner System: (00:0e:83:16:f5:00)
        Partner Key: 13
        Partner Port Priority: 32768
        Partner Port: 25
        Partner State: 0x36, LACP Timeout, Aggregation, Collecting, Distributing
        [Partner State Flags: **DC*GS*]
        Reserved: 000000
```

```
TLV Type: Collector Information (0x03)
       TLV Length: 0x10
       Collector Max Delay: 32768
       Reserved: 000000000000000000000000
     TLV Type: Terminator (0x00)
       TLV Length: 0x00
    Pad: 000000000000000000000000000000000000000000000000…
```

Na OSI sloju dva `(2)` vidimo da ***Cisco*** preklopnik s MAC adresom `00:13:c4:12:0f:0d` šalje poruku na MAC adresu: `01:80:c2:00:00:02` koja predstavlja multicast adresu za takozvani „*Slow protokol*“ [identifikator u ***EtherType*** polju je `0x8809` **(1195)**] na koju se šalje mrežni paket. Na OSI sloju tri `(3)` vidimo da se radi o *LACP* poruci (identifikator: `0x01`) poslije kojega na sljedećem OSI sloju četiri `(4)` slijedi konkretna *LACP*(*PDU*) poruka. Sada pogledajmo barem neka osnovna polja poruke na OSI sloju četiri `(4)` gdje vidimo inačicu *LACP* protokola: `0x01`. Potom slijede tri cjeline:

- Cjelina koja predstavlja aktivnog člana odnosno onoga koji šalje poruku (*LACP* uloga: ***Actor***): `TLV Type: Actor`:
  - Vidimo njegov prioritet (`Actor System Priority`): `32768`
  - Potom vidimo njegov identifikator odnosno MAC adresu sučelja (`Actor System ID`): `00:13:c4:12:0f:0d`
  - Zatim vidimo koju vrijednost ključa (`Actor Key`) je postavio: `13` (on se u komunikaciji s drugom stranom prenosi)
  - Slijedi prioritet konkretnog mrežno sučelja (`Actor Port Priority`): `32768`.
  - Te konkretno mrežno sučelje (*port*) za koji ili preko kojeg se šalje ova *LACPDU* poruka (`Actor Port`): `22`
  - Slijedi stanje “*Actora*” (`Actor State`): `0x85` koje je **85** heksadecimalno prevedeno u binarno: ***1000 0101*** što se čita iz tablice[pogledajte opis bitova ove tablice] kao vrijednost[čita se s desna na lijevo, po bitovima]. Konkretno: ***LACP_Activity, Aggregation, Expired***

Tablica s definicijom stanja *LACP kanala prema bitovima*:

| | | | |
|---|---|---|---|
| ***Bit 0:*** *LACP_Activity* | ***Bit 1:*** *LACP_Timeout* | ***Bit 2:*** *Aggregation* | ***Bit 3:*** *Syncronization* |
| ***Bit 4:*** *Collecting* | ***Bit 5:*** *Distributing* | ***Bit 6:*** *Defaulted* | ***Bit 7:*** *Expired* |

- Cjelina koja predstavlja onoga koji prima poruku (*LACP* uloga: ***Partner***) odnosno drugu stranu u komunikaciji:
  - Ovdje se nalaze ista polja (s drugim vrijednostima) kao i za aktivnog člana.
- Cjelina koja predstavlja Tzv. „*Collectora*“ : `TLV Type: Collector Information (0x03)` se isto odnosi na pasivnog člana odnosno drugu stranu
- Te završetak *LACPDU* poruke: `TLV Type: Terminator (0x00)`

Nakon ovog inicijalnog ***LACPDU*** paketa koji šalje aktivni uređaj prema pasivnom: ***Actor*** → ***Partner*** slijedi nastavak komunikacije u vidu razmjene paketa. Zatim slijede paketi kojima se potvrđuje stanje i/ili stabilnost odnosno dostupnost konkretnog mrežnog sučelja i sâmim time pripadajuće međuveze s drugim uređajem (drugom stranom u komunikaciji).

**Izvori informacija: (798),(799),(800),(801),(1195) te primjer LACPDU poruke/paketa***

Krećemo na opis rada i konfiguraciju ***LACP*** protokola. Naime *LACP* nam omogućuje distribuciju Ethernet okvira na sva fizička mrežna sučelja (mrežne kartice) unutar *bond/LACP* kanala. Jasno je kako će time ukupna propusnost premašiti brzinu prijenosa podataka jednog fizičkog mrežnog sučelja. Pri tome ***IEEE**** standard ne definira specifični algoritam za distribuciju mrežnih okvira (*paketa*) odnosno „*Frame Distribution*“ mehanizam, već daje dvije vrlo važne smjernice:

- Redoslijed mrežnih okvira (*paketa*) za određeni podatkovni promet paketa za komunikaciju ne može se mijenjati. Zahtjev za održavanjem točnog redoslijeda slanja mrežnih okvira ispunjen je time da se svi mrežni okviri koji čine određeni komunikacijski kanal; primjerice u komunikaciji dva računala: **PC1 → PC2** smiju slati **SAMO i isključivo** preko jednog (istog) mrežnog sučelja (mrežne kartice) unutar ***bond/LACP*** kanala, točnim redoslijedom kojim ih kreira klijent (pr. **PC1**). Komunikacija između druge dvije točke u komunikaciji (pr. **PC3 → PC4**) isto tako mora ići uvijek preko drugog (ali uvijek istog) mrežnog sučelja i tako dalje.
- Mrežni okviri (*paketi*) se ne smiju duplicirati.

Dakle prema ovim naputcima svaki proizvođač: ***Cisco***, ***Juniper***, ***Arista, Huawei, HP***, ***Brocade*** i drugi, kao i implementacija unutar Linuxa i drugih operativnih sustava može biti malo drugačija, uz uvjet da se svi drže gore navedenih smjernica*.



Prije nego krenemo dalje, važno je razumjeti da se povezivanjem primjerice dvije gigabitne mrežne kartice (2 x 1Gbps) u jedan bond/LACP kanal upotrebom LACP protokola ukupna propusnost udvostručila. Međutim propusnost svake pojedine veze odnosno komunikacijskog kanala između dva računala koja komuniciraju preko tog bond/LACP kanala je maksimalna brzini jednog fizičkog mrežnog sučelja, dakle u ovom slučaju 1Gbps.

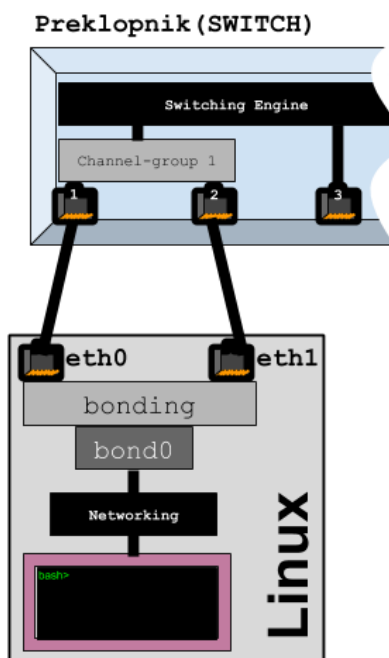
***Kako se šalju podaci kroz bond/LACP komunikacijski kanal odnosno logičko LACP mrežno sučelje?***

Svaka strana: u našem slučaju zasebno Linux, a zasebno preklopnik postavlja svoj algoritam za odlučivanje na osnovu kojega će se za svako računalo (ili uređaj) koji šalje podatke (odnosno mrežne okvire/pakete) kroz *bond/LACP* kanal, odabrati na koje mrežno sučelje (mrežnu karticu) unutar *bond/LACP* kanala poslati te mrežne okvire. Oni će od tog trena prolaziti sâmo i isključivo kroz njega, sve dok ova komunikacija između konkretnih krajnjih točaka u komunikaciji i traje.

Pogledat ćemo sâmo Linux stranu, jer se isto događa i na strani preklopnika. Naime, ako imamo naš linux poslužitelj s dvije mrežne kartice koje su unutar jednog *bond*/LACP kanala odnosno *bond* logičkog mrežnog sučelja, koji želi poslati neke podatke na mrežu, a ovismo o ovom algoritmu, događa se sljedeće.

Ako je algoritam primjerice `layer2` tada će se dogoditi sljedeće; prema shemi spajanja, vidljivoj na slici 176.

**Slika 176. Detaljniji prikaz povezivanje preklopnika s Linuxom.**




Fizički je naša `eth0` mrežna kartica spojena na preklopnik, na njegovo *sučelje* broj **1**. (*Cisco* oznaka `Gi0/1`), a naša druga mrežna kartica `eth1` na preklopnik, na njegovo mrežno sučelje broj **2**. (*Cisco* oznaka `Gi0/2`).

***Cisco** bonding* naziva ***EtherChannel***, a *EtherChannel* grupu naziva *Channel-group*. To znači da unutar jedne *Channel-groupe* dodajemo mrežna sučelja preklopnika (*portove*) za koje želimo da budu unutar tog logičkog mrežnog sučelja, kao što su naše fizičke mrežne kartice `eth0` i `eth1` unutar linuxovog logičkog mrežnog sučelja `bond0`, koje možemo promatrati kao jednu *Channel-group*-u.

Prema odabranom algoritmu za slanje `layer2` za mrežni paket koji se treba poslati na mrežu, izračunava se ***hash*** vrijednost prema sljedećoj formuli:
(ovo vrijedi samo za `layer2` algoritam)

***hash*** = (***source MAC** adresa* ***XOR destination MAC** adresa* ***XOR*** packet type ***ID***)

Nakon što je izračunata *hash* vrijednost za ovaj (i svaki pojedini mrežni paket) odabire se mrežna kartica preko koje će se poslati ovaj mrežni paket i to prema formuli:

***odabrana** slave* ***mrežna kartica*** = (***hash***) ***modulo*** (***ukupan broj*** *slave* ***kartica***)

Ova metoda se radi za svaki pojedini mrežni paket koji dođe do sloja označenog na slici kao `bond0`.

Za više detalja o primjerice `bond0` sučelju, možete pokrenuti naredbu:

```
ip -d link show bond0
```

Sada pogledajmo konfiguraciju na Linuxu za konfiguraciju *LACP* protokola, prema načinu spajanja vidljivom na navedenoj slici 176.

Konfiguracijska datoteka (*ifcfg*) za:

- `eth0` mrežno sučelje je `/etc/sysconfig/network-scripts/ifcfg-eth0`
- `eth1` mrežno sučelje je `/etc/sysconfig/network-scripts/ifcfg-eth1`

***Bonding mod*** dio konfiguracije se ne radi na razini ovih mrežnih sučelja jer se konfiguracija *bond* moda (`bond0`) radi u takozvanoj ***ifcfg*** datoteci: `/etc/sysconfig/network-scripts/ifcfg-bond0` odnosno u `bond0` sučelju.

Od **Red Hat 9.x** zadana konfiguracija se radi pomoću servisa ***NetworkManager*** i pripadajuće naredbe `nmcli`.

Za detalje pogledajte poglavlje: **25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**

Mi stoga u ***ifcfg*** datoteci mijenjamo sâmo redak u kojem definiramo *bonding* način rada, odnosno redak koji je do sada bio:

```
BONDING_OPTS="miimon=100 mode=active-backup"
```

Ovdje mijenjamo gornji redak odnosno dio konfiguracije, u sljedeći redak:

```
BONDING_OPTS="miimon=100 mode=4 lacp_rate=fast xmit_hash_policy=layer2+3"
```

I s time smo prešli u ***bonding** mode* broj **4** odnosno upotrebu *LACP* protokola za uspostavljanje *bonda/agregacije*.

**Što smo ovdje sve dodali kao opcije?**

- `lacp_rate=fast` - ova opcija govori *LACP* protokolu kako će se *LACP* poruke standardno slati/razmjenjivati brže (moguće je slati ih i normalno [`=slow`]).
- `xmit_hash_policy` - ova opcija označava koji algoritam će se koristiti za odabir mrežne konekcije (fizičke mrežne kartice) unutar *bond/LACP* kanala, preko koje će se slati (određeni) mrežni paketi.

Vezano za `xmit_hash_policy` - trenutno dostupni Linux kernel modul `bonding` nudi sljedeće algoritme:

- `layer2` - njega smo opisali; dakle radi odluku o slanju mrežnih okvira (paketa) na osnovi *MAC* adrese.
- `layer2+3` - radi slično kao prethodni, ali se za izračuna *hash* vrijednosti koriste i izvorišne (*source)* i odredišne (*destination)* IP adrese, uz MAC adrese.
- `layer3+4` - radi slično kao prethodni (OSI 2+3), ali se za izračuna *hash* vrijednosti koriste prvo *source* i *destination portovi* pa potom *source* i *destinaton* IP adrese.
- `encap2+3` - koristi istu formulu za *hash* kao `layer2+3`, ali se koristi i zaglavlje od enkapsulacije, ako se koristi neka dodatna enkapsulacija, kao što se koristi za primjerice tuneliranje, VPN-ove i slično.
- `encap3+4` - koristi istu formulu za *hash* kao `layer3+4`, ali se koristi i zaglavlje od enkapsulacije, ako se koristi neka dodatna enkapsulacija, kao što se koristi za primjerice tuneliranje, VPN-ove i slično.



Važno je razumjeti da i preklopnik ima postavljen (neki) svoj algoritam, koji sada nećemo objašnjavati, jer ovisi o vrsti, modelu i inačici softvera preklopnika. Međutim za pravu optimizaciju rada mreže potrebno je testirati koje dvije kombinacije algoritama: jedna sa strane linuxa, a druga sa strane preklopnika daju najbolje rezultate za konkretnu primjenu.

U slučaju kada želimo mijenjati ovaj algoritam u radu, njega možemo postaviti u varijabli:
`/sys/class/net/bond0/bonding/xmit_hash_policy`
Ista varijabla za algoritam se koristi, osim za *bond*: ***LACP*** (***mode=4***) i za sljedeće *bond* načine rada:

- **`balance-xor`** (**mode**=2)
- **`balance-tlb`** (**mode**=5)

Sada pogledajmo kako sâda izgleda naša aktivna konfiguracija `bond0` mrežnog sučelja (kako ju vidi kernel):

```
cat /proc/net/bonding/bond0

Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2+3 (2)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: fast
Min links: 0
Aggregator selection policy (ad_select): stable
Active Aggregator Info:
  Aggregator ID: 1
  Number of ports: 1
  Actor Key: 9
  Partner Key: 550
  Partner Mac Address: 00:24:04:ba:ac:12

Slave Interface: eth0
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 6c:3b:e5:bd:f5:78
Aggregator ID: 1
Slave queue ID: 0

Slave Interface: eth1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 6c:3b:e5:bd:f5:7c
Aggregator ID: 2
Slave queue ID: 0
```

Potom pogledajmo kako napraviti osnovnu konfiguraciju ***Cisco*** preklopnika (***Cisco IOS***) na njegovim *portovima* 1 i 2 (***Gi0/1*** i ***Gi0/2***) i to sâmo dio vezan za *LACP* i za *LACP* (*bond*) grupu broj jedan (**1**). Pogledajmo konfiguraciju *portova* na preklopniku:

```
configure terminal
interface range GigabitEthernet 0/1 – 2
channel-group 1 mode active
channel-protocol lacp
```

I sada pogledajmo konfiguraciju *LACP* grupe broj `1`. Pri tome je konfiguracija u ***ACCESS*** načinu rada (nema *VLANova*):

```
configure terminal
interface port-channel 1
lacp rate fast
switchport mode access
```

Vidimo i da smo LACP brzinu slanja *LACPDU* smanjili na `fast` jer smo isto napravili i na strani Linuxa s kojim smo spojeni.


***LACP protokol radi u granicama OSI sloja dva (OSI 2), ne koristeći niti IP niti TCP/UDP protokole za prijenos pa samim time može egzistirati isključivo unutar lokalne (LAN) mreže.***

**Izvori informacija: (409),(410),(411),(412),(413),(414),(415),(416),(417),(418),(419),(420), `man ip link`, `man ip`.**

## 20.6.7. Link Layer Discovery Protocol (LLDP)

**Slijedi napredna cjelina!**
*Link Layer Discovery Protocol* (***LLDP***) je protokol koji radi na drugom OSI sloju mreže (OSI 2). On se u konačnici koristi za identifikaciju uređaja na mreži. Osim što nam *LLDP* može dati osnovne informacije o uređajima (računala, poslužitelji, usmjerivači, preklopnici, …) odnosno identitetu uređaja na mreži on prenosi i neke detaljnije informacije.
LLDP je službeno definiran u IEEE ***802.1AB*** standardu s dodatnom podrškom u ***IEEE 802.3*** odjeljku 6, klauzula 79. *LLDP* obavlja funkcije slične kao i nekoliko vlasničkih (licenčno zaštićenih) protokola, kao što su *Cisco Discovery Protocol* (*CDP*), *Foundry Discovery Protocol (FDP), Nortel Discovery Protocol (*NDP) i *Link Layer Topology Discovery* (LLTD).
Informacije prikupljene pomoću LLDP-a mogu se pohraniti u informacijsku bazu upravljanja uređajem to jest tzv. "management information base" (***MIB***) i zatražiti pomoću *Simple Network Management Protocol* (***SNMP***) kako je navedeno u **RFC 2922**. Topologija mreže s omogućenim LLDP-om može se otkriti indeksiranjem mrežnih uređaja i upitima na ovu bazu podataka. Informacije koje se mogu dohvatiti preko LLDP-a uključuju:

- Ime uređaja i njegov opis.
- Oznaku mrežnog sučelja i njegov opis.
- Oznaku pripadnosti VLAN-a (ako postoji).
- IP i MAC adresu mrežnog sučelja te ***MDI*** stanja rada mrežnog sučelja.
- Mogućnosti uređaja (usmjerivač [*router*], preklopnik [*switch*], …).
- Agregacija mrežnih sučelja.

Pogledajte tablicu s mogućnostima sustava koje se mogu oglašavati za svaki LLDP uređaj:

| Kôd/oznaka mogućnosti uređaja | Mogućnost uređaja |
|---|---|
| **B** | Mrežni most ili preklopnik (*bridge/switch*). |
| **C** | Poseban kablovski uređaj (*DOCSIS Cable device*). |
| **O** | Neka druga vrsta uređaja (*other*). |
| **P** | *Repeater*. |
| **R** | Usmjerivač (*router*). |
| **S** | Radna stanica/računalo (*station*). |
| **T** | Telefon (*telephone*). |
| **W** | Bežična pristupna točka (*WLAN access point*). |

*LLDP* informacije o sebi šalju uređaji iz svakog od svojih sučelja (ako je tako konfigurirano) u fiksnom intervalu, u obliku *Ethernet* mrežnih okvira. Druga strana to jest uređaj za primanje zatim prikuplja te podatke, dodaje iz kojeg je mrežnog sučelja saznao da podaci dolaze i pohranjuje ih određeno vrijeme. Podaci se razmjenjuju samo između uređaja izravno povezanih putem Ethernet (***LAN***) mreže, tako da je moguće točno znati koji je susjedni uređaj na određenom mrežnom sučelju.
Svaki mrežni okvir sadrži jednu LLDP jedinicu podataka zvanu *LLDP Data Unit* (**LLDPDU**).
Svaki LLDPDU čini niz sastavljen od takozvane *type-length-value* (**TLV**) strukture koja se sastoji od definirane vrste i duljine vrijednosti. Ethernet okvir koji se koristi u *LLDP*-u ima svoju odredišnu MAC adresu postavljenu na posebnu višesmjernu adresu (*multicast* adresu) koju mrežni mostovi (premosnici) ili preklopnici usklađeni sa standardom ***802.1D*** ne prosljeđuju. Dopuštene su i ostale višesmjerne i jednosmjerne odredišne adrese. Polje ***EtherType*** postavljeno je na vrijednost ***0x88cc***.

Svaki LLDP okvir započinje sljedećim obveznim TLV poljima:: ID šasije, ID mrežnog sučelja i vrijeme života paketa (***TTL***). Nakon obveznih TLV polja slijedi bilo koji broj opcionalnih TLV-a. Okvir po želji završava posebnim TLV-om, koji naznačava kraj LLDPDU-a u kojem su i polja vrste i duljine nula (0).

Pogledajmo *Ethernet* okvir LLDPDU-a:

| ***Preambula*** | Odredišna MAC adresa | Izvorišna MAC adresa | EtherType | ID šasije (**TLV**) | ID mrež. sučelja (**TLV**) | TTL (TLV) | Opcionalni TLV-ovi | Završni LLDPU TLV | FCS polje |
|---|---|---|---|---|---|---|---|---|---|
| | **Posebne multicast MAC adrese** | Adresa uređaja | `0x88CC` | Type=1 | Type=2 | Type=3 | Niti jedan ili više njih | Type=0 Length=0 | |

**Posebne multicast MAC adrese za LLDP mogu biti sljedeće adrese:**

- `01:80:c2:00:00:0e`
- `01:80:c2:00:00:03`
- `01:80:c2:00:00:00`



Vezano za MAC adrese, pogledajte poglavlje:
**19.3.1. Što su MAC adrese.**

Opcionalni (proizvoljni) TLV-ovi se mogu kreirati na način da se kao proizvoljni TLV stavi oznaka (***Type=127***) s pripadajućim podvrstama. Postoji i nadogradnja LLDP-a koja se zove ***Media Endpoint Discovery***, poznata i kao ***LLDP-MED***, koja pruža sljedeće sadržaje:

- Automatsko otkrivanje LAN pravila kao što su postavke VLAN-ova, postavljanje prioriteta na OSI sloju dva (OSI 2i diferenciranih usluga (*Diffserv*) koje omogućuju korištenje takozvanih *Plug and Play* usluga.
- Otkrivanje lokacije uređaja kako bi se omogućilo stvaranje baza podataka o lokaciji i, u slučaju protokola *Voice over Internet Protocol* (*VoIP*), poboljšanih usluga prema pozivnim brojevima za nužde (911 u Americi i 112 u Europi).
- Prošireno i automatizirano upravljanje napajanjem krajnjih točaka *Power over Ethernet* (PoE).
- Upravljanje inventarom, dopuštajući mrežnim administratorima da prate svoje mrežne uređaje i određuju njihove karakteristike (inačice proizvoda, proizvođače, softver i hardvera, serijski broj ili broj inventara [imovine]).

Proširenje protokola LLDP-MED službeno je odobrilo i objavilo kao standardni ***ANSI /TIA-1057*** od strane Udruženja telekomunikacijske industrije (***TIA***) u travnju 2006. godine.

**Konfiguracija LLDP pod Linuxom**

LLDP servis pod većinom distribucija Linuxa nije standardno instaliran. Trenutno (2022.g.) najviše su u potrebi dva LLDP servisa za istu namjenu. Mi smo se odlučili za `lldpd` servis. Instalirajmo ga na prvo računalo (poslužitelj):

```
yum -y install epel-release
yum -y install lldpd.x86_64
```

Zatim ga pokrenimo i trajno aktivirajmo (za ***RedHat 7.x+***):

```
systemctl  enable lldpd
systemctl  start lldpd
```

U radu, a pogotovo za testiranje, možemo koristiti naredbu `lldpcli`; primjerice za prikaz LLDP susjednih uređaja:

```
lldpcli show neighbors
```

Za testiranje, možemo koristiti naredbu `lldpcli`; primjerice za slanje LLDP (LLDPDU) poruka na sva mrežna sučelja:

```
lldpcli update
```

Osim gore navedenog načina pozivanja naredbe `lldpcli`, ona nam nudi i svoju ljusku ,ako ju samo ovako pokrenemo:

```
lldpcli
[lldpcli] #
```

I sada možemo stiskanjem tipke *TAB*, dobiti ponuđeno što se sve nudi od naredbi; primjerice:

```
[lldpcli] #
-- Help
       show  Show running system information
      watch  Monitor neighbor changes
     update  Update information and send LLDPU on all ports
  configure  Change system settings
unconfigure  Unconfigure system settings
       help  Get help on a possible command
      pause  Pause lldpd operations
     resume  Resume lldpd operations
       exit  Exit interpreter
```

Ako upišemo bilo koju naredbu; primjerice `show` i ponovno pritisnemo *TAB*, dobivamo izbor svih dostupnih opcija:

```
show
-- Show running system information
             neighbors  Show neighbors data
            interfaces  Show interfaces data
               chassis  Show local chassis data
            statistics  Show statistics
         configuration  Show running configuration
running-configuration  Show running configuration
```

U slučaju kada želimo posebno konfigurirati naše računalo (poslužitelj/sustav), po pitanju LLDP protokola, željenu konfiguraciju možemo snimiti u datoteku: `/etc/lldpd.d/lldpd.conf`.

Za provjeru, primamo li uredno *LLDP* poruke (s MAC adrese: ***01:80:c2:00:00:0e***), za **eth0** mrežno sučelje:

```
tcpdump -s0 -vv -pni eth0 ether dst 01:80:c2:00:00:0e
```

Isto to možemo napraviti i s filtriranjem *EtherType* polja koje definira LLDP protokol (***0x88cc***):

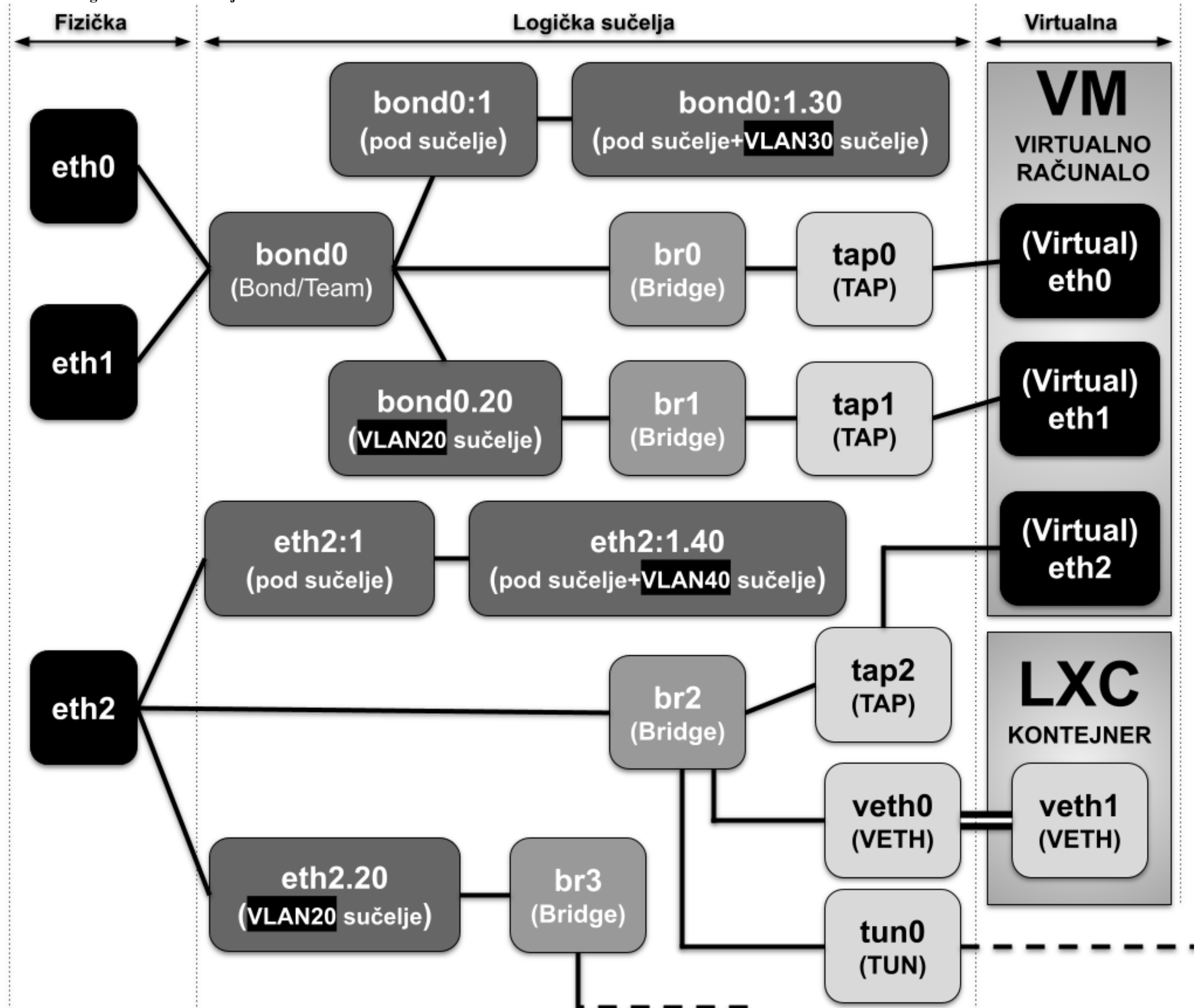
```
tcpdump -s0 -vv -pni eth0 ether proto 0x88cc
```

**Izvori informacija: (1232),(1233),(1234),(1235),(1236),(1237),(1238), *802.1AB-2016, 802.1AB*, `man lldpd`, `man lldpcli`.**

## 20.6.8. Pregled mrežnih sučelja

U ovoj cjelini pregledat ćemo sva do sada spomenuta mrežna sučelja i veze među njima, kako je vidljivo na slici 176.1. dolje.

**Slika 176.1. Pogled na mrežna sučelja.**




Ako gledamo ***fizička*** mrežna sučelja, primjerice: **eth0**, **eth1**, **eno0** i druge, na njima možemo kreirati:

- Pod sučelja (*sub interfaces*), poput: **eth0:1**, **eth0:2**, **eno0:1**, **enp0s1:1**, …
- VLAN sučelja, poput: : **eth0.40**, **eth1.20**, **eno0.20**, **enp0s1.40**, …
- VLAN sučelja na pod sučeljima, poput: **eth0:1.40**, **eth1:1.20**, **eno1:2.20**, **enp0s1:2.40**, …

Fizička mrežna sučelja imaju jedinstvenu MAC adresu dodijeljenu od strane njihovog proizvođača.

Gledamo li s razine **Bond** sučelja; primjerice : **bond0**, **bond1** i drugih, na njima možemo kreirati:

- Pod sučelja (*sub interfaces*), poput: **bond0:1**, **bond0:2**, **bond1:1**, …
- VLAN sučelja, poput: **bond0.20**, **bond0.40**, **bond1.20**, …
- VLAN sučelja na pod sučeljima, poput: **bond0:1.20**, **bond0:3.40**, **bond1:1.20**, …

***Bond*** mrežna sučelja također imaju jedinstvenu MAC adresu jer rade na **OSI** sloju dva (OSI 2).

Pogledajmo sve s točke ***bridge*** mrežnog sučelja odnosno sučelja mrežnog mosta, i što sve s njim možemo ili ne možemo povezivati. U *bridge* MOŽEMO dodavati:

- Fizička mrežna sučelja poput: **eth0**, **eth1**, **eno0**, **eno1**, **enp0s1**, **enp0s2**, …
- Agregirana/team/bond mrežna sučelja poput: **bond0**, **bond1**, **bond2**, …
- VLAN mrežna sučelja poput: **eth0.40**, **eth1.20**, **bond0.20**, **bond1.40**, …
- Virtualna/logička mrežna sučelja poput: **tun0**, **tun1**, **tap2**, **veth0**, **veth1**, …
- Logička VLAN mrežna sučelja (kernel 3.8+) poput: **tun0.10**, **tun1.20**, **veth0.10**, **veth1.20**, …

***Bridge*** mrežna sučelja također imaju jedinstvenu MAC adresu, kao i primjerice **TAP** i **VETH** sučelja jer rade na **OSI 2** sloju.

U ***bridge*** NE MOŽEMO dodavati:

- Pôd sučelja fizičkih mrežnih sučelja, primjerice: **eth0:1**, **eth0:2**, **eno0:1**, **enp0s1:1**, …
- Pôd sučelja agregiranih/team/bond mrežnih sučelja poput: **bond0:1**, **bond0:2**, **bond1:1**, **bond2:1**, …
- Pôd sučelja od VLAN sučelja: **eth0:1.40**, **eth1:1.20**, **eno1:2.20**, **enp0s1:2.40**, …

**Izvori informacija:** `man ifenslave`,`ip link help`,`ip link help bridge`,`man bridge`.

# 20.7. ARP protokol

*Address Resolution Protocol* (***ARP***) je mrežni protokol koji se koristi za prevođenje adresa na OSI sloju dva (sloj mreže [*Network Layer]*) na OSI sloj tri (sloj veze [*Link Layer]*). U *ethernet* mrežama to znači vezu između MAC adrese i njene pripadajuće IP adrese. MAC adrese se nekada nazivaju i hardverske adrese jer su upisane u svaku mrežnu karticu od strane sâmog proizvođača (*Intel, Broadcom, Realtek,* i drugi). ***ARP*** protokol radi na OSI sloju dva što znači da ARP mrežni okviri ne koriste niti IP niti TCP/UDP kao transportne protokole pa samim time mogu egzistirati isključivo unutar lokalne (**LAN**) mreže.

Istovremeno `arp` je i naredba u većini operativnih sustava, koja barata s takozvanim ARP tablicama.
Zbog čega smo gore pričali o generaliziranoj rezoluciji imena: OSI sloj dva → OSI sloj tri. Zbog toga što se ARP ne koristi samo u *Ethernet* mrežama u kojima je na OSI sloju dva MAC adresa već i na drugim mrežama, poput *FDDI, ATM* i *xDSL, X.25*, *Frame Relay* i sličnim. Dalje u tekstu ćemo govoriti isključivo o ***Ethernet*** mrežama pa će stoga biti priče o MAC adresama jer se one nalaze na OSI sloju dva (OSI 2) u *Ethernetu*.
ARP protokol radi prema principu: slanja ARP zahtjeva i primanja ARP odgovora odnosno: *request* i *reply* poruka.
Važno je razumjeti kako se svaki TCP/IP paket, prije nego se uopće pošalje na mrežu, mora sastojati od minimalno:

| **Izvorišna (*Source*) IP adresa** | **Izvorišna (*Source*) MAC adresa** | **Odredišna (*Destination*) IP adresa** | **Odredišna (*Destination*) MAC adresa** | **Ostali dijelovi paketa…** |
|---|---|---|---|---|
| IP adresa pošiljatelja | MAC adresa pošiljatelja | IP adresa primatelja | MAC adresa primatelja | Ovi dijelovi paketa sada nisu toliko važni za razmatranje o ***ARP****u*… |

Naime svaki mrežni uređaj (računalo, poslužitelj, preklopnik, usmjerivač, …) se na mreži identificira sa svojom **MAC adresom** i njenom pripadajućom **IP adresom**. Kada mrežni uređaj želi komunicirati s drugim uređajem u mreži on kreira mrežne pakete koje šalje na mrežu. Da budemo precizniji: ovi paketi (na OSI sloju dva) se zovu mrežni okviri (engl. *Frame*).
Tijekom kreiranja prvog paketa, osim vlastite odnosno izvorišne (*Source*) IP i MAC adrese, u paket se moraju staviti i odredišne IP i MAC adresa. Odredišnu IP adresu znamo jer joj i želimo pristupiti, ali u inicijalnom trenutku nam je odredišna (*Destination*) MAC adresa nepoznata. Naše računalo stoga prvo gleda u svoju lokalnu ARP tablicu koja se sastoji od para: IP adresa i njena pripadajuća MAC adresa, u formatu vidljivom u tablici:

| **IP adresa** | **Pripadajuća MAC Adresa** |
|---|---|
| 10.10.20.10 | 24:FE:48:52:11:4A |
| 10.10.20.18 | 12:48:FA:14:68:AB |

Pošto u inicijalnom trenutku naše računalo poznaje sâmo odredišnu (*Destination*) IP adresu na koju i šalje paket, ali ne i odredišnu MAC adresu, u pomoć nam dolazi *ARP* protokol. U ovom trenutku naše računalo prvo kreira ARP poruku koju šalje na mrežu i u kojoj pita sve koji su na (lokalnoj) mreži (slanjem *Broadcast* poruke); može li netko odgovoriti: koja je *MAC* adresa od naše odredišne (tražene) *IP* adrese.
**SVI** na lokalnoj (***LAN***) mreži primaju ovu ARP poruku, a na nju obično odgovara računalo čija je to IP adresa.
Osim vlasnika, na ovu poruku eventualno može odgovoriti i podrazumijevani usmjerivač odnosno „*Default Gateway*“.
Pri tome se pod pojmom „*Default Gateway*“ misli na naš centralni usmjerivač (*router*) na mreži, obično onaj prema internetu.
Potom nam je naše traženo računalo (s poznatom IP adresom) odgovorilo i dalo nam svoju MAC adresu.
Tek sada naše računalo može kreirati ispravan TCP/IP mrežni paket i poslati ga na mrežu. U ovom trenutku naše računalo osvježava svoju ARP tablicu, odnosno u nju dodaje novi unos za naučeni pâr: odredišna (*destination)* **IP** adresa i odredišna (*destination)* **MAC** adresa, koje je upravo naučio.
Za svako sljedeće kreiranje paketa koji treba ići na ovu odredišnu IP adresu, za koju smo sada naučili MAC adresu, sada se ona pronalazi u našoj lokalnoj ARP tablici te se više ne mora koristiti ARP protokol.
ARP tablica na našem računalu se stalno dopunjavan novo naučenim parovima IP-MAC. Osim toga ARP tablica ima i svoj vijek trajanja, obično oko 1 minutu za *Linux* odnosno oko pet (5) minuta za većinu *Windowsa*.
Nakon isteka ovog vremena, naše računalo briše svoju cijelu ARP tablicu te ponovno kreće u proces učenja i popunjavanja odnosno dopunjavanja iste tablice.



**U slučaju kada ne koristimo *IPv4* već *IPv6* protokol, tada se klasičan *ARP* protokol ne koristi!.**
Naime ***IPv6*** protokol koristi poseban protokol imena *Neighbor Discovery Protocol* (***NDP***) koji unutar sebe nudi zamjenu za ***ARP*** protokol odnosno njegovu funkcionalnost**.** On za tu namjenu koristi posebne **ICMP v6** poruke: ***Type=135 (neighbor solicitation)*** *koji je ekvivalent* ***ARP upitu*** te ***TYPE=136 (neighbor advertisement)*** *koji je ekvivalent* ***ARP odgovoru***, kako je vidljivo u **RFC4861**. Dakle kod upotrebe **IPv6** protokola, umjesto **ARP** poruka, šalju se posebne **ICMP v6** poruke pomoću kojih **NDP** implementira ***ARP*** funkcionalnost.
Osim **ARP** funkcionalnosti, ***NDP*** (odnosno ***IPv6***) je uveo i sljedeće mogućnosti:

- *Otkrivanje ili oglašavanja usmjerivača (**Router advertisement [Type=134]** i **Router Solicitation [Type=133]**).*
- *Poruku o preusmjeravanju na bolji (neki drugi) usmjerivač za odredišnu mrežu (**Redirect [Type=137]**).*
- *Detekciju duplicirane IPv6 adrese na mreži, nedostupnost susjednog uređaja, parametara mreže (pr. MTU).*

***NDP*** protokol je definiran u **RFC4861** te **RFC1122.**

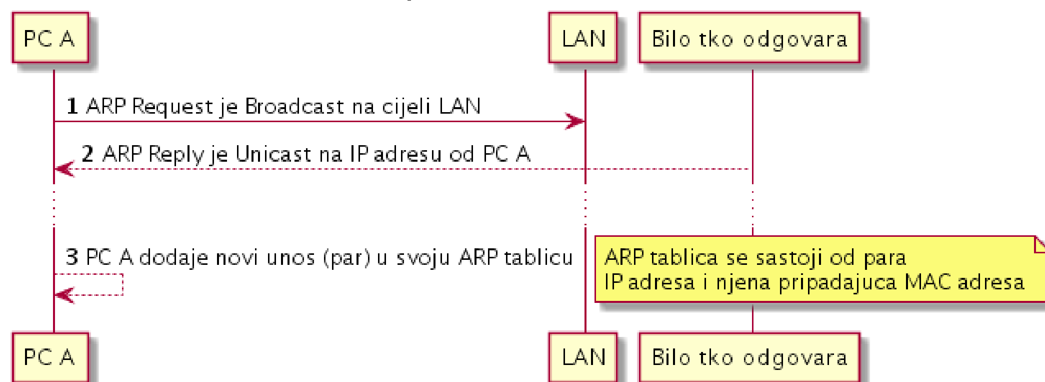
**Izvori informacija: (705), (706), (K-6), (K-10), RFC4861, RFC1122, RFC 826.**

**ARP protokol radi na OSI sloju dva, a u komunikaciji koristi dva scenarija:**

**Standardni scenarij osvježavanja (doznavanja) pripadajuće MAC adrese za traženu IP adresu**. Pri tome se šalje:
1. **ARP upit** odnosno *ARP Request*: **tko ima IP adresu** x.y.z.w
   [*Opcode 1 - Request*] - *Broadcast* poruka
2. **ARP odgovor** odnosno *ARP Reply*: **za IP adresu** x.y.z.w **koristi se MAC adresa AA:BB:CC:DD:EE:FF**
   [*Opcode 2 - Reply*] - *Unicast* poruka

Važno je znati da svaki uređaj na mreži može poslati ***ARP*** zahtjev (*request*) koji se šalje na *Broadcast* MAC adresu `FF:FF:FF:FF:FF:FF` koju primaju sva računala unutar određene ***LAN*** mreže odnosno unutar *Broadcast* domene.
Isto tako bilo tko može odgovoriti na ovaj zahtjev odnosno upit, te poslati ***ARP*** odgovor (*response*) koji je u*nicast* poruka odnosno odgovor se šalje na MAC adresu pošiljatelja *ARP upita (Request-a)*. Ovakvu komunikaciju pogledajmo na slici 177.

**Slika 177. Osnovni način rada ARP protokola.**






O manje standardnom scenariju rada **ARP** protokola, više u poglavlju:
**20.7.1.3. Scenarij za prisilno osvježavanje ARP tablice (*Gratuitous ARP*) detaljnije.**

## 20.7.1.1. Izgled ARP paketa

**ARP** protokol (*Address Resolution Protocol*) kako je definirao standardom RFC 826 ima sljedeću strukturu paketa:

| | | |
|---|---|---|
| Hardware type (HTYPE) | Protocol type (PTYPE) [**Ether Type**] **0x0806** | |
| Hardware length (HLEN) | Protocol length (PLEN) | Operation |
| Sender Hardver address (SHA) | | |
| Sender protocol address (SPA) | | |
| Target hardware address (THA) | | |
| Target protocol address (TPA) | | |

Slijedi opis polja:

- `Hardware type (HTYPE)` - označava vrstu mrežnog hardvera.
  U većini slučajeva je to **Ethernet** koji ima vrijednost `1` odnosno `(0x1)`.
- `Protocol type (PTYPE)` nekada se označava kao ***Ether Type*** - on označava sâm protokol
  - **`0x0806`** - označava **ARP** protokol. U slučaju drugih protokola ovdje su druge vrijednosti [*pogl*. (**1195**)]
- `Hardware length (HLEN)` - veličina polja za (MAC) adrese - ovdje je to uvijek 6 *bajta* (6 okteta).
- `Protocol length (PLEN)` - veličina adrese višeg protokola (za IPv4 je to 4 okteta).
- `Operation` - predstavlja vrstu **ARP** operacije, koja može biti:
  - `1` za upit (*request*).
  - `2` za odgovor (*reply*).
- `Sender hardware address (SHA)` - MAC adresa pošiljatelja zahtjeva (*sender*).
- `Sender protocol address (SPA)` - adresa višeg protokola pošiljatelja. U našem slučaju je to IP adresa pošiljatelja.
- `Target hardware address (THA)` - MAC adresa primatelja zahtjeva (*receiver*), ako je ovo bio ARP upit (*ARP request*) tada je ovo polje prazno jer tražimo ovu MAC adresu. Ako je ovo ARP odgovor (*ARP reply*) tada se ovdje upisuje MAC adresa traženog uređaja.
- `Target protocol address (TPA)` - Odredišna adresa višeg protokola. U našem slučaju je to IP adresa primatelja.

**Izvori informacija: (705),(1195),(K-6), (K-10), RFC 826.**

### 20.7.1.2. Rad s ARP protokolom

Pokrenimo program **Wireshark** te idimo na ***Capture*** potom ***Interfaces*** te odaberimo našu mrežnu karticu (*interface*) i krenimo sa snimanjem mrežnog prometa. Umjesto programa ***Wireshark***, možemo koristiti i program `tcpdump`(*Opisano u poglavlju:* ***25.7.8***).
Naredba `arp -a` ispisuje **ARP** tablicu na ***Windows*** operativnom sustavu, a sâmo naredba `arp` bez prekidača, na *Linuxu*.
Ovako izgleda ARP tablica (za **IPv4**) na linux računalu s kojega testiramo (tablica je izmijenjena i skraćena za potrebe testa):

```
arp

  Address         HWtype     HWaddress            Flags Mask              Iface
  192.168.1.11    ether      02-02-02-02-02-02    C                       eth0
  192.168.1.58    ether      bc-b3-08-98-f5-e2    C               eth0
  192.168.1.61    ether      00-01-e3-b1-4f-27    C               eth0
```

Sada kada smo se uvjerili da unos za traženu IP adresu nemamo u ARP tablici, idemo dalje.
Dovoljno je otvoriti naredbeno sučelje (u Linuxu je to *Shell*) odnosno u Windowsima je to: ***CMD.exe***.



Umjesto naredbe `arp` možemo koristiti i noviju naredbu `ip neighbor` koja dolazi u softverskom paketu `iproute` jer je ona u stanju prikazati **ARP** unose i za **IPv4** i **IPv6** protokole. Naime **IPv6** ima malo drugačije mehanizme vezane za **ARP**.

Ako je odredišna IP adresa: **192.168.1.1** tada upišimo `ping -c3 192.168.1.1` i *Enter*, s čime će se poslati samo tri takozvane *ping* (odnosno *ICMP*) poruke kao u primjeru dolje.

```
ping -c3 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.228 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.197 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.178 ms
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.178/0.201/0.228/0.020 ms
```

Vidimo kako smo poslali 3 *ping* (***ICMP***) poruke te da smo dobili odgovor na sve tri poruke (`0% loss`). Inače nije problem, ako na početku dobijemo manje *ping* odgovora nego upita (poslanih paketa) jer je u pozadini i proces učenja MAC adrese za odredište, kao i slanje *ARP* paketa u tu svrhu. Sada možemo zaustaviti snimanje mrežnog prometa u programu **Wireshark** ili programu `tcpdump`. Pošto ćemo imati veći broj mrežnih paketa, filtrirajmo samo *ARP* pakete. U programu **Wireshark**, u polje filter upišite **`arp`** te stisnite **Apply**.

Sada ćemo vidjeti sâmo *ARP* pakete, kao na slici 178. (dolje).

**Slika 178. Pogled na ARP pakete iz programa *Wireshark*.**

*LAN

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

arp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1063 | 13.386533 | 7c:89:56:4c:5d:d4 | ff:ff:ff:ff:ff:ff | ARP | 60 | Who has 192.168.1.1? Tell 192.168.1.165 |
| 1367 | 18.541426 | 40:8d:5c:57:26:63 | 00:30:05:ae:8d:d7 | ARP | 42 | Who has 192.168.1.254? Tell 192.168.1.103 |
| 1368 | 18.541557 | 00:30:05:ae:8d:d7 | 40:8d:5c:57:26:63 | ARP | 60 | 192.168.1.254 is at 00:30:05:ae:8d:d7 |

```
> Frame 1063: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
> Ethernet II, Src: 7c:89:56:4c:5d:d4, Dst: ff:ff:ff:ff:ff:ff
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 7c:89:56:4c:5d:d4
    Sender IP address: 192.168.1.165
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.1.1
```

**Što se događa u našem testu?**
Pošto smo prvo pokušali poslati *ping* (***ICMP***) paket na mrežu, konkretno na IP adresu: ***192.168.1.1***, a njenu pripadajuću MAC adresu ne znamo, naš sustav će prije nego uopće pošalje *ping* poruku prvo poslati *ARP* upit na mrežu i tražiti da nam netko s mreže odgovori koju MAC adresu im tražena IP adresa ***192.168.1.1***. Tek kada nam netko odgovori imat ćemo sve potrebno da pošaljemo *ping* poruku odnosno dobiti ćemo MAC adresu pripadajuće IP adrese. Ne zaboravite da za slanje IP paketa na mrežu, moramo imati izvorišne i odredišne i IP i MAC adrese. Sada pogledajmo te ARP poruke (pakete) koje smo poslali na mrežu. Uočite kako pogledom na ***ARP*** mrežni promet unutar programa **Wireshark** nisu vidljivi dijelovi ARP paketa na OSI slojevima tri i četiri već samo OSI sloj dva, pa se u gornjem dijelu ne vide IP adrese (OSI 3 sloj) niti transportni protokoli: UDP ili TCP (OSI 4 sloj). To je stoga što se **ARP** protokol oslanja samo na OSI sloj dva. Dakle gledano od OSI sloja dva, vidljivo je da računalo s MAC adresom: `7c:89:56:4c:5d:d4` (naše računalo s kojeg radimo test) šalje *ARP* mrežni paket na MAC adresu: `ff:ff:ff:ff:ff:ff` koja označava ***broadcast*** adresu.

Što znači da će paket biti proslijeđen na sva računala na lokalnoj mreži. Na sljedećem OSI sloju je vidljiv naš ARP protokol (*Address Resolution Protocol*) i u njemu vidimo da je na `Opcode` postavljena vrijednost `1` što označava da se radi o ARP poruci `request` (upit). Dalje se u ARP dijelu poruke vidi i da je IP adresa našeg računala (s kojeg šaljemo poruku): `192.168.1.165` (`Sender IP address`) te se vidi i naša MAC adresa:`7c:89:56:4c:5d:d4`. Zatim se vidi IP adresa za koju i tražimo ARP odgovor: `192.168.1.1`, ali su za njenu MAC adresu (`Sender MAC address`) postavljene sve nule (`00:00:00:00:00:00`) jer ju tražimo s ovim upitom.
U **ARP** odgovoru dolazi poruka slična ovoj, uz razliku da ju šalje onaj tko zna odgovor, te ju šalje direktno onome tko je poslao upit (konkretno na našu ***MAC*** adresu). Odgovor u **ARP** dijelu poruke ima za `Opcode` postavljenu vrijednost `2` `reply` (odgovor) sa popunjenim informacijama koje smo tražili odnosno koja MAC adresa pripada traženoj IP adresi.

Uočite da su na *ARP* paketima u dijelu ***Source*** i ***Destination*** sâmo MAC adrese, kako je vidljivo na slici179 (skratili smo ispis).

**Slika 179. Pogled na ARP pakete iz programa *Wireshark, samo gornji dio analize paketa na mreži.***

*LAN

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

arp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1063 | 13.386533 | 7c:89:56:4c:5d:d4 | ff:ff:ff:ff:ff:ff | ARP | 60 | Who has 192.168.1.1? Tell 192.168.1.165 |
| 1367 | 18.541426 | 40:8d:5c:57:26:63 | 00:30:05:ae:8d:d7 | ARP | 42 | Who has 192.168.1.254? Tell 192.168.1.103 |
| 1368 | 18.541557 | 00:30:05:ae:8d:d7 | 40:8d:5c:57:26:63 | ARP | 60 | 192.168.1.254 is at 00:30:05:ae:8d:d7 |

Vratimo se na `arp` naredbu i Linux te idimo na naš server te pokrenimo `arp` s prekidačem `-n` kako bismo vidjeli IP adrese, a ne imena računala iz tih IP adresa (ispis smo skratili na samo nekoliko unosa):

```
arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.1.10             ether   00:1e:64:54:81:3d   C                     eth0
192.168.1.254            ether   00:1e:64:54:9d:f3   C                     eth0
```

Pri tome je IP adresa `192.168.1.254` adresa našeg usmjerivača (*Default Gateway*) preko kojega dolazimo do vanjskih mreža.
Ispis *ARP* tablice možemo napraviti i s novijom naredbom `ip` na sljedeći način:

```
ip neighbor show
```

Sadržaj *ARP* tablice, nalazi se i u posebnoj datoteci: `/proc/net/arp`. Pogledajmo (skraćeni) sadržaj ove datoteke:

```
cat /proc/net/arp

IP address       HW type     Flags       HW address            Mask     Device
192.168.1.10       0x1         0x2       00:1e:64:54:81:3d       *       eth0
192.168.1.254      0x1        0x2       00:1e:64:54:9d:f3       *       eth0
192.168.1.1        0x1        0x2       d4:ca:6a:74:8d:23       *       eth0
```

Stupac `HW type` označava vrstu mreže (`0x1` označava Ethernet), a polje `Flags` označava vrstu ARP unosa.

Pogledajmo nekoliko mogućih oznaka (`Flags`):

- `0x0` – ovo je (*incomplete*) unos (*MAC-IP*) koji će ubrzo automatski biti obrisan od strane sustava.
- `0x2` – ovo je standardni unos (*MAC-IP*), `0x4` – ovo je permanentni odnosno trajni (*MAC-IP*) unos.
- `0x6` – ovo je ručni (*MAC-IP*) unos.

Dakle ARP tablica se sâma osvježava te dinamički popunjava, ali i briše. Međutim u nekim slučajevima možemo imati potrebu unijeti i statički (ručni unos) ili ga naknadno obrisati. Prvo dodajmo jedan unos; IP: `192.168.1.233` i njene MAC adrese: `00:c0:01:02:1f:cd` na mrežnom sučelju `eth0`:

```
arp -s 192.168.1.233 00:c0:01:02:1f:cd -i eth0
```

Gornji unos (ručno dodavanje) smo mogli napraviti i pomoću naredbe `ip` na sljedeći način:

```
ip neighbor add 192.168.1.233 lladdr 00:c0:01:02:1f:cd dev eth0.
```

Pogledajmo sada ARP tablicu:

```
cat /proc/net/arp

IP address       HW type     Flags       HW address            Mask     Device
192.168.1.233      0x1        0x6       00:c0:01:02:1f:cd       *       eth0
```

Vidimo da je zastavica (`FLAGS`) postavljena na `0x6` jer je ovo ručni unos.
Ručno uneseni unosi se brišu nakon restarta računala. Sada ručno i obrišimo unos koji smo napravili:

```
arp -d 192.168.1.233
```

Pogledajmo kako to sada izgleda (filtrirali smo sâmo ovaj jedan unos koji nas zanima):

```
cat /proc/net/arp

IP address       HW type     Flags       HW address            Mask     Device
192.168.1.233      0x1        0x0       00:c0:01:02:1f:cd       *       eth0
```

Pod `Flags` vidimo kako je njena vrijednost: `0x0` što znači kako je unos označen za brisanje, i biti će ubrzo obrisan.



S ovime smo unos označili da će ga sustav obrisati pomoću tzv. ***ARP garbage collector**-a*. Međutim u većini slučajeva ovi unosi se referenciraju i od strane *tablice usmjeravanja* *(routing)*. Pa ih obično više ne briše *ARP garbage collector* jer je standardni interval čišćenja *routing cache* tablice (*ruting* međumemorije) znatno veći, pa će ih najvjerojatnije obrisati *Routing garbage collector*. Naime vrijeme čišćenja tablice usmjeravanja od strane *Routing garbage collector-a* je obično 300 sekundi (5 minuta).

Moguće je i obrisati cijelu ***ARP*** tablicu, u slučaju kada se na nekom poslužitelju, usmjerivaču ili drugom važnom uređaju na mreži primjerice mijenjala mrežna kartica te, ako ne želimo čekati da istekne vrijeme čuvanja ***ARP*** tablice.

To jednostavno možemo napraviti s naredbom `ip neighbor` iz paketa **iproute2**:

```
ip -s -s neighbor flush all
```

S naredbom `ip neigh` možemo i dodavati i brisati ručne ***ARP*** unose i raditi sve što možemo i s naredbom `arp` s time da s njom možemo baratati i s **IPv6** unosima s kojima naredba `arp` ne može raditi. Pogledajmo ukratko i kako:

| Naredba | Opis |
|---|---|
| `ip neigh show` | Ispiši ARP tablicu. |
| `ip neigh del` **`IP`** `lladdr` **`MAC`** `dev` **`LAN`** | Obriši ARP unos; **IP** je IP adresa, **MAC** je MAC adresa i **LAN** je *ethernet* kartica (eth0 ili eth1, ili ...). |
| `ip neigh add` **`IP`** `lladdr` **`MAC`** `dev` **`LAN`** | Kreiraj ARP unos; **IP** je IP adresa, **MAC** je MAC adresa i **LAN** je *ethernet* kartica (eth0 ili eth1, ili ...). |

Jasno je da Linux privremeno sprema sve unose parova: MAC-IP u svoju ARP tablicu. Ipak, ovdje imamo dvije opcije koje utječu na osvježavanje ove tablice odnosno unosa u njoj, a definiraju se preko `sysctl` naredbe. Naime ARP tablica se popunjava svakom novom spoznajom o novom paru: IP adresa i njena pripadajuće IP adresa. Svi ovi parovi se čuvaju u ARP tablici, a takozvani ***ARP* garbage collector** (***GC***) se pokreće svakih ***N*** sekundi, kako bi provjerio koliki je vijek čuvanja za svaki pojedini unos u ovoj tablici. Dakle za svaki novi ARP unos u tablici se zapisuje vrijeme kada je zapisan. Kada svakom pojedinom unosu istekne njegovo vrijeme čuvanja u ARP tablici on se briše.

Stoga ovdje imamo definirane dvije opcije koje možemo i mijenjati:

- `net.ipv4.neigh.default.gc_interval` – definira svakih koliko sekundi će se pokretati ***ARP* garbage collector** mehanizam čišćenja, te početi provjeravati je li neki od unosa u ARP tablici zastario (istekao). Standardno definirano vrijeme je 30 sekundi, što znači kako se ovaj mehanizam provjere pokreće svakih 30 sekundi.
- `net.ipv4.neigh.default.gc_stale_time` – definira koliko sekundi će se čuvati pojedini ARP unosi u ARP tablici. Nakon što za svaki pojedini unos istekne ovo vrijeme, unos kojemu je isteklo ovo vrijeme, se briše, od strane ***ARP* garbage collector**-a. Standardna vrijednost je 60 sekundi, što znači kako se svaki pojedini zapis u ARP tablici čuva do 60 sekundi (jednu minutu).

Ispišimo trenutno postavljene vrijednosti ovih varijabli, pomoću naredbe `sysctl` na sljedeći način:

```
sysctl -a |grep -i net.ipv4.neigh.default.gc_[is]

net.ipv4.neigh.default.gc_interval = 30
net.ipv4.neigh.default.gc_stale_time = 60
```

Dodatno imamo i tri ***sysctl*** varijable s kojima se definiraju dodatni parametri same ARP tablice. To su sljedeće varijable (opcije):

- `net.ipv4.neigh.default.gc_thresh1` – definira minimalan broj unosa u ARP tablici koji označava da se uopće neće pokretati ***ARP* garbage collector**, ako je broj unosa u ARP tablici manji od broja unosa definiranog ovdje.
- `net.ipv4.neigh.default.gc_thresh2` – definira standardni maksimalni broj unosa koje je moguće pohraniti u ARP tablicu. Ako kojim slučajem broj unosa bude imalo veći od ovog broja ***ARP* garbage collector** će se već nakon pet (5) sekundi pošto se ovaj broj unosa premašio, pokrenuti i početi analizirati treba li koji unos obrisati jer je istekao.
- `net.ipv4.neigh.default.gc_thresh3` - definira stvarni maksimalan broj unosa u ARP tablici, koji se ne može preći. Stoga će se ***ARP* garbage collector** pokrenuti momentalno čim se pokuša preći ovaj broj unosa u ARP tablici.

Ispišimo trenutno postavljene vrijednosti ovih varijabli (vezanih za ***ARP garbage collector***):

```
sysctl -a |grep -i net.ipv4.neigh.default.gc_thresh
net.ipv4.neigh.default.gc_thresh1 = 128
net.ipv4.neigh.default.gc_thresh2 = 512
net.ipv4.neigh.default.gc_thresh3 = 1024
```



Ako želite permanentno (trajno) mijenjati ove vrijednosti, pročitajte poglavlje:
**4.5.8 Naredba *sysctl*.**

Sve navedene vrijednosti mogu se mijenjati i provjeravati u sljedećim *sysctl* varijablama (uz vidljive inicijalne vrijednosti):

```
sysctl -a |grep net.ipv4.neigh.default.gc_
net.ipv4.neigh.default.gc_interval = 30
net.ipv4.neigh.default.gc_stale_time = 60
net.ipv4.neigh.default.gc_thresh1 = 128
net.ipv4.neigh.default.gc_thresh2 = 512
net.ipv4.neigh.default.gc_thresh3 = 1024
```

Za detalje o navedenim varijablama (pr. `gc_XY`), pokrenite:

```
man 7 arp
```

---

Čisto za informaciju, ***Routing cache garbage collector*** koji možda netko može pobrkati s prethodno navedenim ***ARP* garbage collector**-om i njegove postavke su definirane u sljedećim *sysctl* varijablama, za **IPv4** protokol:

```
sysctl -a |grep net.ipv6.route.gc_
net.ipv6.route.gc_elasticity = 9
net.ipv6.route.gc_interval = 30
net.ipv6.route.gc_min_interval = 0
net.ipv6.route.gc_min_interval_ms = 500
net.ipv6.route.gc_thresh = 1024
net.ipv6.route.gc_timeout = 60
```



Za više detalja o optimizaciji *Routing cache garbage collector-a*, pogledajte poglavlje:
**25.6.8 Ruting i optimizacija ruting parametara.**



U slučaju kada ne koristimo ***IPv4*** već ***IPv6*** protokol, tada se klasičan ***ARP*** protokol ne koristi.
Naime ***IPv6*** protokol koristi poseban protokol imena *Neighbor Discovery Protocol* (***NDP***) koji unutar sebe nudi zamjenu za ***ARP*** protokol odnosno njegovu funkcionalnost. On za tu namjenu koristi posebne ICMP v6 poruke: ***Type=135 (neighbor solicitation)*** te ***TYPE=136 (neighbor advertisement)***, kako je vidljivo u RFC4861.

**Izvor informacija: (325),(705),(1406),(K-6),(K-10),** `man arp`, `man ip`, `man 7 arp`, `man sysctl`, `man 7 proc`.

---

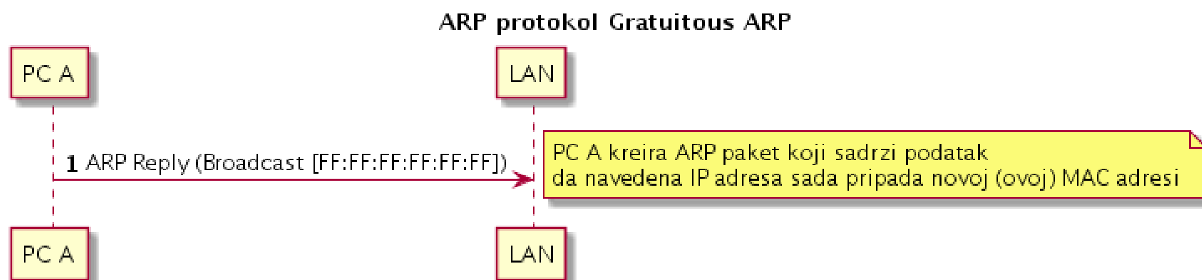
## 20.7.1.3. Scenarij za prisilno osvježavanje ARP tablice (*Gratuitous ARP*) detaljnije

Osim normalnog funkcioniranja *ARP* zahtjeva za provjeru pripadajuće MAC adrese traženoj IP adresi postoji i vrlo specifičan slučaj kada neki mrežni uređaj i to obično usmjerivač (engl. *Router*) ili ***VRRP*** uređaj u posebnoj konfiguraciji, mora poslati informaciju o tome da se njegova MAC adresa promijenila, i to na SVA računala na mreži. Tada sva računala na mreži, trenutno MORAJU osvježiti svoju *ARP* tablicu s ovom promjenom odnosno novom MAC adresom koju je poslao.

Važno je napomenuti da sva računala na mreži to odrađuju trenutno, u roku od nekoliko milisekundi. U ovom slučaju nema potrebe za dva koraka kao u prethodnom slučaju, već bilo koji uređaj može slati ovakvu poruku: ***ARP Reply* [Opcode 2 - Reply]**

Razlika u odnosu na *normalan **ARP** odgovor (Reply)* je u tome što se ova poruka šalje na *Broadcast* MAC adresu tj. `FF:FF:FF:FF:FF:FF` pa ju stoga bezuvjetno moraju primiti sva računala (i uređaji) na mreži, vidljivo na slici **180**.

**Slika 180. Pogled na ARP komunikaciju kod načina rada znanog kao „*Gratuitous ARP*“.**



Primjer ovakvog mrežnog okvira (paketa) je vidljiv na ispisu:

```
Ethernet II, Src: 02:02:02:02:02:02, Dst: ff:ff:ff:ff:ff:ff
    Destination: ff:ff:ff:ff:ff:ff (Broadcast)
    Source: 02:02:02:02:02:02 (02:02:02:02:02:02)
    Type: ARP (0x0806)
    Trailer: 000000000000000000000000000000000000
Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: 02:02:02:02:02:02 (02:02:02:02:02:02)
    Sender IP address: 192.168.1.1 (192.168.1.1)
    Target MAC address: ff:ff:ff:ff:ff:ff (Broadcast)
    Target IP address: 192.168.1.1 (192.168.1.1)
```

Ovdje vidimo da uređaj s MAC adresom `02:02:02:02:02:02` šalje *Broadcast* poruku svima (`FF:FF:FF:FF:FF:FF`) da od sada IP adresa (`192.168.1.1`) pripada MAC adresi (`02:02:02:02:02:02`).
Konkretno je na ovaj način on, sâm sebe, proglasio nadležnim za IP adresu (`192.168.1.1`). Ovo rade mnogi uređaji koji koriste takozvane virtualne IP adrese koje se dijele s drugim uređajima; primjerice ***HSRP***, ***VRRP***, ***GLBP*** te za protokole unutar raznih *clustera*, kako bi se najavila promjena uređaja koji je postao aktivan vlasnik virtualne IP adrese.
Odnosno kako bi poslao svima obavijest da se za virtualnu IP adresu promijenila pripadajuća MAC adresa.



Za detalje pogledajte način funkcioniranja protokola HSRP odnosno novijeg VRRP protokola, kako je upisano u poglavlju:
**23.4.5. Redundancija na OSI sloju tri (OSI 3) i *VRRP* protokol.**

Za test, slanje ***gratuitous ARP*** poruke možemo napraviti i direktno iz Linuxa. Za tu potrebu moramo imati instaliran softverski paket imena `iputils` koji je uglavnom i instaliran, a ako nije možete ga instalirati sa sljedećom naredbom:

```
yum -y install iputils
```

Za ovu potrebu koristit ćemo naredbu `arping` koja dolazi u paketu `iputils`.
Dakle probajmo napraviti sličnu stvar; poslati ***gratuitous ARP*** poruku na mrežu, s kojom govorimo da sada IP adresa: `192.168.1.1` ima pripadajuću MAC adresu naše mrežne kartice `eth0`.

```
arping -U -I eth0 192.168.1.1
```



U slučaju kada ne koristimo ***IPv4*** već ***IPv6*** protokol, tada se klasičan ***gratuitous ARP*** uopće ne može koristiti. Naime ***IPv6*** protokol koristi poseban protokol imena *Neighbor Discovery Protocol* (***NDP***) koji unutar sebe nudi zamjenu za ***gratuitous ARP*** odnosno njegovu funkcionalnost korištenjem posebnih ICMP v6 poruka (***Type=136 neighbor advertisement***). ***NDP*** radi na *Link* sloju mreže, koji je definiran u RFC1122.

Za ispis **IPv6 ARP** unosa, kao i **IPv4** unosa, možemo koristiti sljedeću naredbu:

```
ip neighbor show
```

U slučaju problema u slanju ***gratuitous ARP*** poruka (IPv4), proučite *sysctl* varijablu (i njene vrijednosti): `net.ipv4.conf.all.arp_notify` odnosno za IPv6 proučite: `net.ipv6.conf.all.ndisc_notify`.

| Sysctl varijabla | Opis |
|---|---|
| `net.ipv4.conf.all.arp_notify` | Definirajte način obavijesti o promjenama adrese i uređaja: `0` - (zadano): ne radi ništa, `1` - Generiraj ***gratuitous arp*** zahtjeve kada se uređaj pokrene ili promijeni hardverska adresa. ***Definira se po mrežnom sučelju!*** |
| `net.ipv6.conf.all.ndisc_notify` | Definirajte način obavijesti o promjenama adrese i uređaja: `0` - (zadano): ne radi ništa, `1` - Generiraj „*unsolicited neighbour advertisements*“ kada se uređaj pokrene ili promijeni hardverska adresa. ***Definira se po mrežnom sučelju!*** |

**Izvori informacija: (642),(705),(706),(1406),(K-6),(K-10),** `man arping`.

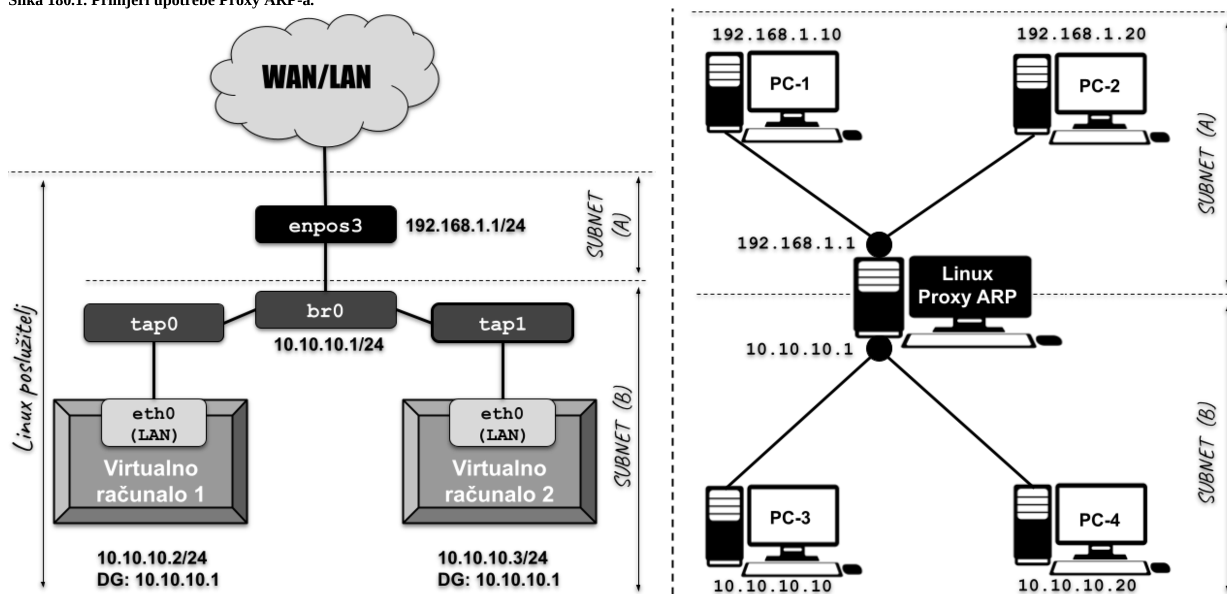
### 20.7.1.4. Proxy ARP i druge opcije

**Proxy ARP**
***Proxy ARP*** je tehnika kojom uređaj na određenoj mreži odgovara na ARP upite za mrežnu adresu koja nije na toj mreži, što znači da računala na jednoj mreži izgledaju kao da su logički dio druge fizičke mreže.
Uređaj koji koristi ovakvu vezu, koji prema logici rada radi kao mrežni most [*bridge*], prosljeđivati će ARP zahtjeve iz svoje unutarnje mreže prema vanjskoj mreži i odgovarati na ARP upite (zahtjeve) iz vanjske mreže u ime unutarnjih računala.
U tom procesu *Proxy ARP* poslužitelj (računalo) na ARP upite za različite tražene IP adrese odgovara sa svojom MAC adresom, stoga jer radi kao posrednički (*proxy*) poslužitelj. Nadalje, Linux će to učiniti samo za računala koja su mu poznata preko njegove tablice usmjeravanja. I na kraju potrebno je omogućiti IP prosljeđivanje jer nakon završetka razmjene ARP mrežnih okvira kreće IP promet, koji je potrebno prosljeđivati između mrežnih sučelja (tzv. proces usmjeravanja koji se odvija na OSI tri sloju mreže [*Layer 3*]).

Ako imamo potrebu koristiti Proxy ARP značajku (aktivira u *sysctl* varijabli `net.ipv4.conf.all.proxy_arp`), potrebno je uključiti i IP prosljeđivanje (engl. *IP forwarding*), što je moguće upotrebom *sysctl* varijable: `net.ipv4.ip_forward`.

| Sysctl varijabla | Opis |
|---|---|
| `net.ipv4.ip_forward` | Standardno je ova značajka isključena (0).<br>Ako je uključena (1) tada se omogućuje usmjeravanje odnosno prosljeđivanje mrežnih paketa između mrežnih sučelja na računalu (prema **RFC 1122** i **RFC 1812**). |

**Slika 180.1. Primjeri upotrebe Proxy ARP-a.**




Na slici 180.1. (lijevi dio slike) pogledajmo primjer rada ***Proxy ARP*** poslužitelja u virtualizaciji, kada ne koristimo standardni način rada *hipervizora* koji radi u **Bridge** već u **Routing** načinu rada odnosno u načinu rada kao usmjerivač. U ovakvom načinu rada on mora sav mrežni promet sa svoje unutarnje mreže [*Subnet B*], od i sa virtualnih računala, preusmjeriti preko svog vanjskog mrežnog sučelja, prema vanjskoj mreži [*Subnet A*].

Na istoj slici (desni dio slike) vidimo drugačiju primjenu *proxy ARP* poslužitelja koji odvaja dva segmenta mreže:

- **Subnet A**: 192.168.1.0/24
- **Subnet B**: 10.10.10.0/24

Kod ovakve primjene, kada primjerice računalo iz mreže A [*Subnet A*]: **PC-1** (192.168.1.10) želi poslati mrežni paket prema računalu **PC-3** (10.10.10.10) koje se nalazi u drugoj mreži [*Subnet B*], ono prvo mora saznati njegovu MAC adresu. Stoga ono mora poslati ARP upit na svoju cijelu mrežu [*Subnet A*] slanjem ARP upita na *broadcast* adresu mreže (to je standardna metoda ARP upita). Pošto je to *broadcast* poruka svi na ovom segmentu mreže primaj ovaj paket, pa tako i ***Linux Proxy ARP*** poslužitelj. S obzirom da ***Linux Proxy ARP*** poslužitelj zna da se traženo računalo **PC-3** (IP 10.10.10.10) nalazi na njegovoj donjoj mreži [*Subnet B*], on odgovara s ARP odgovorom i daje svoju MAC adresu u ARP odgovoru. To znači da on odgovara da je IP adresi 10.10.10.10 pripadajuća MAC adresa, MAC adresa njegovog (*Linux Proxy ARP*) mrežnog sučelja na mreži A [*Subnet A*]. Kasnije, kada krene IP promet od računala **PC-1**, prema **PC-3**, ***Linux Proxy ARP*** poslužitelj (sada kao usmjerivač) će ih proslijediti prema odredišnom računalu **PC-3** (na IP adresu10.10.10.10).
Pri tome je računalo **PC-1**, naučilo da je IP adresi 10.10.10.10 pripadajuća MAC adresa od ***Linux Proxy ARP*** poslužitelja, pa tako i gradi svoju ARP tablicu.

Koje su prednosti i mane upotrebe *Proxy ARP poslužitelja*:

- Pomaže računalima na (pod)mreži da dosegnu udaljene podmreže bez potrebe za konfiguriranjem usmjeravanja ili zadanog pristupnika (engl. *Default gateway*).
- Omogućuje Proxy ARP poslužitelju (što može biti i usmjerivač ili napredni preklopnik) da odgovori na ARP upite, nudeći svoju vlastitu MAC adresu te preusmjeravajući paket prema odredišnoj IP adresi.
- Povećava se ARP promet na mreži.
- Potrebne su veće ARP tablice (veće zauzeće memorije).
- Sigurnost može biti narušena jer bilo koje računalo na mreži može tvrditi da je neko drugo, kako bi presreo pakete, što se naziva "*spoofing*".

Pogledajmo inicijalnu konfiguraciju potrebnu za Proxy ARP značajku uz aktivirano prosljeđivanje paketa između mrežnih sučelja:

```
sysctl net.ipv4.conf.all.proxy_arp=1
sysctl net.ipv4.ip_forward=1
```

Postoje i druge *sysctl* varijable s kojima se mogu optimirati mehanizmi sustava vezani za ARP protokol.
Sve ih možemo vidjeti sa sljedećom naredbom:

```
sysctl -a | grep net.ipv4.conf. | grep arp
```

Navedene *sysctl* varijable s kojima se mogu optimirati mehanizmi sustava vezani za ARP protokol su:

| Sysctl varijabla | Opis |
|---|---|
| `net.ipv4.conf.all.arp_accept` | Definira ponašanje za ***gratuitous* ARP** mrežne okvire čija IP adresa već nije prisutna u ARP tablici:<br>0 - ne stvara nove unose u ARP tablici.<br>1 - kreira nove unose u ARP tablici.<br>I odgovori i zahtjevi vrste ***gratuitous* ARP** pokrenut će ažuriranje ARP tablice ako je ova postavka uključena.<br>Za **gratuitous ARP** pogledajte poglavlje: **20.7.1.3. Scenarij za prisilno osvježavanje ARP tablice (*Gratuitous ARP*) detaljnije.** |
| `net.ipv4.conf.all.arp_announce` | Definira različite razine ograničenja za najavu lokalne izvorišne IP adrese iz IP paketa u ARP zahtjevima poslanim na mrežnom sučelju:<br>0 - (zadano) koristi bilo koju lokalnu adresu, konfiguriranu na bilo kojem sučelju.<br>1 - Pokušava izbjeći lokalne adrese koje nisu u ciljnoj podmreži za ovo sučelje. Ovaj način rada je koristan kada su ciljna računala dostupna preko ovog sučelja, zahtijeva da izvorna IP adresa u ARP zahtjevima bude dio njihove logičke mreže konfigurirane na mrežnom sučelju za primanje paketa. |
| `net.ipv4.conf.all.arp_evict_nocarrier` | Kada je postavljeno na 1 (zadano za nove kernele 5.x ), ARP predmemorija bit će izbrisana ako dođe do *NOCARRIER* događaja. |
| `net.ipv4.conf.all.arp_filter` | 0 - (zadano) Kernel može odgovoriti na ARP zahtjeve s adresama s drugih mrežnih sučelja. Ovo se može činiti pogrešnim, ali obično ima smisla jer povećava šanse za uspješnu komunikaciju. S obzirom da IP adrese pripadaju cijelom računalu na Linuxu, a ne samo određenim mrežnim sučeljima. Ovo ponašanje uzrokuje probleme samo kod složenijih postavki poput balansiranja opterećenja (engl. load balancing).<br>1 - Omogućuje vam da imate više mrežnih sučelja na istoj podmreži i da ARP zahtjevi za svako mrežno sučelje dobiju odgovore na temelju toga hoće li kernel usmjeriti paket s ARP zahtjeva, IP adresa van tog mrežnog sučelja (stoga morate koristiti *source based routing* da ovo radi). Drugim riječima, omogućuje kontrolu kod koje će mrežna sučelja (obično jedno) odgovoriti na ARP zahtjev. |

| Sysctl varijabla | Opis |
|---|---|
| `net.ipv4.conf.all.arp_ignore` | Definira različite načine za slanje odgovora kao odgovor na primljene ARP zahtjeve koji rješavaju lokalne ciljne IP adrese:<br>0 - (zadano): odgovara za bilo koju lokalnu ciljnu (odredišnu) IP adresu, konfiguriranu na bilo kojem sučelju.<br>1 - odgovara samo ako je ciljna IP adresa konfigurirana lokalna adresa na dolaznom sučelju.<br>2 - odgovara samo ako je ciljana IP adresa lokalna adresa konfigurirana na dolaznom sučelju i obje s pošiljateljevom IP adresom su dio iste *podmreže* na ovom sučelju.<br>3 - ne odgovara za lokalne adrese konfigurirane s opsega adresa računala, samo rezolucije za globalne adrese i adrese veza su odgovorene.<br>4-7 - rezervirano.<br>8 - ne odgovaraj za sve lokalne adrese. |
| `net.ipv4.conf.all.arp_notify` | Definira način obavijesti o promjenama adrese i uređaja.<br>0 - (zadano): ne radi ništa.<br>1 - Generira ***gratuitous ARP*** zahtjeve kada se uređaj pokrene ili se promijeni hardverska (MAC) adresa.<br>Za **gratuitous ARP** pogledajte poglavlje:<br>**20.7.1.3. Scenarij za prisilno osvježavanje ARP tablice (*Gratuitous ARP*) detaljnije.** |
| `net.ipv4.conf.all.drop_gratuitous_arp` | (1) Odbacuje/ispušta sve ***gratuitous ARP*** mrežne okvire, na primjer ako postoji poznati *ARP proxy* poslužitelj na mreži i takvi se okviri ne moraju koristiti (ili u slučaju 802.11, ne smiju se koristiti za sprječavanje napada.)<br><br>Zadano: isključeno (0) odnosno ne odbacuju se *gratuitos* ARP mrežni okviri.<br>Za **gratuitous ARP** pogledajte poglavlje:<br>**20.7.1.3. Scenarij za prisilno osvježavanje ARP tablice (*Gratuitous ARP*) detaljnije.** |
| `net.ipv4.conf.all.proxy_arp` | Uključuje *proxy ARP* značajku. Proxy ARP za sučelje bit će omogućen ako je barem jedna od postavki *conf/{all,interface}/proxy_arp* postavljena na TRUE, u suprotnom će biti onemogućen.<br>Standardno je isključen Proxy ARP (0).<br><br>Proxy ARP se može koristiti i u virtualizaciji, ako se ne koristi ***Bridge*** već ***Routing*** model povezivanja virtualnih računala na mrežno sučelja hipervizora.<br><br>Pogledajte poglavlje o virtualizaciji:<br>**27.1. Virtualizacija.** |
| `net.ipv4.conf.all.proxy_arp_pvlan` | Omogućuje proxy ARP za privatne VLANove.<br>U osnovi dopušta proxy ARP odgovore natrag na isto sučelje (s kojeg je primljen ARP zahtjev/poziv).<br>Ova tehnologija poznata je pod različitim nazivima:<br>• Prema **RFC 3069** naziva se **VLAN Aggregation**.<br>• *Cisco* i *Allied Telesyn* ju zovu **Private VLAN**.<br>• *Hewlett-Packard* ju naziva **Source-Port filtering** ili **port-isolation**.<br>• *Ericsson* ju naziva **MAC-Forced Forwarding**. |

**Izvori informacija: (1439),(1440),(1441),(K-8)**

# 21. IP adrese i usmjeravanje

Nakon što smo nešto naučili o osnovama mreža upoznajmo se s njima malo detaljnije. U daljnjem tekstu pričat ćemo samo o ***IPv4*** protokolu i IP adresama. Što je uopće IP adresa?. IP adresa je jedinstvena adresa koja identificira uređaj na internetu ili lokalnoj mreži. IP označava "internetski protokol", koji čini skup pravila koja definiraju oblik (format) podataka poslanih putem interneta ili lokalne mreže. IP adrese su identifikatori koji omogućuje slanje informacija između uređaja na mreži.
IP adresu možemo promatrati kao našu jedinstvenu poštansku adresu koju čine: država, grad i poštanski kôd [pr. 31000 Osijek], ulica i kućni broj, koji su u cijelom svijetu jedinstveni i identificiraju: odredišnu (ako primamo poštu) ili izvorišnu (ako šaljemo poštu) adresu. Isto tako i IP adresa jedinstveno identificira pošiljatelja i primatelja mrežnih paketa. IP adresa se na najnižoj razini koristi u binarnom formatu, ali se zbog lakšeg prikaza koristi dekadski prikaz. Kako uopće izgleda jedna IP (IPv4) adresa u dekadskom i binarnom sustavu?. Uzmimo za primjer IP adresu: **172.17.100.18** koja binarno izgleda ovako: **10101100.00010001.01100100.00010010.**

Pogledajmo navedenu IP adresu u tablici, kako bi nam sva ova pretvorba bila malo jasnija:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 172 | 17 | 100 | 18 |
| **Binarno** | 10101100 | 00010001 | 01100100 | 00010010 |
| **Binarni broj bitova** | 8 | 8 | 8 | 8 |

Ovdje je vidljivo i to, kako je IP adresa podijeljena u četiri segmenta od 8 bitova odvojenih s točkom ( . ).

## 21.1. Klase IP adresa

Kada se IP adresa dodjeljuje računalu, neki od bitova s lijeve strane predstavljaju sâmu mrežu. Broj bitova koji će prezentirati mrežu ovisi o klasi mreže. Naime IP adrese su podijeljene u točno definirane klase, od kojih svaka ima svoju (preporučenu) namjenu. Pogledajmo tablicu s klasama mreža i njihovim namjenama.

| Klasa IP adresa | Opseg (prvi oktet) | Namjena |
|---|---|---|
| **Klasa A** | **0.0.0.0.** …do… **127.255.255.255** | Za ekstremno velike mreže |
| | **127.** | Rezervirano za *loopback* (lokalno računalo i sl.) |
| **Klasa B** | **128.0.0.0** …do… **191.255.255.255** | Za srednje do velikih mreža |
| **Klasa C** | **192.0.0.0** …do… **223.255.255.255** | Manje mreže |
| **Klasa D** | **224.0.0.0** …do… **239.255.255.255** | *Multicast* adrese |
| **Klasa E** | **240.0.0.0** …do… **255.255.255.255** | Rezervirana od IETF za istraživanje |

Primjerice sve IP adrese koje počinju od 128.0.0.0 pa sve do 191.255.255.255 su adrese koje su unutar klase **B**. Osim navedenih klasa u novije vrijeme zatraženo je da se unutar klasa **A**, **B** i **C** rezerviraju određeni dijelovi mreža za privatne lokalne mreže (engl. *Private Networks*) kako je definirano u standardu **RFC1918**.

Pogledajmo i ove opsege IP adresa koje pripadaju privatnim (lokalnim) mrežama, prema **RFC1918**, **RFC 6598** i drugima.

| Klasa adresa | Opseg | Maska mreže (*Netmask*) | Max broj IP adresa |
|---|---|---|---|
| **A** | 10.0.0.0 …do… 10.255.255.255 | 255.0.0.0 | 16.777.216 |
| **CGN/CGNAT/LSN** | 100.64.0.0…do…10.127.255.254 | 255.192.0.0 | 4.194.304 |
| **B** | 172.16.0.0 …do… 172.31.255.255 | 255.240.0.0 | 1.048.576 |
| **C** | 192.168.0.0 …do… 192.168.255.255 | 255.255.0.0 | 65.536 |

IP adrese unutar ovih okvira se smatraju privatnim IP adresama i standardno, za usmjerivače telekoma tj. *ISPova* (Engl. *Internet service provider*) nisu *rutabilne* odnosno javno dostupne jer ih usmjerivači ne propuštaju izvana. To znači kako primjerice na kućnom usmjerivaču, kako bi izašli na "internet" morate koristiti neku od tehnika skrivanja odnosno preslikavanja (translacije) unutarnjih privatnih adresa (pr. iz ovog opsega) na vanjsku javnu IP adresu (pr. iz klase **A**,**B** ili **C**).



Vezano za translaciju adresa (**NAT**) pogledajte poglavlje:
**23.4.4. Translacija adresa (*NAT*).**

*Carrier-grade NAT* (***CGN*** ili ***CGNAT***), također poznat kao NAT velikih razmjera (***LSN***), vrsta je prevođenja mrežnih adresa (**NAT**) u IPv4 mrežama. S CGNAT-om, krajnje točke (uređaji), posebno kućne/privatne mreže, konfigurirane su s privatnim mrežnim adresama koje se potom prevode u javne IPv4 adrese pomoću mehanizma za prevođenje mrežnih adresa (**NAT**) ugrađenih u mrežu mrežnog operatera. Na taj način se omogućuje dijeljenje malih i ograničenih skupova javnih adresa među mnogim krajnjim točkama (uređajima) koje se nalaze unutar skupa privatnih adresa.

**Izvori informacija: RFC 1518, RFC 1519, RFC 1918, RFC 5735, RFC 6598.**

# 21.2. Mreže

Korištenjem kombinacije IP adrese i maske mreže (Engl. *Netmask)* definiran je svaki mrežni element na mreži: računalo, poslužitelj, laptop, tablet, mobitel, mrežni uređaji, … Zbog toga i svaka mreža ima svoju:

- **IP adresu** mreže i pripadajuću ***masku mreže** (Netmask adresu)***.**
- **Opseg (*range*) IP adresa** za upotrebu.
- ***Broadcast* IP adresu.**

To ćemo objasniti na primjeru. Uzmimo jednu mrežu iz **B** klase. Unutar velike mreže **B** klase odabrat ćemo dvije mreže:

- Mreža A: `198.150.11.*` koja se označava sa `198.150.11.0`
- Mreža B: `198.150.12.*` koja se označava sa `198.150.12.0`

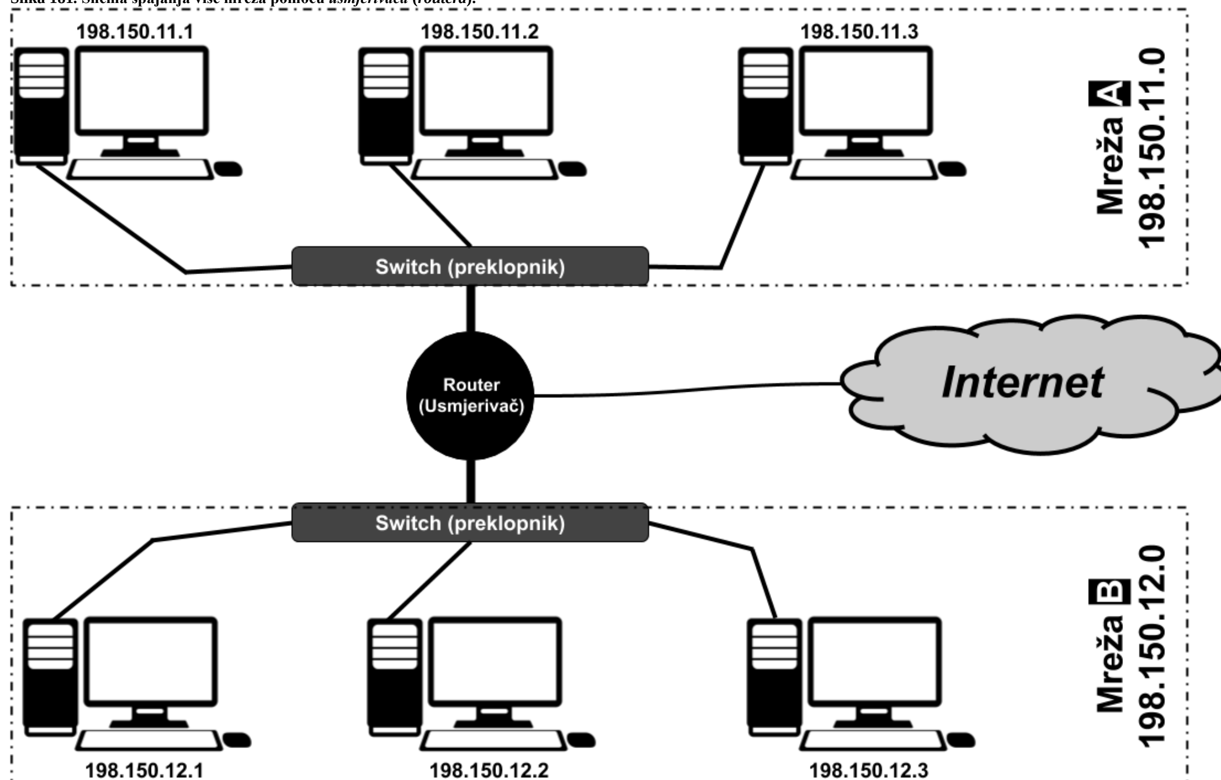
Pogledajmo što te mreže sadrže:

| **Adresa mreže** (*Network Address*) | **Opseg adresa za upotrebu** | **Broadcast adresa za (ovu) mrežu** |
|---|---|---|
| **198.150.11.0** | **198.150.11.1** …do… **198.150.11.254** | **198.150.11.255** |
| **198.150.12.0** | **198.150.12.1** …do… **198.150.12.254** | **198.150.12.255** |

Dakle sâma mreža koju smo nazvali: A ima IP adresu 198.150.11.0.
Unutar te mreže svim računalima, poslužiteljima i ostalim uređajima ćemo dodijeliti neku IP adresu.

Upotrebljive IP adrese iz ovog opsega IP adresa su od: 198.150.11.1 sve do 198.150.11.254. Na kraju našeg opsega IP adresa se nalazi *broadcast* IP adresa ove mreže koja je rezervirana, pa nju ne možemo koristiti.
U ovom konkretnom slučaju to je *broadcast* adresa: 198.150.11.255. Isto pravilo vrijedi i za mrežu koju smo nazvali: B, kao i sve druge mreže. Zamislit ćemo naše dvije navedene mreže s računalima, spojene na sljedeći način, vidljiv na slici 181.

**Slika 181. Shema spajanja više mreža pomoću *usmjerivača* (*routera*).**




Mreža A je na jednoj strani, a mreža B na drugoj te su spojene preko usmjerivača preko kojeg obje mreže imaju izlaz na Internet i preko kojeg su i međusobno povezane. U slučaju kada komunikacija teče sâmo između računala u mreži A isto kao i u slučajevima kada komunikacija teče samo između računala u mreži B, računala normalno komuniciraju preko TCP/IP skupa protokola, i to direktno bez usmjerivača kao posrednika. Pri tome prvo računalo (pr. 198.150.11.1) komunicira s drugim direktno (pr. 198.150.11.2) i obratno, pošto uz polazišnu i odredišnu MAC adresu (*Source* i *Destination* MAC) znaju svoju IP adresu i IP adresu računala s kojim komuniciraju. Međutim kada je potrebno da računala iz mreže A komuniciraju s računalima iz mreže B, neke osnovne stvari moraju biti zadovoljene:

- Naravno uz svaku ***IP*** adresu računala, mora biti definirana i maska mreže odnosno ***Netmask*** (o tome u kasnije).
- Mora biti definiran i podrazumijevani usmjerivač (Engl. ***Default Gateway)*** tj. uređaj koji spaja našu mrežu s drugim mrežama, što je u ovom slučaju naš usmjerivač odnosno ***ROUTER*** na slici.

Što se dalje događa, vidjet ćemo kada naučimo još nekoliko osnovnih pojmova.

# 21.3. Maska mreža (*Netmask*)

Korištenjem kombinacije ***IP*** adrese i maske mreže (*Netmask* adrese) definiran je svaki mrežni element: računalo, poslužitelj, laptop, tablet. mobitel, usmjerivač, preklopnik,…
Dakle svaka IP adresa dolazi u pâru s takozvanom *netmask* adresom koja se naziva i maskom mreže.
To znači da svaka mreža ima svoj:

- **Opseg IP adresa** (Engl. *IP range*) za upotrebu.
- **IP adresu mreže** (Engl. *Network IP*).
- **Masku mreže (***Engl. Netmask).*
- **Broadcast IP adresu** (Engl. *Broadcast IP*).

Maska mreže je 32 bitni broj ($2^{32}$) koji uz pripadajuću **IP** adresu identificira svaki uređaj na mreži, ali i sâmu mrežu.
Maska mreže i njena pripadajuća IP adresa govore nam na koji način će IP adresa biti interpretirana odnosno kojoj mreži određena IP adresa pripada. Zbog potrebe podjela mreža na *podmreže* (engl. *Subnet*) i nadmreže (engl. *Supernets*) te sâmih klasa mreža, uz IP adresu se ***mora*** koristiti i maska mreže.

Pogledajmo standardne maske mreža, za različite klase mreža.

| Klasa mreže | Standardna maska mreže (*netmask*) |
|---|---|
| Klasa **A** | 255.0.0.0 |
| Klasa **B** | 255.255.0.0 |
| Klasa **C** | 255.255.255.0 |

Vezano za masku mreže, na slici 182 je vidljiva podjela na četiri okteta od kojih svaki ima 8 bitova (slično kao IP adrese).
Bitovi s lijeve strane (plavo) označavaju mrežu (*Network*), a s desne strane identificiraju računalo (*Host*) na mreži.

Maska mreže (***netmask***) se logički sastoji od dva dijela, kako je vidljivo na slici 182.

**Slika 182. Pogled na klase mreža.**

| Klasa A | Network | Host | | |
|---|---|---|---|---|
| Binarno - Oktet - | 1 | 2 | 3 | 4 |

| Klasa B | Network | | Host | |
|---|---|---|---|---|
| Binarno - Oktet - | 1 | 2 | 3 | 4 |

| Klasa C | Network | | | Host |
|---|---|---|---|---|
| Binarno - Oktet - | 1 | 2 | 3 | 4 |

| Klasa D | Host | | | |
|---|---|---|---|---|
| Binarno - Oktet - | 1 | 2 | 3 | 4 |

Unutar maske mreže imamo:

- **Mrežni** dio (***Network*** dio) - lijevi dio (plavo).
- **Dio za računala** (***Host*** dio) - desni dio.

U normalnoj upotrebi, maska mreže se često određuje tako da prvo moramo saznati kojoj klasi pripada naša IP adresa. U slučaju korištenja podmreža i nadmreža, potrebno je izračunati koliko podmreža želimo dobiti od jedne mreže, te koliko velike podmreže uopće želimo. Dalje u tekstu, govorit ćemo o "standardnim" mrežama koje nisu podijeljene u podmreže, odnosno imat će standardnu masku mreže, prema njihovoj klasi mreže iz koje dolaze.

**Mrežni dio maske mreže**
Ovaj dio odnosno prvi dio maske mreže s lijeve strane, opisuje mrežu. Popunimo sve jedinice s lijeve strane, dok ne dođemo do naše mreže. Primjerice standardna maska mreže za klasu ***C*** je 255.255.255.0 i pri tome prva tri okteta čine sve jedinice odnosno ukupno 24 bita za ovaj primjer. Kod drugog čestog načina označavanja maske mreže se prebrojavaju bitovi (1) od lijevo na desno pa se gore navedena maska označava i kao **/24**.

Tako primjerice za našu IP adresa 198.150.11.1 možemo reći sljedeće:

- Ona je iz klase **C** odnosno pripada opsegu IP adresa od 192.0.0.0 do 223.255.255.255.
- Prema tablici gore, za **C** klasu mreže vidimo kako ***host*** dio u četvrtom oktetu čine nule; gledano s desno na lijevo.
- Njena maska mreže će prema tome biti: 255.255.255.0. Pogledajte mrežni dio u tablici gore.

Naime popunili smo sve jedinice binarno od lijeva na desno, i to prva tri okteta (8 bitova) pa dobivamo: **11111111.11111111.11111111**.00000000 i dobili smo dekadsku masku mreže: 255.255.255.0.

Mrežni dio maske mreže se popunjava jedinicama od lijevo prema desno i to:

- Prema definiciji klasa mreža ili
- Prema potrebama naše mreže ili
- Prema potrebama pod mreže (*subnet*) ili nad mreže (*supernet*) **sâmo, ako ih koristimo.**

Pogledajmo to i ovako; mrežni dio maske mreže se kreće:

- Od klase **A** mreže; od 255.0.0.0: binarno: **11111111**.00000000.00000000.00000000
  - Dakle prve pod mreže kreću od: 11111111.**1**0000000.00000000.00000000
  - Te ih možemo povećavati, dodavanjem **1** od lijevo prema desno, pa bi sljedeća pod mreža bila: 11111111.**11**000000.00000000.00000000 i tako dalje sve dok ne dođemo do sljedeće klase mreže.
- Potom dolazimo do klase **B** koja kreće od 255.255.0.0 binarno: **11111111.11111111**.00000000.00000000
  - Koja se proširuje na isti način s jedinicama.
- Do klase **C** mreže - od 255.255.255.0: binarno: **11111111.11111111.11111111**.00000000.
- I dalje do realno iskoristive maske: 255.255.255.248 binarno: **11111111.11111111.11111111.11111**000.

**Host dio maske mreže**

Nule na kraju, odnosno prebrojavanjem **0** od desno na lijevo, sve dok ne dođemo do jedinica, daje nam naš ***Host*** dio odnosno dio za upotrebu (računala/*host*) unutar kojega možemo koristiti IP adrese. Dakle koliko bitova s nulama imamo u maski mreže, od desno na lijevo, toliko bitova za upotrebu imamo na istoj poziciji gledajući na našu IP adresu mreže.
S navedenom maskom mreže za naš slučaj vidimo kako je cijeli zadnji oktet (8 puta po 0 tj. $2^8$=256) slobodan za računala, što čini 256 adresa (0 do 255 uključujući nulu).
Dakle : 11111111.11111111.11111111.**00000000** → sada gledamo ove nule s desne strane za *host* dio.

**Ne zaboravimo kako u svakoj mreži postoje i dvije dodatne IP adrese:**

- Prva moguća (dostupna) IP adresa u konkretnoj mreži je rezervirana za mrežu, a to je IP adresa (ove) mreže.
- Zadnja moguća (dostupna) IP adresa u konkretnoj mreži je rezervirana za *Broadcast* IP adresu.

Nama zbog navedenog, u svakoj mreži ostaje ukupan broj IP adresa umanjen za ove dvije IP adrese.
U našem slučaju su to: 256 - 2 = 254 upotrebljive adrese.

## Kako se računa adresa mreže odnosno kako usmjerivači i uređaji koji rade na OSI sloju tri (OSI 3) znaju kojoj mreži pripada koja IP adresa?

Na sâmom početku, kada naše računalo dobije IP adresu i pripadajuću mu masku mreže, ono radi izračune navedene dolje, kako bi izračunalo kojoj mreži ono pripada (prema IP adresi mreže) te koja je *broadcast* adresa te mreže. Ovo je važno u normalnoj komunikaciji i unutar sâme ***LAN*** mreže, bez izlaska mrežnih paketa vân, preko usmjerivača. Naime svaki element u mrežni (računalo, poslužitelj, usmjerivač, preklopnik, mrežni štampač i sl.) prihvaća samo one mrežne pakete koji su namijenjeni njegovoj mreži.

Ovo je dodatno važno i stoga što neki mrežni protokoli u komunikaciji na IP sloju koriste *broadcast* metodu komunikacije.
U ovakvoj komunikaciji odredišna je IP adresa na koju se šalje paket upravo *broadcast* IP adresa mreže, a koju moraju moći primiti sva računala unutar određene IP mreže, koja znaju odnosno moraju znati da je to *broadcast* IP adresa njihove mreže.
Kada mrežni paket dođe do usmjerivača, on za svaki paket koji mu dolazi na pojedino mrežno sučelje provjerava kojoj ***IP adresi mreže*** on pripada. Pošto usmjerivač ima svoju tablicu usmjeravanja u kojoj se nalaze i parovi: IP adresa mreže i njen pâr, masku mreže, te mrežno sučelje kojem oni pripadaju, on na osnovi IP adrese paketa koji mu je došao jasno zna gdje će svaki taj paket i usmjeriti. Obratite pažnju na to kako IP adresa mreže nije IP adresa računala.

**Kako se izračunava IP adresa mreže?**

Podsjetimo se prvo logičke operacije **I** (Engl. ***AND***) kako je vidljivo u tablici (ovisno u ulazima **A** i **B**):

| Ulaz **A** | Ulaz **B** | Rezultat |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Uzmimo IP adresu računala: 198.150.11.1

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 198 | 150 | 11 | 1 |
| **Binarno** | 11000110 | 10010110 | 00001011 | 00000001 |

Te pogledajmo njenu pripadajuću masku mreže (*Netmask)* adresu:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 255 | 255 | 255 | 0 |
| **Binarno** | 11111111 | 11111111 | 11111111 | 00000000 |

Sada se radi logička **I** operacija (Engl. ***AND***) s IP adresom računala i njenom pripadajućom maskom mreže:

| | | | | |
|---|---|---|---|---|
| **IP adresa** | 11000110 | 10010110 | 00001011 | 00000001 |
| **Netmask** | 11111111 | 11111111 | 11111111 | 00000000 |
| **Rezultat AND operacije - binarno** | 11000110 | 10010110 | 00001011 | 00000000 |
| **Rezultat AND operacije - dekadski** | 198 | 150 | 11 | 0 |

Dobili smo kao rezultat: **198.150.11.0**. Dakle ovo je IP adresa mreže za računalo s IP adresom: 198.150.11.1 koje ima masku mreže: 255.255.255.0. Mreže koja nije podijeljena na pod mreže (*subnetirana)*. Rekli smo i kako je IP adresa mreže nulta odnosno prva IP adresa koju ne smijemo i ne možemo koristiti te kako nakon nje slijedi nîz upotrebljivih IP adresa, te kako je zadnja IP adresa unutar te mreže *broadcast* IP adresa koju isto ne smijemo koristiti za računala na mreži. **Broadcast** adresa mreže se izračunava pomoću *inverzne* maske mreže; naše maske mreže te IP drese mreže, ali o tome malo kasnije.

**Pogledajmo jedan primjer.**
Uzmimo jednu mrežu **C** klase. Unutar velike mreže **C** klase odabrat ćemo dvije manje mreže, s time da obje imaju standardnu masku mreže (***netmask)*** *255.255.255.0* za mrežu klase **C**:

- Mreža **1**: `198.150.11.*` koja se označava sa: `198.150.11.0`
- Mreža **2**: `198.150.12.*` koja se označava sa: `198.150.12.0`

Pogledajmo što te mreže sadrže:

| **Adresa mreže** (*Network Address*) | **Opseg adresa za upotrebu** | **Broadcast adresa za (ovu) mrežu** |
|---|---|---|
| **198.150.11.0** | **198.150.11.1** …do… **198.150.11.254** | **198.150.11.255** |
| **198.150.12.0** | **198.150.12.1** …do… **198.150.12.254** | **198.150.12.255** |

Dakle mreža **1** ima IP adresu 198.150.11.0 (ova IP adresa označava cijelu ovu mrežu).
Unutar te mreže svim računalima, poslužiteljima i ostalim uređajima kojima ćemo dodijeliti neku IP adresu, IP adresu možemo dodijeliti iz opsega IP adresa od: 198.150.11.1 sve do 198.150.11.254, a na kraju našeg opsega IP adresa se nalazi ***broadcast*** IP adresa (koja je rezervirana i nju ne možemo koristiti jer je to broadcast IP adresa). U ovom slučaju to je adresa: 198.150.11.255 (prethodno smo ju izračunali). Isto pravilo vrijedi i za mrežu **2**, ali i druge mreže s istom mrežnom maskom.

Provjerimo u tablici dolje (**183**) za masku mreže: `255.255.255.0`. Dakle maska mreže*:* `255.255.255.0` ima `24` jedinice s lijeve strane (*11111111.11111111.11111111.00000000*) pa je njena ***CIDR*** oznaka `/24`. Takozvana ***CIDR*** notacija koristi se za skraćeno označavanje maske mreže (*netmaska)*, a označava se tako da prebrojimo jedinice s lijeve strane maske mreže.

Pogledajmo sve maske mreže (*netmask*-e) za sve klase mreža, kako je vidljivo na slici **183**.



Za detalje oko **IP** protokola, pogledajte poglavlje:
**23. Internet (IP) sloj (OSI 3).**

Slika 183. Prikaz svih klasa mreža sa pripadajućim maskama mreža.

| Bitmask (CIDR) | Maska mreže (Netmask) | Maska mreže binarno | Ukupan broj upotrebljivih IP adresa |
|---|---|---|---|
| /32 | 255.255.255.255 | 11111111.11111111.11111111.11111111 | 1 |
| /31 | 255.255.255.254 | 11111111.11111111.11111111.11111110 | **2*** |
| /30 | 255.255.255.252 | 11111111.11111111.11111111.11111100 | 2 |
| /29 | 255.255.255.248 | 11111111.11111111.11111111.11111000 | 6 |
| /28 | 255.255.255.240 | 11111111.11111111.11111111.11110000 | 14 |
| /27 | 255.255.255.224 | 11111111.11111111.11111111.11100000 | 30 |
| /26 | 255.255.255.192 | 11111111.11111111.11111111.11000000 | 62 |
| /25 | 255.255.255.128 | 11111111.11111111.11111111.10000000 | 126 |
| /24 | 255.255.255.0 | 11111111.11111111.11111111.00000000 | 254 **← C klasa mreža** |
| /23 | 255.255.254.0 | 11111111.11111111.11111110.00000000 | 510 |
| /22 | 255.255.252.0 | 11111111.11111111.11111100.00000000 | 1.022 |
| /21 | 255.255.248.0 | 11111111.11111111.11111000.00000000 | 2.046 |
| /20 | 255.255.240.0 | 11111111.11111111.11110000.00000000 | 4.094 |
| /19 | 255.255.224.0 | 11111111.11111111.11100000.00000000 | 8.190 |
| /18 | 255.255.192.0 | 11111111.11111111.11000000.00000000 | 16.382 |
| /17 | 255.255.128.0 | 11111111.11111111.10000000.00000000 | 32.766 |
| /16 | 255.255.0.0 | 11111111.11111111.00000000.00000000 | 65.534 **← B klasa mreža** |
| /15 | 255.254.0.0 | 11111111.11111110.00000000.00000000 | 131.070 |
| /14 | 255.252.0.0 | 11111111.11111100.00000000.00000000 | 262.142 |
| /13 | 255.248.0.0 | 11111111.11111000.00000000.00000000 | 524.286 |
| /12 | 255.240.0.0 | 11111111.11110000.00000000.00000000 | 1.048.574 |
| /11 | 255.224.0.0 | 11111111.11100000.00000000.00000000 | 2.097.150 |
| /10 | 255.192.0.0 | 11111111.11000000.00000000.00000000 | 4.194.302 |
| /9 | 255.128.0.0 | 11111111.10000000.00000000.00000000 | 8.388.606 |
| /8 | 255.0.0.0 | 11111111.00000000.00000000.00000000 | 16.777.214 **← A klasa mreža** |
| /7 | 254.0.0.0 | 11111110.00000000.00000000.00000000 | 33.554.430 |
| /6 | 252.0.0.0 | 11111100.00000000.00000000.00000000 | 67.108.862 |
| /5 | 248.0.0.0 | 11111000.00000000.00000000.00000000 | 134.217.726 |
| /4 | 240.0.0.0 | 11111000.00000000.00000000.00000000 | 268.435.454 |
| /3 | 224.0.0.0 | 11110000.00000000.00000000.00000000 | 536.870.910 |
| /2 | 192.0.0.0 | 11100000.00000000.00000000.00000000 | 1.073.741.822 |
| /1 | 128.0.0.0 | 11000000.00000000.00000000.00000000 | 2.147.483.646 |
| /0 | 0.0.0.0 | 10000000.00000000.00000000.00000000 | 4.294.967.294 (Cijeli IP prostor) |

*** /31** je poseban slučaj detaljno opisan u **RFC 3021** gdje mreže s ovom vrstom maske mreže mogu dodijeliti dvije IP adrese za tzv. point-to-point veze.

Za različite klase mreža pogledajmo standardne maske mreža, te opsege adresa unutar njih.

| **Klasa mreže** | **Opseg IP adresa** | **Opseg privatnih adresa** | **Standardna maska mreže (*netmask*)** |
|---|---|---|---|
| Klasa **A** | 1.0.0.0 do 127.0.0.0 | 10.0.0.0 do 10.255.255.255 | 255.0.0.0 |
| Klasa **B** | 128.0.0.0 do 191.255.0.0 | 172.16.0.0 do 172.31.255.255<br>100.64.0.0 do 100.127.255.255 (**ISP CGN**) | 255.255.0.0 |
| Klasa **C** | 192.0.0.0 do 223.255.255.0 | 192.168.0.0 do 192.168.255.255 | 255.255.255.0 |
| Klasa **D** *(multicast)* | 224.0.0.0 do 239.255.255.255 | - | - |
| **Klasa E** | 240.0.0.0 do 255.255.255.255 | - | |

**Izvori informacija, za poglavlja 21. – 21.3.: (988),(989),(K-6), (K-10), (K-11), RFC 791, RFC 3021, RFC 6598.**

## 21.3.1. Podmreže (*Subnets*)

Pôd mreže (engl. *Subnets*) se koriste za segmentiranje IP adresa (unutar klasa **A**, **B** i **C**) na razini pojedine mreže, na manje dijelove. *Subnetiranjem* razbijamo određenu mrežu na pod mreže koje se u potpunosti ponašaju kao obične mreže, dakle imaju:

- IP adresu mreže (i masku mreže).
- Opseg IP adresa za upotrebu.
- *Broadcast* IP adresu.

*Subnetirati* mrežu na pod mreže je moguće uz pomoć *subnet maske* odnosno posebne maske mreže. *Subnetirana* IP adresa se sastoji od mrežnog dijela te *subnet* i *host* dijela IP adrese. Pošto se kod svake TCP/IP konekcije uz IP adresu provjerava i maska mreže, svako računalo (mrežna oprema) na osnovu maske mreže prepoznaje je li riječ o *subnetiranju* ili nije, odnosno kojoj mreži pripada određeni par: *IP adresa + maska mreže (Netmask)*.

**Kako prepoznati je li mreža podijeljena u pod mreže (subnetirana)?**
U slučajevima kada se držimo maske mreže definirane prema klasama mreža za određenu IP adresu, tada mreža nije *subnetirana*. Podsjetimo se koje su standardne vrijednosti maske mreže prema klasama mreža za mreže koje nisu *subnetirane*:

| Klasa mreže | Standardna maska mreže (Netmask) |
|---|---|
| **A** | 255.0.0.0 |
| **B** | 255.255.0.0 |
| **C** | 255.255.255.0 |

**Kako mrežni uređaji znaju kojoj mreži pripada određena IP adresa i je li ta mreža *subnetirana*?**
Za svaki par: ***IP adresa + maska mreže*** se prvo radi logička **AND** operacija (hrv. logička operacija **I**) te se dobiva ***IP adresa mreže*** kojoj pripada ta IP adresa. Na osnovi IP adrese mreže jasno je gdje će se svaki mrežni paket koji ima tu *IP/Netmask* kombinaciju i proslijediti od strane uređaja koji radi na OSI sloju tri odnosno od usmjerivača ili *Multilayer* preklopnika.

Uzmimo sljedeću IP adresu:

| **Dekadski** | 172 | 17 | 100 | 18 |
|---|---|---|---|---|
| **Binarno** | 10101100 | 00010001 | 01100100 | 00010010 |

Te njenu pripadajuću masku mreže (*Netmask* adresu):

| **Dekadski** | 255 | 255 | 255 | 0 |
|---|---|---|---|---|
| **Binarno** | 11111111 | 11111111 | 11111111 | 00000000 |

Na osnovi IP adrese se radi logička **AND** operacija s njenom maskom mreže:

| **IP adresa** | 10101100 | 00010001 | 01100100 | 00010010 |
|---|---|---|---|---|
| **Netmask** | 11111111 | 11111111 | 11111111 | 00000000 |
| **Rezultat AND operacije - binarno** | 10101100 | 00010001 | 01100100 | 00000000 |
| **Rezultat AND operacije - dekadski** | 172 | 17 | 100 | 0 |

Dobili smo kao rezultat: **172.17.100.0**. Dakle ovo je IP adresa mreže koja nije *subnetirana* jer pripada klasi **C** sa standardnom maskom mreže. Podsjetimo se da je ova IP adresa ujedno i nulta IP adresa unutar ove mreže. Rekli smo kako je nulta IP adresa mreže, odnosno prva IP adresa koju ne smijemo i ne možemo koristiti te da nakon nje slijedi nîz upotrebljivih IP adresa.
Važno za znati je kako se tijekom prolaska mrežnih paketa kroz mrežne uređaje za svaki par *IP/Netmask* unutar svakog mrežnog paketa, u pozadini prvo odrađuje logička **AND** operacija jer se jedino na osnovi rezultata logičke **AND** operacije utvrđuje gdje određeni mrežni paket treba biti proslijeđen. Ovaj posao odrađuju uređaji na OSI sloju tri (usmjerivači ili Multilayer preklopnici). Dakle oni poznaju mrežnu topologiju te znaju gdje im je spojena koja mreža, te kada saznaju na koju mrežu (prema izračunatoj IP adresi mreže) poslati paket, u tom smjeru ga i prosljeđuju. Sada kada znamo IP adresu mreže te nam još nedostaje i zadnja IP adresa unutar ove mreže ili *subneta,* koja predstavlja *broadcast* adresu te mreže.

**Kako saznati broadcast adresu konkretne mreže?**
Ovaj izračun se radi u dva koraka.

**1.Izračun inverzne maske**
Prvo se radi logička operacija ***XOR*** s maskom mreže koji ima sve jedinice (dekadski: 255.255.255.255) i s našom maskom mreže, što je u našem slučaju: 255.255.255.0. Prvo se podsjetimo **XOR** logičke operacije (ovisno o ulazima **A** i **B**):

| Ulaz **A** | Ulaz **B** | Rezultat |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Krenimo u izračun za naš slučaj:

| | | | | |
|---|---|---|---|---|
| ***Maska mreže (Netmask)* sa svim jedinicama** | 11111111 | 11111111 | 11111111 | 11111111 |
| **Naša maska mreže (*Netmask*)** | 11111111 | 11111111 | 11111111 | 00000000 |
| **Rezultat XOR operacije** | 00000000 | 00000000 | 00000000 | 11111111 |

Rezultat je dekadski: **0.0.0.255**

**2.Konačni rezultat**

Kako bismo dobili zadnju IP adresu unutar naše mreže, a to je *Broadcast* IP adresa naše mreže, radimo logičku operaciju **`OR`** (hrv. **`ILI`**) od našeg rezultata **XOR** operacije (0.0.0.255) i naše IP adrese mreže, koju smo također izračunali (172.17.100.0): Podsjetimo se logičke operacije **`ILI`** (Engl. ***`OR`***) operacije ovisno o ulazima **A** i **B**:

| Ulaz **A** | Ulaz **B** | Rezultat |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Sada izračunajmo što je potrebno:

| | | | | |
|---|---|---|---|---|
| **Rezultat XOR operacije** | 00000000 | 00000000 | 00000000 | 11111111 |
| **IP adresa naše mreže - binarno** | 10101100 | 00010001 | 01100100 | 00000000 |
| **Rezultat OR operacije - Broadcast IP adresa** | 10101100 | 00010001 | 01100100 | 11111111 |
| **Dekadski rezultat** | 172 | 17 | 100 | 255 |

Dakle dekadski rezultat je: 172.17.100.255 što znači kako je za našu konkretnu mrežu: 172.17.100.0 *Broadcast* IP adresa: 172.17.100.255, što je i zadnja IP adresa u toj mreži. Sada imamo sve podatke za mrežu koji su nam potrebni:

- Naša IP adresa: 172.17.100.10 s pripadajućom maskom mreže: 255.255.255.0, nalazi se u mreži: 172.17.100.0.
- Naša *broadcast* IP adresa i sâmim time zadnja IP adresa u mreži je: 172.17.100.255.
- Stoga su upotrebljive IP adrese u ovoj našoj mreži od: 172.17.100.1 do 172.17.100.254.

Kako bismo odlučili kako *subnetirati* mrežu, ako to želimo, od strane administratora mreže potrebno je procijeniti:

- Koliko pod mreža (*subneta*) želimo unutar pojedine mreže.
- Koliko računala (*hostova*) odnosno slobodnih IP adresa želimo imati unutar svake pod mreže (*subneta*).

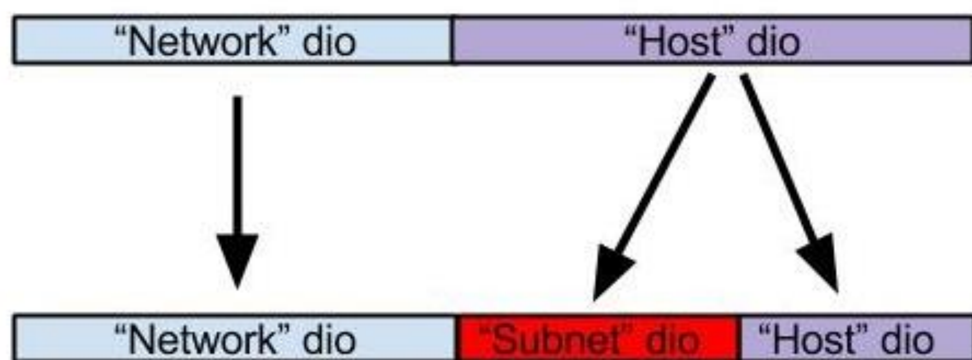
Potom se izračunava pôdmreža (*subnet)* koji nam odgovara. Kada smo odredili *subnet*, potrebno je za svaki *subnet* zapisati:

- Adresu mreže svakog pojedinog *subneta.*
- *Broadcast* adresu svakog pojedinog *subneta*.
- Opseg iskoristivih IP adresa unutar svakog pojedinog *subneta.*

**Kako se radi *subnetiranje* odnosno pôdmreže?**

Kada smo pričali o maski mreže odnosno *netmasku* rekli smo da postoji dio koji je rezerviran za mrežu i dio za računala (*Hostove*). Kod upotrebe standardnih maski mreže tu je kraj priče, ali kod *subnetiranja* mi od *Host* dijela moramo posuditi lijevi dio za proširenje postojeće maske mreže (*netmaska*) tako da dobivamo sljedeće stanje, kako je vidljivo na slici **184.**

**Slika 184. Pogled na mrežni dio i dio rezerviran za računala (host) dio.**




Mi zapravo od **„*Host*"** dijela posuđujemo bitove s lijeve strane i popunjavamo ih isto s lijeva na desno, kao što popunjavamo kod maske mreže (*netmaska)* s time da se dodani bitovi nadodaju na postojeću masku mreže.

Primjerice za masku mreže (*netmask*): 255.255.255.0 to izgleda ovako:

| | | | | |
|---|---|---|---|---|
| *Netmask* dekadski | 255 | 255 | 255 | 0 |
| *Netmask* binarno | `11111111` | `11111111` | `11111111` | `00000000` |

U ovom primjeru onaj desni oktet koji ima sve jedinice (`00000000`) je ***`Host`*** dio. Taj dio moramo podijeliti tako da dodavanjem svakog bita s lijevo na desno praktično dijelimo mrežu na dva (2) dijela, a ostatak prema desno je upotrebljiv za upotrebu (opseg IP adresa unutar *subneta*).
Ako želimo podijeliti ovu mrežu na dva dijela dodati ćemo jedan bit s lijeve strane, pa ćemo dobiti sljedeću masku mreže.

Pogledajte taj bit (**`1`**) koji smo dodali u zadnji oktet i ostale nule koje su slobodne za upotrebu:

| | | | | |
|---|---|---|---|---|
| ***Netmask* dekadski** | 255 | 255 | 255 | 128 |
| ***Netmask* binarno** | `11111111` | `11111111` | `11111111` | **`1`**`0000000` |

Pogledajmo tablicu za pôdmreže za klasu **`A`** mreža:

| **Netmask** | **CIDR notacija** | **Broj mogućih podmreža (*subneta*)** | **Broj iskoristivih IP adresa (Hostova) unutar *subneta*** | **Ukupan (z)broj svih upotrebljivih IP adresa** |
|---|---|---|---|---|
| 255.0.0.0 | /8 | 1 | 16777214 | 16777214 |
| 255.128.0.0 | /9 | 2 | 8388606 | 16777212 |
| 255.192.0.0 | /10 | 4 | 4194302 | 16777208 |
| 255.224.0.0 | /11 | 8 | 2097150 | 16777200 |
| 255.240.0.0 | /12 | 16 | 1048574 | 16777184 |
| 255.248.0.0 | /13 | 32 | 524286 | 16777152 |
| 255.252.0.0 | /14 | 64 | 262142 | 16777088 |
| 255.254.0.0 | /15 | 128 | 131070 | 16776960 |

Pogledajmo i tablicu za sve moguće pôdmreže za klasu **`B`** mreža:

| **Netmask** | **CIDR notacija** | **Broj mogućih podmreža (subneta)** | **Broj iskoristivih IP adresa (Hostova) unutar subneta** | **Ukupan (z)broj svih upotrebljivih IP adresa** |
|---|---|---|---|---|
| 255.255.0.0 | /16 | 1 | 65534 | 65534 |
| 255.255.128.0 | /17 | 2 | 32766 | 65532 |
| 255.255.192.0 | /18 | 4 | 16382 | 65528 |
| 255.255.224.0 | /19 | 8 | 8190 | 65520 |
| 255.255.240.0 | /20 | 16 | 4094 | 65504 |
| 255.255.248.0 | /21 | 32 | 2046 | 65472 |
| 255.255.252.0 | /22 | 64 | 1022 | 65408 |
| 255.255.254.0 | /23 | 128 | 510 | 65280 |

I na kraju slijedi tablica za sve moguće pôdmreže za klasu **`C`** mreža:

| **Netmask** | **CIDR notacija** | **Broj mogućih podmreža (subneta)** | **Broj iskoristivih IP adresa (Hostova) unutar subneta** | **Ukupan (z)broj svih upotrebljivih IP adresa** |
|---|---|---|---|---|
| 255.255.255.0 | /24 | 1 | 254 | 254 |
| 255.255.255.128 | /25 | 2 | 126 | 252 |
| 255.255.255.192 | /26 | 4 | 62 | 248 |
| 255.255.255.224 | /27 | 8 | 30 | 240 |
| 255.255.255.240 | /28 | 16 | 14 | 224 |
| 255.255.255.248 | /29 | 32 | 6 | 192 |
| 255.255.255.252 | /30 | 64 | 2 | 128 |
| 255.255.255.254 | /31 | 128 | 2 - samo za *point-to-point* mreže | 256 - samo za *point-to-point* mreže |

*Na osnovi naših potreba te korištenjem ovih tablica možemo odabrati pravilnu masku mreže. Uzmimo sljedeći primjer: imamo mrežu 172.17.100.0 s maskom mreže: 255.255.255.0. Tu mrežu želimo podijeliti na dvije (2) mreže unutar kojih nam treba biti maksimalno 126 korisnih (iskoristivih) IP adresa.*

Pogledajmo našu navedenu mrežu:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 172 | 17 | 100 | 0 |
| **Binarno** | 10101100 | 00010001 | 01100100 | 00000000 |

Te njenu pripadajući masku mreže:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 255 | 255 | 255 | 0 |
| **Binarno** | 11111111 | 11111111 | 11111111 | 00000000 |

Korištenjem tablice zaključili smo da, ako želimo dva subneta, od kojih svaki može imati do 126 korisnih adresa, maska mreže mora biti: **255.255.255.128** tj. prema ***CIDR*** notaciji /25 što znači kako imamo 25 jedinica u *subnetu* (s lijeva na desno).

To izgleda ovako:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 255 | 255 | 255 | 128 |
| **Binarno** | 11111111 | 11111111 | 11111111 | **1**0000000 |

Dakle posudili smo jedan bit iz ***host*** dijela da bi ga iskoristili za *subnetiranje*. Ovo je najbrža metoda izračuna *subneta*:
S jednim posuđenim bitom imamo dvije moguće mreže: prva mreža ako je bit u 0-li i druga kada je taj bit u 1-ici.
Bit ima vrijednost 0:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 172 | 17 | 100 | 0 |
| **Binarno** | 10101100 | 00010001 | 01100100 | **0**0000000 |

Bit ima vrijednost 1 :

| | | | | |
|---|---|---|---|---|
| **Binarno** | 10101100 | 00010001 | 01100100 | **1**0000000 |
| **Dekadski** | 172 | 17 | 100 | 128 |

Ovdje smo dobili dvije mreže (kako smo i odabrali/planirali):
**Prva (1)** mreža: 172.17.100.0 s maskom mreže 255.255.255.128.
**Druga (2)** mreža: 172.17.100.128 s maskom mreže 255.255.255.128.

**Što to znači za prvu (1) mrežu** :

- IP adresa ove mreže je: 172.17.100.0.
- Mreža se proteže sve do 172.17.100.127, s time da je adresa 127 rezervirana za *Broadcast*.
- Prema tome *Broadcast* IP adresa je: 172.17.100.127.
- Upotrebljive IP adrese su od: 172.17.100.1 do 172.17.100.126.

**Što to znači za drugu (2) mrežu**

- IP adresa ove mreže je: 172.17.100.128.
- Mreža se proteže sve do 172.17.100.255, s time da je adresa 255 rezervirana za *Broadcast*.
- Prema tome *Broadcast* IP adresa je: 172.17.100.255.
- Upotrebljive IP adrese su od: 172.17.100.129 do 172.17.100.254.

**Izvori informacija: (988),(989), (K-6), (K-10), (K-11), RFC 917, RFC 1918.**

### 21.3.2. Nadmreže (*Supernetting*)

Nad mreža (*Supernetting*) je po logici suprotnost pôdmrežama odnosno *subnettingu*. U slučajevima kada od više malih mreža želimo napraviti jednu veliku mrežu koristimo *supernetting*. *Supernetting* je opisan u dva dokumenta: RFC1338 i RFC1519. Određivanje nad mreža radi se u suprotnom smjeru od pôdmreža (*subnets*). Pogledamo li način na koji smo pomicali tj. posuđivali bitove u maski mreže (*netmask*) kod pôdmreža smo to radili od lijeva na desno, u slučaju nadmreža se zapravo vraćamo unazad tj. od desno na lijevo. Ova metoda se često koristi kod sumarizacija *ruta* često zvana i *agregacija rute*.

**O čemu se radi?**
Za svaku mrežu koju imamo, naš mrežni uređaj koji radi na OSI sloju tri (obično usmjerivač) mora imati *rutu* do nje.
Svaka nova *ruta* zauzima memoriju i oduzima određeno vrijeme za procesiranje (tzv. *CPU load*) od usmjerivača.
Ako primjerice imamo sljedeće mreže koje smo *subnetirali* prije:

- **Mreža (1)**: 172.17.100.0 s maskom mreže 255.255.255.128.
- **Mreža (2)**: 172.17.100.128 s maskom mreže 255.255.255.128.

Koje za izlaz na internet i sa interneta dolaze s drugog mrežnog sučelja na usmjerivaču (*routeru*):

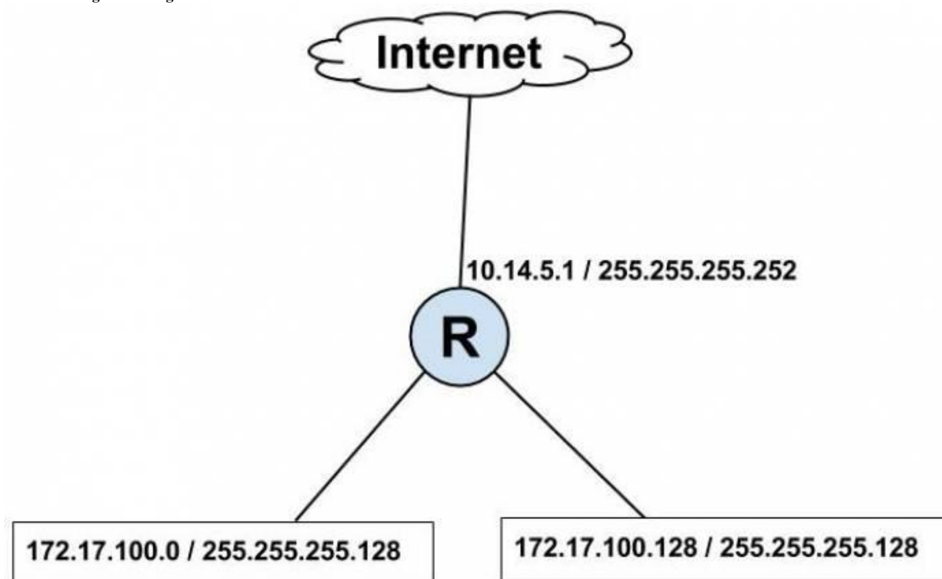
- 10.14.5.1 / 255.255.255.252.

Te želimo smanjiti broj zasebnih *ruta*, jer bi ih u ovom slučaju morali imati dvije: (1) i (2). Napravit ćemo nadmrežu koja uključuje sve postojeće pôdmreže odnosno u konačnici će objediniti ove dvije *rute* u sâmo jednu koja će ih sve sažeti.
Naša maska mreže sada izgleda ovako, u slučaju gornjeg *subnetiranja*:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 255 | 255 | 255 | 128 |
| **Binarno** | 11111111 | 11111111 | 11111111 | **10000000** |

Pogledajmo sliku ove mreže (slika 185):

**Slika 185. Pogleda na logičku shemu navedene mreže.**




Kao primjer smo dali mali broj mreža.
Međutim ako imamo sustave s jako velikim brojem mreža, pojavit će se potreba da ubrzamo proces usmjeravanja takozvanom *sumarizacijom ruta*, pa ćemo stoga i napraviti nadmreže.

U našem slučaju možemo napraviti sljedeće:
Ako sada koristimo sljedeću masku mreže, objedinit ćemo obje mreže:

| | | | | |
|---|---|---|---|---|
| **Dekadski** | 255 | 255 | 255 | 0 |
| **Binarno** | 11111111 | 11111111 | 11111111 | **00000000** |

S ovime smo smanjili tablicu usmjeravanja odnosno *ruting* tablicu na pola.

**Što smo napravili?**
Promijenili smo masku mreže koja sada govori da cijela nad mreža: 172.17.100.0 / 24 što znači IP adrese 172.17.100.0 do 172.17.100.255 ima jednu *rutu* za ulaz i izlaz na internet, tako da sve mreže koje dolaze s interneta znaju da ovaj usmjerivač iza sebe ima jednu veću mrežu 172.17.100.0/24 bez potrebe da znaju da smo ju mi iznutra razbili na dvije manje mreže odnosno dvije pôdmreže (*subneta)*. Na taj način smo i sebi i susjednim usmjerivačima smanjili tablice usmjeravanja, o kojima ćemo ubrzo govoriti.

**Izvori informacija: (988),(989), (K-6), (K-10), (K-11).**

# 21.4. Usmjeravanje (*Routing*)

Usmjeravanje (engl. *Routing*) je proces odabira najboljeg puta za prolaz mrežnih paketa prema odredištu.
Uređaji koji odrađuju usmjeravanje se zovu usmjerivači (engl. *Routers*). Za potrebe usmjeravanja možemo koristiti: **statičke rute** ili **protokole za usmjeravanje** (*routing* protokole) odnosno dinamičke rute.

Statičke *rute* se dodaju statički odnosno ručno, za svaku novu mrežu. To postaje malo nezgodno kada imamo više mreža ili kada se mreže često mijenjaju. U slučajevima kada imamo veliki broj mreža, statičke *rute* je teško održavati ručnim i stalnim dodavanjem i brisanjem. Stoga se za sve imalo veće mreže koriste protokoli za usmjeravanje koji dinamički, automatski dodaju i brišu odnosno mijenjaju *rute*. Usmjerivači na internetu obično koriste neki od protokola za usmjeravanje. Statički se dodaje samo tzv. podrazumijevana ruta (***Default route***) koja je osnovna *ruta* koja prolazi preko primarnog usmjerivača koji se zove i *podrazumijevani usmjerivač ili pristupnik (Engl.* ***Default gateway****).*
U manjim mrežama to je jedini usmjerivač na mreži. Drugi slučaj upotrebe statičkih ruta je, ako imamo prilično statične (nepromjenjive) ili vrlo male mreže u kojima nam je praktično ručno dodavati, mijenjati ili brisati *rute*. *Podrazumijevana (Default) ruta* s *podrazumijevanim* usmjerivačem obično ima oblik:

| IP Adresa | Maska mreže | Podrazumijevani usmjerivač (*Default Gateway*) IP |
|---|---|---|
| `0.0.0.0` | `0.0.0.0` | `192.168.1.1` |

***Podrazumijevana (Default) ruta*** koju vidite znači: *za sve* IP adrese (`0.0.0.0`) odnosno sve mreže za koje ne postoji izričito definirana *ruta*. U slučaju kada usmjerivač ili računalo na mreži ne zna gdje poslati neki paket odnosno kada nema definiranu *rutu* (putanju) za njega, paket se šalje na IP adresu ***Default Gateway*** usmjerivača, a obično je to glavni usmjerivač prema internetu. Rute pod Linuxom možemo vidjeti s naredbom `netstat -rn` ili s `ip route`, a navedena *podrazumijevana ruta* izgleda ovako: `0.0.0.0 192.168.1.1 0.0.0.0 UG 0 0 0 eth0` ili ako ju ispisujemo s naredbom `ip route`: `default via 192.168.1.1 dev eth0`.

Stoga pogledajmo podrazumijevanu rutu s dvjema navedenim naredbama:

```
netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG        0 0          0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

U ovom slučaju je jasno da podrazumijevana ruta (`Destination 0.0.0.0`) pokazuje na podrazumijevani usmjerivač `192.168.1.1` te da se do njega dolazi preko mrežnog sučelja: `eth0`.

Odnosno s naredbom `ip route` bi to izgledalo ovako:

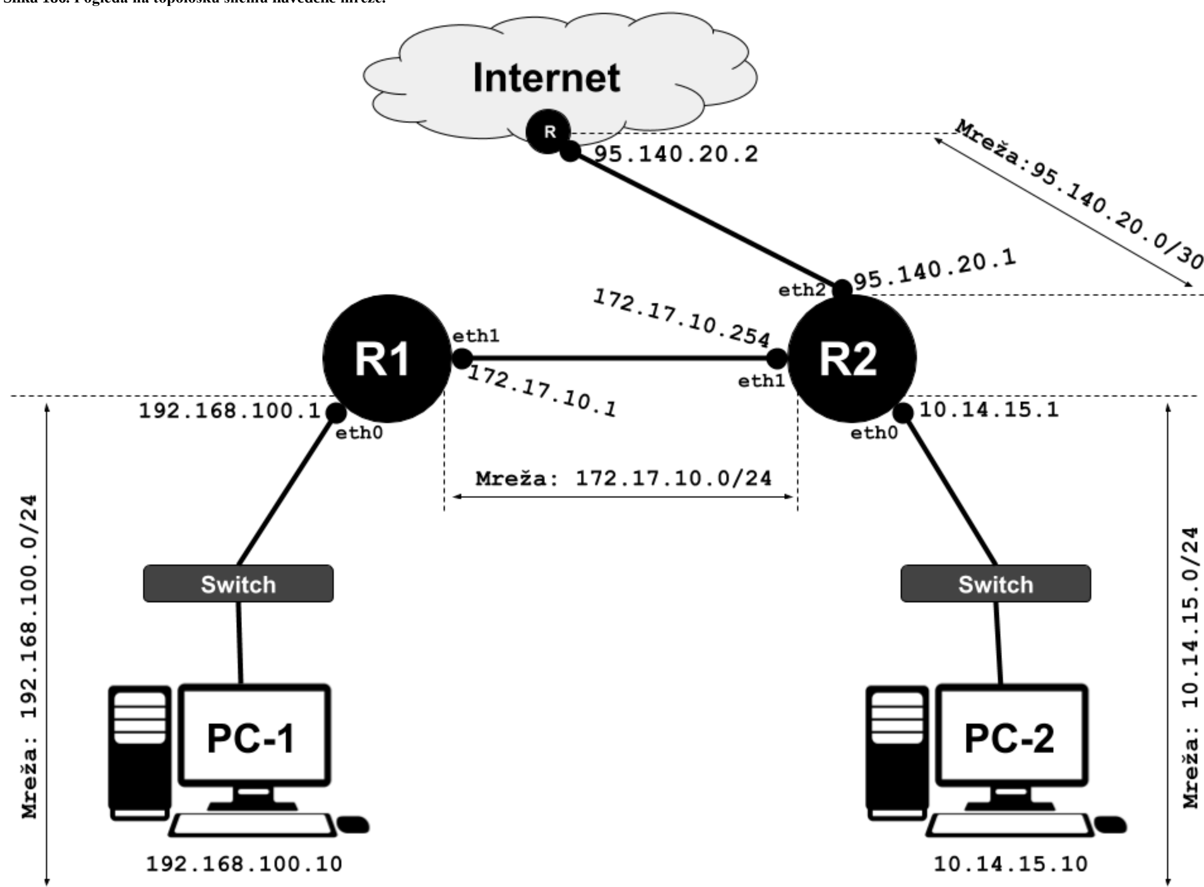
```
ip route
default via 192.168.1.1 dev eth0 proto dhcp metric 100
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.129 metric 100
```

**Što su uopće tablice usmjeravanja odnosno *rute*?**
*Rute* nam pokazuju tko je od usmjerivača odgovoran za koju mrežu, tako da naše računalo (pr. *Linux*) zna kome odnosno gdje poslati koji mrežni paket, koji izlazi iz naše mreže prema drugoj mreži. Možemo promatrati *rutu* kao putokaz za mrežne pakete, poput autoceste: treba li skrenuti lijevom ili desnom cestom, prema odredištu.
Primjer tablice usmjeravanja na usmjerivaču (**R1**) na slici **186** možemo opisno promatrati na sljedeći način. Za mrežu ***192.168.100.0 / 255.255.255.0 (/24)*** koristiti mrežno sučelje ***eth0***. Za mrežu ***10.14.15.0 / 255.255.255.0 (/24)*** koristiti usmjerivač na IP adresi ***172.17.10.254.*** Kao podrazumijevana *ruta* (***mreža 0.0.0.0 / 0.0.0.0)*** koristiti IP adresu usmjerivača ***R2: 172.17.10.254.*** U konkretnom primjeru to znači kako će primjerice komunikacija unutar mreže **192.168.100.0** biti bez posredovanja usmjerivača **R1** jer je računalima unutar te mreže to lokalna mreža, dostupna i usmjerivaču preko njegovog mrežnog sučelja **eth0** s IP adresom 192.168.100.1. Računalo **PC-1** ima postavljeno kao podrazumijevani usmjerivač IP adresu svog usmjerivača: **192.168.100.1**, do kojega može doći preko svoje lokalne mreže.
U slučajevima kada **PC-1** šalje paket na mrežu: 10.14.15.0, prema odredištu: 10.14.15.10 on ga može poslati sâmo na usmjerivač **R1**: **192.168.100.1** koji će paket potom poslati na svoje sučelje ***eth1*** prema usmjerivaču **R2** odnosno prema IP adresi:**172.17.10.254** jer mu je on podrazumijevani usmjerivač. Usmjerivač **R2** će primiti taj paket na sučelju **eth1** te ga proslijediti na svoje sučelje **eth0** koje ima IP adresu 10.14.15.1 i poslati ga dolje prema računalu **PC-2**. Računalo **PC-2** mora imati postavljen kao podrazumijevani usmjerivač IP adresu svog usmjerivača **R2**: 10.14.15.1 kako bi mogao odgovoriti, ali i komunicirati prema internetu (i drugim mrežama).
Usmjerivač **R2** mora imati i rutu za mrežu 192.168.100.0/24 prema usmjerivaču **R1** kako bi znao vratiti pakete nazad**.** Usmjerivač **R2** mora imati i podrazumijevanu rutu koja pokazuje na IP adresu 95.140.20.2 kako bi cijela ova lokalna mreža (***LAN***) mogla imati vezu prema internetu koja prolazi preko njega.



*Svako mrežno sučelje usmjerivača mora pripadati drugoj mreži kako bi usmjeravanje uopće radilo. Svi međusobno povezani usmjerivači moraju poznavati sve mreže drugih usmjerivača kako bi komunikacija među njima mogla funkcionirati. Odnosno u vrlo jednostavnim mrežama koristiti sâmo svoju podrazumijevanu rutu za dolazak do drugih mreža (ako je to izvedivo).*

Slika 186. Pogleda na topološku shemu navedene mreže.




Protokoli za usmjeravanje prema logici rada mogu koristiti dvije vrste algoritama.
Prema tome protokole za usmjeravanje dijelimo prema vrsti algoritma koji koristi, a to su:

- *Distance vector* protokoli, poput: **RIP**, **RIP v.2**, **IGRP**, …
- *Link-state* protokoli, poput: **OSPF**, **IS-IS**, …
- Hibridni protokoli koji su kombinacija gornja dva, poput: **BGP**, **EIGRP** i drugih.

**Malo detaljnije o osnovnom usmjeravanju**
U slučaju usmjeravanja koje se događa na **OSI** sloju tri (IP adrese prema TCP/IP modelu), uređaji odluku za usmjeravanje paketa donose na osnovu IP adresa. Tablica na osnovu koje se odrađuje usmjeravanje se zove *Ruting tablica* odnosno ***tablica usmjeravanja***. Dakle slična priča kao za preklopnike, ali na višem OSI sloju (OSI 3) te malo kompleksnija zbog protokola usmjeravanja i mogućnosti dinamičkih promjena tablica usmjeravanja pomoću protokola za usmjeravanje. Uređaji koji se bave usmjeravanjem zovemo usmjerivači, ali postoje i preklopnici koji mogu odrađivati i taj dio posla. Ovakve preklopnike zovemo **Multilayer preklopnici** odnosno preklopnici koji rade na više OSI slojeva (OSI 2, 3 i 4). Ovakvi preklopnici vrlo su važni u današnjim mrežama jer nam daju mogućnost da uredno segmentiramo lokalnu mrežu, a da to ne unese usporavanja koja bi uveli klasični usmjerivači. Tijekom procesa usmjeravanja *usmjerivač* ili *Multilayer* preklopnik, zaprimaju svaki pojedini mrežni paket te gledaju sâmo OSI sloj tri odnosno sloj na kojemu je prema TCP/IP modelu IP protokol. IP protokol na svom sloju između ostalih polja sadrži polja u kojima se nalaze:

- Izvorišna IP adresa (Engl. *Source IP*) te odredišna IP adresa (Engl. *Destination IP*).
- Provjerni zbroj (Engl. *Checksum*) te ukupna duljina paketa (što obuhvaća IP zaglavlja i podataka).
- ***TTL*** (Engl. *Time to Live*) odnosno vrijeme dozvoljenog života paketa na mreži, kao i druge opcije i zastavice.

Usmjerivač za svaki primljeni mrežni paket, pristigao na bilo kojem njegovom mrežnom sučelju (*portu*) prvo provjerava *IP* zaglavlje i to dio s izvorišnom i odredišnom IP adresom. On potom uspoređuje odredišnu IP adresu sa svojom tablicom usmjeravanja. Na osnovi ove tablice usmjerivač usmjerava paket preko onog mrežnog sučelja preko kojega će paket moći doći do svog odredišta: ili preko drugih usmjerivača ili direktno, sve ovisno o topologiji mreže.



Protokoli za usmjeravanje mogu osigurati visoku dostupnost sustava. Za detalje o njoj, pogledajte poglavlje: **26.8.4. Visoko dostupni sustavi (*High Availability*).**

**Izvori informacija:** `man 7 ip`,`man 7 tcp`,`man 7 udp`,`man ip route`.

### 21.4.1. Distance vector protokoli

Protokoli za usmjeravanje koji koriste vektor udaljenosti (engl. ***distance vector)*** za usmjeravanje, jednostavni su za konfiguriranje i održavanje. *Distance vector* protokoli troše i manje resursa. Princip njihovog rada je u tome da svaki usmjerivač korištenjem *distance vector* protokola obavještava susjedne usmjerivače o promjenama, i to periodički. ***Link-State*** protokoli za usmjeravanje s druge strane moraju obavještavati sve usmjerivače unutar topologije o promjenama u mrežama.
*Distance-vector* protokoli su bazirani na izračunavanju smjera (engl. *Direction*) i udaljenosti (engl. *Distance*) do odredišne mreže. Pod smjerom se smatra IP adresa sljedećeg usmjerivača (engl. *Next Hop*) na koji se šalje paket, uz podatak o tome koje je izlazno mrežno sučelje našeg usmjerivača. Pod udaljenosti se smatra "*cijena*" (engl. *Cost*) do odredišta. Kako se određuje takozvana „*cijena*" ovisi o sâmom protokolu tj. takozvanoj metrici. U konačnicu, udaljenost je mjera *cijene* dolaska do određenog mrežnog čvora. Najjeftinija ruta između bilo koja dva čvora je ruta s minimalnom udaljenosti.
Međutim kako bi uspostavili najbolju rutu, usmjerivači redovito razmjenjuju informacije sa susjednim usmjerivačima, obično njihovu tablicu usmjeravanja, broj skokova do odredišne mreže i eventualno druge informacije vezane za promet. Usmjerivači koji implementiraju protokol vektora udaljenosti oslanjaju se isključivo na informacije koje im pružaju drugi usmjerivači i ne procjenjuju topologiju mreže.
Navedeni protokoli ažuriraju tablice usmjeravanja usmjerivača i određuju rutu na koju će paket biti poslan sljedećim skokom koji predstavlja izlazno mrežno sučelje usmjerivača i IP adresa mrežnog sučelja usmjerivača primatelja.
Ažuriranja se povremeno izvode tako da se cijela ili dio tablice usmjeravanja usmjerivača šalje svim njegovim susjedima koji su konfigurirani da koriste isti protokol usmjeravanja s vektorom udaljenosti. Nakon što usmjerivač posjeduje ove informacije, može izmijeniti vlastitu tablicu usmjeravanja kako bi odražavao promjene, a zatim obavijestiti svoje susjede o promjenama. Ovaj proces je opisan kao 'usmjeravanje po glasinama' jer se usmjerivači oslanjaju na informacije koje primaju od drugih usmjerivača i ne mogu utvrditi jesu li informacije stvarno valjane i istinite. Postoji niz značajki koje se mogu koristiti za pomoć kod nestabilnosti i netočnih informacija o usmjeravanju.
Što se tiče konkretne implementacije, primjerice ***RIP*** protokol koristi tzv. *hop count* do odredišta odnosno prebrojavanje usmjerivača kroz koje sve paket mora proći da dođe do odredišta. S druge strane ***IGRP*** protokol koristi kašnjenje (engl. *Delay*) i dostupnu propusnost (*Bandwidth)* do tog odredišta. Na osnovu svih navedenih parametara usmjerivač gradi tzv. vektore minimalnih udaljenosti do svake mreže, u tablicu koja se zove *Routing* tablica odnosno tablica usmjeravanja. Procedura kada usmjerivači razmjenjuju *rute* i osvježavaju svoje tablice usmjeravanja (*ruting* tablice) se zove konvergencija. Konvergencija *Distance-vector* protokola je veća u odnosu na *Link state* protokole; dakle sporiji su.
Primjeri *Distance vector* protokola za usmjeravanje su:

- ***RIP*** - *Routing Information Protocol* (v.1 i v2.).
- ***IGRP*** - *Interior Gateway Routing Protocol*.
- ***EIGRP*** - *Enhanced IGRP (iako je on hibrid između **distance vector** i **link-state)*** te neki drugi protokoli.
- ***BGP*** - *Border Gateway Protocol (iako je on tzv. **path vector** protokol).*

---

### 21.4.2. Link-state protokoli

***Link-state*** protokoli za usmjeravanje rade tako da svaki usmjerivač kreira kartu veza svih mreža u kojoj je vidljivo kako su usmjerivači međusobno spojeni odnosno kreiraju si topologiju mreže. Tada si svaki usmjerivač izračunava najbolje logičke puteve za sva mrežna odredišta prema kojima se, korištenjem određenih algoritama kreira tablica usmjeravanja (*routing* tablica). To je očito potpuno drugačiji način rada u odnosu na protokole usmjeravanja koji koriste vektor udaljenosti (*distance vector*), koji rade tako da svaki čvor dijeli svoju tablicu usmjeravanja sa svojim susjedima. Naime *link-state* protokolima stanje veza jedina je informacija koja se prenosi između čvorova (usmjerivača).
U svakom slučaju, ovaj proces je prilično zahtjevniji (CPU i RAM) od *distance vector* protokola, ali je puno praktičniji za veće mreže jer smanjuje konvergenciju uslijed promjena u mrežama.

Protokoli za usmjeravanje iz ove skupine načelno rade na sljedeći način:
1. Pronalaze se susjedni usmjerivači pomoću protokola dostupnosti, koji radi periodično i odvojeno za svaki od svojih izravno povezanih susjeda (usmjerivača).
2. Između susjeda se razmjenjuju informacije, periodički, ali i u slučajevima kada dolazi do nekih promjena u mreži. Periodičko slanje poruka se zove ***link-state advertisement***, a služi tome da se identificiraju: čvorovi (usmjerivači) koji šalju poruke, ali i svi drugi direktno spojeni usmjerivači. Pri tome svaka poslana poruka sadrži serijski broj, koji se sa svakom novom porukom povećava, kako bi se moglo identificirati je li došlo do nove inačice (promjene) poruke.
3. Kreira se topologija mreže, a na osnovi topologije mreže se kreira (preračunava) tablica usmjeravanja.

Ono što dodatno čini ove protokole boljima za veće mreže je:

- Mogućnost slanja podataka sâmo, ako je došlo do promjena na nekoj od mreža.
- Uzimanja cijele topologije mreže kao bitne za odluke tijekom usmjeravanja (*routinga*).

Neki od protokola koji pripadaju u ovu kategoriju su:

- ***OSPF*** - *Open Shortest Path First.*
- ***IS-IS*** - *Intermediate System to Intermediate System.*

---

**Izvori informacija za poglavlja 21.4.1 i 21.4.2: (695),(696), (K-6), (K-10), (K-11).**

### 21.4.3. Upotreba protokola za umjeravanje pod Linuxom

Potreba za razvojem projekta koji bi objedinio mnoge protokole za usmjeravanjem pod Linuxom, izvorno je došla od *Kunihira Ishigura*, koji je u suradnji s *Yoshinarijem Yoshikawom* pokrenuo takozvani ***Zebra*** projekt (1995.g.) kao projekt otvorenog kôda, ali su vrlo brzo osnovali tvrtku ***IP Infusion*** unutar koje su razvili svoj ***ZebOS***. On uz protokole za usmjeravanje podržava preko 200 mrežnih protokola, a objedinio je i mnoge druge napredne značajke. Navedeni ***ZebOS*** unutar svojih rješenja koriste mnoge druge velike tvrtke poput: *Fortinet* (unutar *FortiOS*-a), *Citrix* NetScaler, *F5 Networks* i drugih. Kako je zajednica i dalje trebala ovakvo rješenje, a razvoj ***Zebra*** projekta se praktično zaustavio 2005.g., nastao je novi projekt imena ***Quagga***, koji je tada osim na Linux, prenesen i na druge operativne sustave poput: *Solaris*, *FreeBSD* i *NetBSD* Unixa. Arhitektura *Quagga* se sastoji od centralnog servisa (*zebra*) koji čini sloj apstrakcije prema kernelu (jezgri sustava) odnosno nudi takozvani Zserv API prema Unix socketu ili TCP socketu te prema *Quagga* klijentima. *Zserv* klijenti implementiraju protokole usmjeravanja obično prema principu: jedan protokol – jedan servis te prenose njihova ažuriranja usmjeravanja *zebra* servisu. Međutim i u razvoju ovog projekta je bilo problema te je konačno nastao novi projekt (od 2017.g.) na ovom dizajnu i prethodnom radu, koji se naziva: ***Free Range Routing*** ili ***FRRouting*** to jest jednostavno ***FRR***. Ovaj projekt podržava i Linux fondacija te je kasnije široko prihvaćen i od velikih igrača na ovom području, poput tvrtki: *Cumulus*, *big switch*, *Pluribus Networks*, *VMware* i mnogih drugih. Paralelno s navedenim projektima od 2000. godine, razvijao se i projekt ***BIRD*** (*BIRD Internet Routing Daemon*) koji se i dalje razvija i koristi u određenim primjenama (pr. u nekim slučajevima se pokazao mnogo skalabilnijim od ***FRR***). Međutim (2022.g.) ***FRR*** je praktično postao standard za usmjeravanje pod mnogim distribucijama Linuxa i raznim varijantama Unixa pa ćemo dalje govoriti isključivo o njemu.

***Free Range Routing*** podržava mnoge protokole za usmjeravanje, ali i neke druge protokole i servise poput:

- Vršnog ***Zebra*** servisa, za koji su zaduženi servisi (*deamoni*) `zebra` (*term. port **2601***) i/ili `zebrasrv` (*term. port **2600***) te mrežni protokoli poput:
    - *Babel protokola*, preko servisa `babeld` *sa sučeljem terminala na portu: **2609**.*
    - *Bidirectional Forwarding Detection protokola* (**BFD**), preko servisa `bfdd` *sa sučeljem terminala na portu: **2617**.*
    - *Border Gateway Protocol* (**BGP**), preko servisa `bgpd` *sa sučeljem terminala na portu: **2609**.*
    - *Enhanced Interior Gateway Routing Protocol (**EIGRP**), preko servisa* `eigrpd` *sa sučeljem terminala na portu **2613**.*
    - *Intermediate System to Intermediate System* (**IS-IS**), *preko servisa* `isisd` *sa sučeljem terminala na portu **2608**.*
    - *Label Distribution Protocol* (**LDP**), *preko servisa* `ldpd` *sa sučeljem terminala na portu **2612**.*
    - *Open Fabric, preko servisa* `fabricd` *sa sučeljem terminala na portu **2618**.*
    - *Next Hop Resolution Protocol* (**NHRP**), *preko servisa* `nhrpd` *sa sučeljem terminala na portu **2610**.*
    - *Open Shortest Path First (**OSPF**v2), preko servisa* `ospfd` *sa sučeljem terminala na portu **2604**.*
    - *Open Shortest Path First (**OSPF**v3), preko servisa* `ospf6d` *sa sučeljem terminala na portu **2606**.*
    - *Path Computation Element (PCE) Communication Protocol (PCEP) preko servisa* `pathd`.
    - *Protocol Independent Multicast (**PIM**), preko servisa* `pimd` *sa sučeljem terminala na portu **2611.***
    - *Routing Information Protocol (**RIP**), preko servisa* `ripd` *sa sučeljem terminala na portu **2602.***
    - *Routing Information Protocol Next Gen (**RIPng**), preko servisa* `ripngd` *sa sučeljem terminala na portu **2603.***
    - *Virtual Router Redundancy Protocol (**VRRP**), preko servisa* `vrrpd` *sa sučeljem terminala na portu **2619.***
    - *... i mnogih drugih.*

Na *RedHat* kompatibilnim distribucijama Linuxa ***FRR*** je jednostavno instalirati. Međutim, ako želite zadnju stabilnu inačicu, (***RPM***), potrebno je dodati novi repozitorij. Sve ovisno o inačici *Red Hat* kompatibilne distribucije Linuxa koju imate:

***Za Red Hat 7.x. kompatibilne distribucije Linuxa:***
```
yum -y install https://rpm.frrouting.org/repo/frr-stable-repo-1-0.el7.noarch.rpm
```

***Za Red Hat 8.x. kompatibilne distribucije Linuxa:***
```
yum -y install https://rpm.frrouting.org/repo/frr-stable-repo-1-0.el8.noarch.rpm
```

Te je na kraju potrebno i instalirati ***FRR*** pakete softvera:
```
yum -y install frr frr-pythontools
```

Odnosno, ako ipak želite instalaciju ***FRR*** paketa nešto starije inačice, ali koji dolaze s vašim postojećim repozitorijima, onda nemojte dodavati gornje repozitorije već samo napravite instalaciju na sljedeći način:
```
yum -y install frr
```

U svakom slučaju, nakon instalacije dobivamo centralnu *FRR* konfiguracijsku datoteku (`/etc/frr/daemons`) u kojoj je potrebno definirati koji od protokola za usmjeravanje želimo aktivirati, tako da za svaki protokol imamo naveden njegov servis (kako smo gore popisali), koji je potrebno aktivirati.

Pogledajmo samo manji dio `/etc/frr/daemons` konfiguracijske datoteke:

```
bgpd=no
ospfd=no
ospf6d=no
ripngd=no
isisd=no
```

Ako primjerice želimo aktivirati ***RIP*** protokol, tada umjesto `ripd=no`, moramo postaviti: `ripd=yes`.
Nakon što promjenu snimimo, potrebno je pokrenuti ***FRR*** servis te ga trajno aktivirati sa sljedećim naredbama:

```
systemctl start frr
systemctl enable frr
```

Sada u našem slučaju u kojem smo uključili samo ***RIP*** protokol, dobivamo sljedeće, ako pokrenemo naredbu:

```
systemctl status frr

● frr.service - FRRouting
Loaded: loaded (/usr/lib/systemd/system/frr.service;enabled;vendor preset: disabled)
Active: active (running) since Sat 2022-05-21 20:58:38 CEST; 4s ago
Docs: https://frrouting.readthedocs.io/en/latest/setup.html
Process: 3489 ExecStart=/usr/lib/frr/frrinit.sh start (code=exited, status=0/SUCCESS)
 Main PID: 3494 (watchfrr)
   Status: "FRR Operational"
    Tasks: 9 (limit: 12438)
   Memory: 12.1M
   CGroup: /system.slice/frr.service
           ├─3494 /usr/lib/frr/watchfrr -d -F traditional zebra ripd staticd
           ├─3511 /usr/lib/frr/zebra -d -F traditional -A 127.0.0.1 -s 90000000
           └─3516 /usr/lib/frr/ripd -d -F traditional -A 127.0.0.1
localhost.localdomain watchfrr[3494]: [QDG3Y-BY5TN] zebra state -> up : connect succeeded
localhost.localdomain systemd[1]: Started FRRouting.
localhost.localdomain watchfrr[3494]: [QDG3Y-BY5TN] ripd state -> up : connect succeeded
localhost.localdomain watchfrr[3494]: [KWE5Q-QNGFC] all daemons up, doing startup-complete
```

Nadalje, svaki od protokola usmjeravanja koje smo naveli, ima svoju konfiguracijsku datoteku, prema imenu servisa, a koja se mora nalaziti unutar vršnog direktorija `/etc/frr/`. i imati ekstenziju datoteke `.conf`.
Tako je primjerice za **RIP** protokol (servis `ripd`) njegova konfiguracijska datoteka: `/etc/frr/ripd.conf`, dok je za ***OSPF*** protokol (servis `ospfd`) njegova konfiguracijska datoteka: `/etc/frr/ospfd.conf`, te je za ***BGP*** protokol (servis `bgpd`) to datoteka: `/etc/frr/bgpd.conf` i tako dalje.



Unutar svake konfiguracijske datoteke, specifične za pojedini mrežni protokol za koji je zadužena, potrebno je pratiti upute s naredbama specifičnim za određeni protokol. To je stoga jer svaki protokol ima specifične naredbe, a koje su vrlo slične ili čak identične naredbama kakve možemo vidjeti na usmjerivačima poznatijih tvrtki poput: ***Juniper***, ***Cisco***, ***Palo Alto***, ***F5 Networks***, ***Extreme networks*** i drugih.

Ako želimo raditi na Linuxu kao na usmjerivaču s objedinjenim svim mrežnim protokolima u jednoj ljusci, ***FRR*** nam nudi posebnu naredbenu ljusku, naziva `vtysh`. U njoj radimo slično kao na ***Cisco IOS*** usmjerivačima, pa ju stoga i pokrenimo:

```
vtysh
```



Za detalje oko ubrzavanja rada **FRR** i drugih sličnih servisa, pogledajte poglavlje:
**26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP* + *eBPF*).**



Za konfiguracijske detalje i naredbe, pogledajte izvor informacija: **(1219)** te **FRR** *dokumentaciju za*: ***BFD***, ***BGP***, ***EIGRP***, **IS-IS**, **LDP**, *Open Fabric*, **NHRP**, ***OSPFv2***, ***OSPFv3***, **PATH**, ***PIM***, ***RIP***, ***RIPng***, ***VRRP*** i druge protokole.

**Izvori informacija: (1218),(1219),(1220), `man vtysh`.**
**FRR** *dokumentacija prema protokolima za*: ***BFD***, ***BGP***, ***EIGRP***, **IS-IS**, **LDP**, *Open Fabric*, **NHRP**, ***OSPFv2***, ***OSPFv3***, **PATH**, ***PIM***, ***RIP***, ***RIPng***, ***VRRP*** **i druge.**

# 22. Metode komunikacije

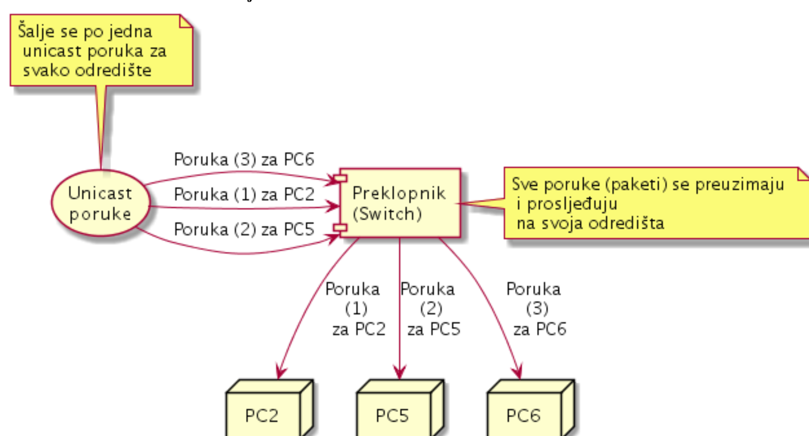
Različiti mrežni protokoli koriste različite metode odnosno načine komunikacije.
Najčešće razlikujemo tri osnovne metode komunikacije:

- Direktna komunikacija *(Engl. Unicast).*
- Komunikacija emitiranjem *(Engl. Broadcast).*
- Komunikacija odašiljanjem na grupu *(Engl. Multicast).*
- Komunikacija u kojoj jedna odredišna adresa ima više putanja usmjeravanja do dva ili više odredišta krajnjih točaka (*Engl. Anycast*).

## 22.1. Unicast

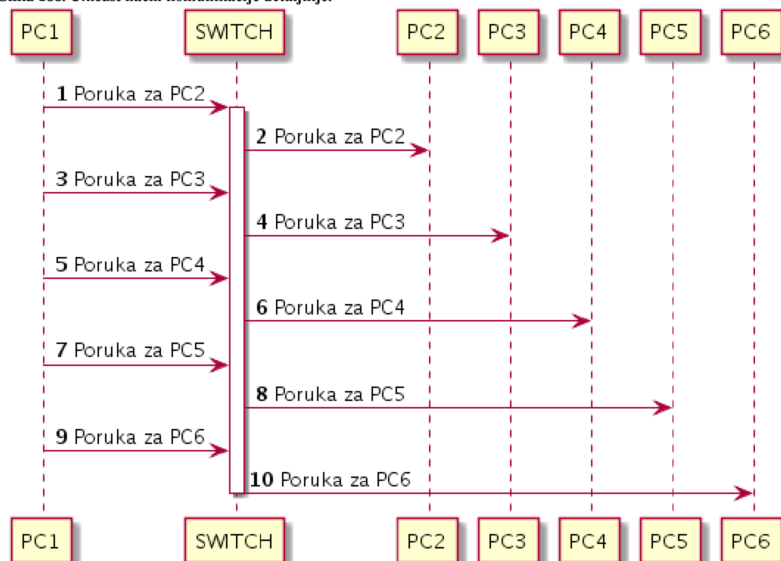
*Unicast* se koristi za komunikaciju koja se naziva „*Host-to-host*" odnosno (direktna) komunikacija dva računala ili uređaja na mreži. Koristi se kada dvije krajnje strane u komunikaciji trebaju komunicirati sâmo i direktno među sobom bez potrebe da se u komunikacijski kanal između njih uključuje i netko treći. Sjetimo se kako su ***MAC*** adrese mrežnih kartica jedinstvene, kao i IP adrese računala u komunikaciji. Stoga u komunikaciji s protokolima koji komuniciraju između dvije MAC adrese ili između dvije IP adrese kao identifikatore krajnjih točaka komunikacije, kažemo kako je takva komunikacija *unicast* vrsta komunikacije.
Dakle to bi bio sljedeći oblik (direktne) komunikacije: **IP 1 (Računalo 1) ←→ IP 2 (Računalo 2)**
Ovakva komunikacija, logički je prikazana na slici **187**.

**Slika 187. Unicast način komunikacije.**




Sljedeća komunikacija je također *unicast* komunikacija:
IP adresa: 192.168.100.10 direktno komunicira s IP adresom:192.168.100.20, kao i slučaj u kojem IP adresa: 192.168.100.10 direktno komunicira s IP adresom: 10.14.5.20 i slično.
Kod *unicast* komunikacije slanje podataka (paketa) logički izgleda kao na slici **188**. U ovom primjeru prvo računalo (***PC1***) želi komunicirati s računalima: ***PC2***, ***PC3***,***PC4*** i ***PC5*** pa prema tome mora slati pet nezavisnih mrežnih paketa (poruke vidljive kao: **1**,**3**,**5**,**7**,**9**), svaki prema svakom pojedinom odredišnom računalu, i to sve preko mrežnog preklopnika (*switcha*) koji ih šalje dalje prema odredištu.

**Slika 188. *Unicast* način komunikacije detaljnije.**

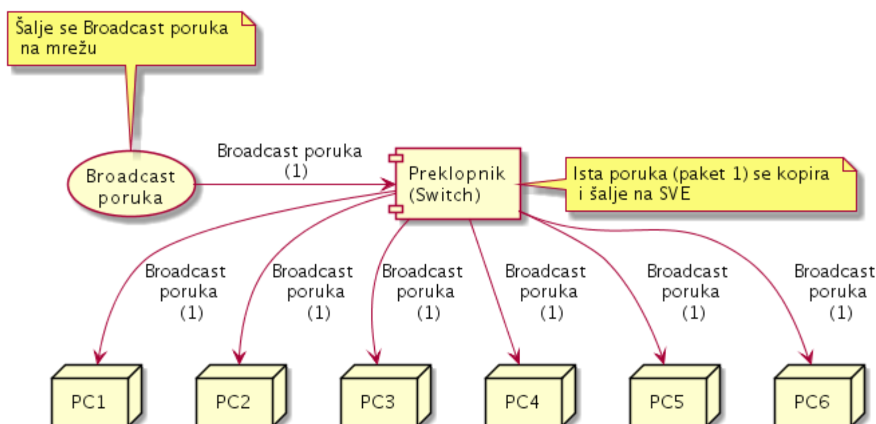



**Izvori informacija: (K-6), (K-10).**

# 22.2. Broadcast

*Broadcast* način komunikacije koristi se kada se određeni mrežni paket želi poslati na sva računala na mreži. Prema logici rada ovo je suprotnost u odnosu na *unicast*. Zamislimo IP komunikaciju u mreži: 192.168.100.0 s maskom mreže 255.255.255.0. Dakle govorimo o mreži 192.168.100.*. Znamo da je IP adresa 192.168.100.255 u ovoj mreži, *Broadcast* IP adresa ove mreže. Ako neko računalo iz postojeće mreže, a to je bilo koja IP adresa od 192.168.100.1 do 192.168.100.254, pošalje neki paket na adresu 192.168.100.255 (ovo je posebna odnosno *Broadcast IP adresa*) taj paket će primiti apsolutno ***SVI*** na toj mreži.
Dakle *broadcast* komunikacija je način komunikacije u kojemu jedan šalje na sve prema principu: ***jedan → SVI***
Takav način komunikacije, logički izgleda kao na slici 189.

**Slika 189. Broadcast način komunikacije.**




*Broadcast* način komunikacije je potreban za sâmo neke protokole, a za sve ostale nepotrebno zagušuje cijelu mrežu.
***Što se transportnih protokola tiče, Broadcast komunikacija je moguća samo upotrebom UDP, ali ne i TCP protokola!***

## Broadcast domena

Razni protokoli za svoj rad tj. u komunikaciji koriste *broadcast* način komunikacije koji radi tako da se poruke šalju na specijalnu *broadcast* adresu koju primaju svi uređaji: računala, poslužitelji i drugi, unutar određene mreže. Problem je u tome što ovakav način komunikacije zagušuje mrežu. ***Hub***-ovi i preklopnici (*switchevi*) propuštaju *broadcast* komunikaciju odnosno takozvanu *broadcast* domenu, a usmjerivači (*routeri*) standardno ju **ne** propuštaju. Dakle ako imamo potrebu ograničiti *Broadcast* domenu, mreža mora biti razdvojena na segmente koji su spojeni preko usmjerivača ili drugih uređaja koji rade na OSI sloju tri (pr. *Multilayer* preklopnika).

**Izvori informacija: (K-6), (K-10).**

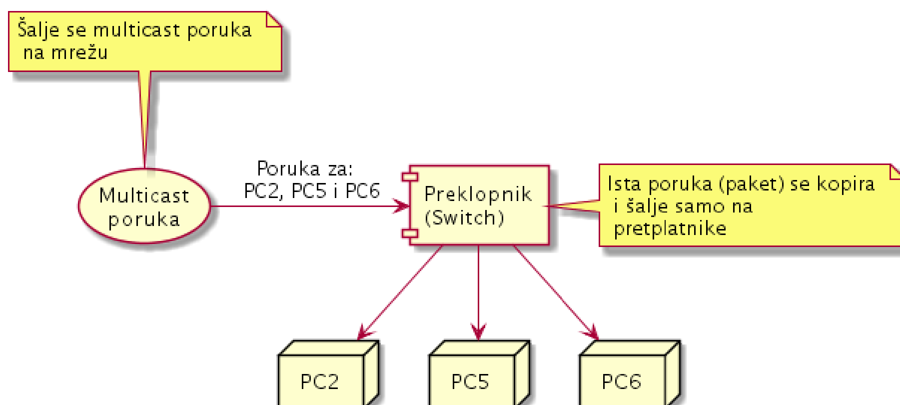
---

# 22.3. Multicast

*Multicast* protokol definiran je prvi puta u standardu RFC 1112 te proširen u RFC 4604 i u RFC 5771.
*Multicast* komunikacija je komunikacija prema principu: **jedan → više** njih, odnosno jedan na odabranu grupu (*multicast*) korisnika/računala. U primjeru na slikama dolje, sâmo računala: **PC2**, **PC5** i **PC6** su se pretplatila na *multicast* grupu na koju **pošiljatelj** šalje podatke.

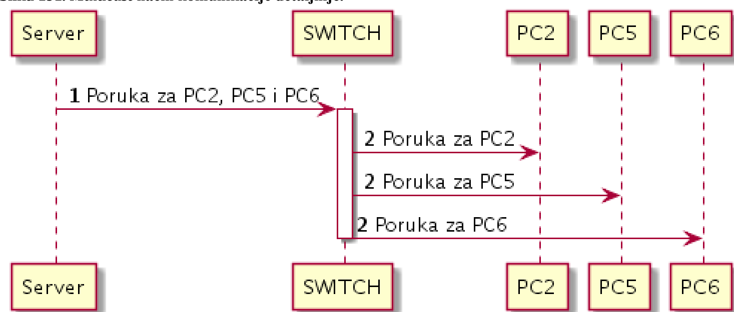
To logički izgleda kako je vidljivo na slici 190.

**Slika 190. Multicast način komunikacije detaljnije.**




*Multicast* komunikacija tada izgleda kao na slici 191.

**Slika 191.** ***Multicast*** **način komunikacije detaljnije.**




Prvo računalo u komunikaciji (***Server***) šalje jednu (**1**) jedinu poruku (mrežni paket) na *multicast* grupu u kojoj se nalazi više računala, a u našem slučaju su to: **PC2**, **PC5** i **PC6**.
O pripadnosti *multicast* grupi se ovdje brinu preklopnik i/ili usmjerivač, koji onda tu poruku raspoređuju (kopiraju) prema svim računalima koja pripadaju toj *multicast* grupi odnosno na sva računala koja su se pretplatila na tu grupu.

*Multicast* IP adrese su jedinstvene IP adrese koje prosljeđuju paket s *multicast* odredišnom adresom na pred definiranu grupu računala na mreži. Tako da svako pojedino računalo koje šalje podatke na pojedinu *multicast* adresu, zapravo može slati podatke na višestruke primaoce. *Multicast* je baziran na konceptu grupa. Svaki korisnik (*Host*/računalo) može se pretplatiti na neku *Multicast* grupu te će primati sadržaj namijenjen za tu grupu. *Multicast* IP adrese su iz **D** klase IP adresa.

**Multicast adrese su adrese unutar sljedećeg točno definiranog opsega IP adresa:**

**Za IPv4** protokol je to sljedeći opseg IP adresa:

| Početna *multicast* IP adresa | Zadnja *multicast* IP adresa |
|---|---|
| 224.0.0.0 | 239.255.255.255 |

**Za IPv6** protokol je to sljedeći opseg IP adresa:

| Početna *multicast* IP adresa | Zadnja *multicast* IP adresa |
|---|---|
| FF00:0000:0000:0000:0000:0000:0000:0000 | FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF |

Vratimo se na **IPv4** protokol jer ćemo govoriti uglavnom o njemu. Ovdje postoji niz *multicast* adresa koje su rezervirane za lokalnu upotrebu te one rezervirane za točno određenu namjenu. Kraća lista rezerviranih *multicast* adresa izgleda ovako:

| **Multicast adresa** | **Namjena** |
|---|---|
| 224.0.0.0 | Rezervirano. |
| 224.0.0.1 | Svi (*hostovi/računala/uređaji*) unutar mrežnog segmenta, koji koriste *multicast*. |
| 224.0.0.2 | Svi usmjerivači (*Routeri*) unutar mrežnog segmenta, koji koriste *multicast*. |
| 224.0.0.4 | Svi usmjerivači (*Routeri*) koji koriste *Distance Vector Multicast Routing Protocol* (*DVMRP*) unutar mrežnog segmenta. |
| 224.0.0.5 | ***OSPF*** protokol za slanje "*HELLO*" paketa unutar mrežnog segmenta. |
| 224.0.0.9 | ***RIP*** *v.2* protokol za slanje informacija usmjeravanje svim ***RIP v.2*** usmjerivačima. |
| 224.0.0.18 | *Virtual Router Redundancy Protocol* (**VRRP**) komunikacija. |
| 224.0.0.22 | *Internet Group Management Protocol* (**IGMP v3**) uređaji. |
| →224.0.1.255 | … slijedi niz rezerviranih *multicast* adresa |
| 224.0.2.0 | Odavde počinju *multicast* IP adrese koje nisu rezervirane za posebne mrežne protokole. |

*Dakle navedene rezervirane multicast adrese ne smijemo koristiti za svoje potrebe, jer svaka od njih već ima svoju namjenu.*

*Multicast* komunikacija se odrađuje u dva koraka:

- ***Aplikacijski***: ovdje se sama aplikacija ili operativni sustav, brinu o prijavljivanju na *multicast* grupe, na mrežne uređaje preko kojih se odrađuje pretplaćivanje na konkretne *multicast* grupe.
- ***Na mrežnoj razini***: mrežni uređaji primaju zahtjev od aplikacije i dodjeljuju računalo u *multicast* grupu (da ne bude zabune to su *multicast* IP adrese). Dakle svi koji žele biti članovi određene *multicast* grupe moraju se prijaviti mrežnom uređaju da ih uključi u tu grupu. Kada netko šalje paket na tu grupu tada mrežni uređaj u pozadini taj isti paket kopira i prosljeđuje na sve one koji su se pretplatili da budu članovi te (*multicast*) grupe.

Kada aplikacija (proces) više ne želi biti pretplaćen na određenu *multicast* grupu, onda taj zahtjev šalje prema operativnom sustavu (*kernelu*). Nadalje je zadaća operativnog sustava da se odjavi iz tražene *multicast* grupe, a isto je i sa prijavljivanjem.



**Kao transportni protokol, za multicast se obično koristi UDP protokol (ili neki drugi), ali ne i TCP!**
**Osim IP adresa, za multicast se koriste i MAC adrese iz posebnog opsega adresa (o tome u sljedećoj cjelini).**

**Izvori informacija: (697),(698),(699),(700),(K-6),(K-10), RFC 1112, RFC 4604 i RFC 5771.**

### 22.3.1. Oblik *multicast* poruke (paketa)

**Slijedi napredna cjelina!**

Pogledajmo izgled jednog *multicast* paketa na mreži, a koji **standardno** koristi *UDP* protokol kao transportni protokol:

```
1.  Frame 1: 112 bytes on wire (896 bits), 112 bytes captured (896 bits) on interface 0
2.  Ethernet II, Src: 08:00:27:ec:c0:1a, Dst: 01:00:5e:01:02:03
3.  Internet Protocol Version 4, Src: 192.168.1.149, Dst: 234.1.2.3
4.  User Datagram Protocol, Src Port: 4321, Dst Port: 4321
```

Ako gledamo ovaj TCP/IP paket, *multicast* dio ima važnost u dva polja:

a) Na OSI sloju dva – MAC (`2.`) jer je kod *multicast* paketa (`Dst: 01:00:5e:01:02:03`) odnosno odredišna MAC adresa obično je adresa iz opsega od **`01-00-5E`**`-00-00-00` do **`01-00-5E`**`-7F-FF-FF`. Prije ovog opsega *multicast* IP adresa rezervirane su još neke za posebnu namjenu, kao i nakon ovog opsega (**tablica dolje**).

b) Na OSI sloju tri – IP (`3.`) jer je (`Dst: 234.1.2.3`) odnosno odredišna IP adresa uvijek iz opsega od `224.0.0.0` do `239.255.255.255` koja i predstavlja *multicast* IP adresu (za IPv4). O ovim adresama nešto kasnije.

**Pogled na neke od rezerviranih Multicast MAC adresa:**

| Multicast MAC Adresa | Namjena |
|---|---|
| 01-00-0C-CC-CC-CC | **CDP** (*Cisco Discovery Protocol*), **VTP** (*VLAN Trunking Protocol*) |
| 01-80-C2-00-00-00 | *Spanning Tree Protocol* (IEEE 802.1D) |
| 01-80-C2-00-00-00<br>01-80-C2-00-00-03<br>01-80-C2-00-00-0E | **LLDP** *Link Layer Discovery Protocol* |
| 01-80-C2-00-00-08 | *Spanning Tree Protocol* (IEEE 802.1ad) |
| 01-80-C2-00-00-01 | *Ethernet flow control* (Pause frame: IEEE 802.3x) |
| **01-00-5E-00-00-00→01-00-5E-7F-FF-FF** | Opseg IPv4 *multicast* adresa dostupan za korištenje. |

Za komunikaciju klijenata (računala) s mrežnim uređajima, odnosno za njihovo prijavljivanje i odjavljivanje s *multicast* grupa koristi se ***IGMP*** protokol. ***IGMP*** (*Internet Group Management Protocol*) protokol **v.1** definiran je u standardu RFC 1112, inačica 2. je definirana u RFC 2236, a inačica **3** u standardu RFC 3376.

Pogledajmo sada i strukturu *IGMP* paketa koji radi na OSI sloju tri (IP). Na slici je IGMP **v.3**. uz vidljive dijelove OSI slojeva **1**, **2** i **3** kako je vidljivo dolje na ispisu jednog *IGMP* mrežnog paketa:

```
1.  Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
2.  Ethernet II, Src: 08:00:27:ec:c0:1a, Dst: 01:00:5e:00:00:16
3.  Internet Protocol Version 4, Src: 192.168.1.149, Dst: 224.0.0.22
      Internet Group Management Protocol
         [IGMP Version: 3]
         Type: Membership Report (0x22)
         Reserved: 00
         Checksum: 0xedf9 [correct]
         [Checksum Status: Good]
         Reserved: 0000
         Num Group Records: 1
           Group Record : 234.1.2.3  Change To Exclude Mode
```

Sada pogledajmo malo detaljnije, samo dio **IGMP v.2.** paketa:

<table>
<tr><th colspan="4">IGMP v.2. paket</th></tr>
<tr><th>Version</th><th>Type</th><th>Max Resp Time</th><th>Checksum</th></tr>
<tr><th colspan="4">Group Address</th></tr>
</table>

Slijedi opis polja unutar ***IGMP*** paketa:

- `Version` - označava inačicu (inačicu) *IGMP* protokola, moguće vrijednosti su: 1,2 ili 3
- `Type` - označava vrstu poruke (što ovisi i o inačici *IGMPa*):
  - `Membership Query (0x11)` - za sve inačice *IGMP* protokola.
  - `Membership Report`: ovisi o vrsti *IGMP* protokola.
    - `0x12` – ovo je postavljena vrijednost, ako se radi o *IGMP v1*.
    - `0x16` – ovo je postavljena vrijednost, ako se radi o *IGMP v2*.
    - `0x22` – ovo je postavljena vrijednost, ako se radi o *IGMP v3*.
  - `Leave Group (0x17)` - za sve inačice *IGMP* protokola.
- `Max Resp Time` - maksimalno vrijeme koje *IGMP* klijent ima za odgovor. Ovo polje je važno samo za poruku *Membership Query* (`0x11`).
- `Group Address` - ovo je *multicast* IP adresa za koju se šalje upit. Ako je vrijednost nula (`0.0.0.0`) tada se upit šalje za sve *multicast* IP adrese.
- `Checksum` - ovdje se zapisuje provjerni zbroj/izračun (*checksum*) za *IGMP* dio paketa.

*IGMP* protokol v.2. `Membership Query` šalje na IP adresu: `224.0.0.1` (sva *multicast* sposobna računala), dok `Leave Group (0x17)` poruku šalje na IP adresu: `224.0.0.2` (svi *multicast* usmjerivači).
Struktura *IGMP* v.3. paketa je malo složenija, pa ju sada nećemo objašnjavati.
Pogledajmo što su nam donijele koje *IGMP* inačice protokola:

- **IGMP v.1.** imala je sustav *query - response* (upit - odgovor). Ova inačica omogućavala je sâmo prijavljivanje *multicast* klijenta na željenu *multicast* grupu.
- **IGMP v.2.** je poboljšala brzinu rada u odnosu na prvu inačicu. Dodatno je uvedena mogućnost odjavljivanja iz *multicast* grupe (0x17) te upite (*query*) za točno definirane *multicast* grupe kao i maksimalno vrijeme unutar kojeg klijent mora poslati odgovor.
- **IGMP v.3.** dodatno proširuje funkcionalnosti, uvodeći i filtriranje *multicast* grupa.

**Izvori informacija: (698),(1443), RFC 1112, RFC 2236, RFC 3376.**

## 22.3.2. Kako to izgleda u praksi

Pogledajmo kako izgleda komunikacija korištenjem *multicasta*. U prvom koraku se računala prijavljuju preklopniku (*switch*) na željenu *multicast* grupu (koji to mora podržavati). On tada gradi svoju tablicu za *multicast* pretplatnike (*snooping forwarding entry's*). U ovoj tablici je povezano mrežno sučelje preklopnika (*interface*) s pripadnosti spojenog računala *multicast* grupi kako bi se *multicast* poruke mogle prvo kopirati, a potom isporučiti do računala pretplatnika *multicast* grupe.
Pogledajmo kako izgleda jedan unos u ovoj tablici preklopnika:

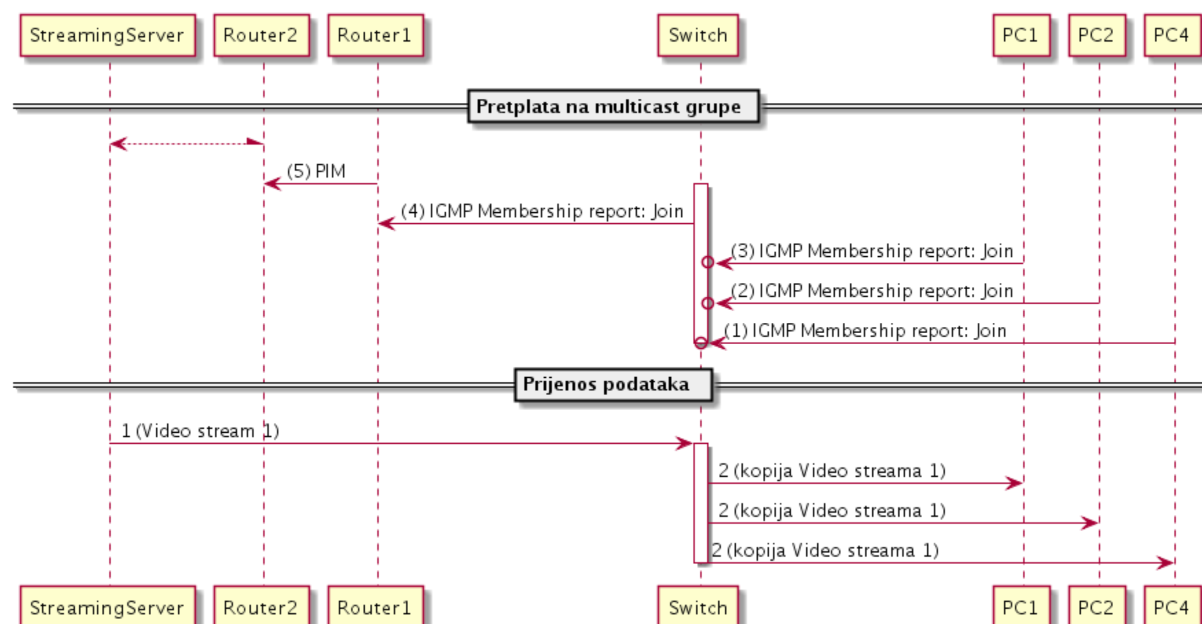
| Rb. | VLAN | Multicast MAC ili IP | LTL | Mrežno sučelje preklopnika (Port/interface) |
|---|---|---|---|---|
| 1. | 200 | (*,224.1.2.3) | 0x879 | . |
| 2. | 200 | (*,224.1.2.3) | 0x924 | Gi 1/0/2 |
| 3. | 200 | (172.17.15.78,224.1.2.3) | 0x78F | Gi 1/0/10 |

Opis unosa u tablici slijedi:
**1.** Prvi unos (Rb.**1.**) se odnosi na cijelu *multicast* grupu (`224.1.2.3`) i rad s njom.
**2.** Drugi unos (Rb.**2.**) je specifičan za jedno mrežno sučelje na preklopniku:

- `200` je **VLAN** unutar kojeg prolazi ovaj *multicast* promet.
- `(*,224.1.2.3)` znači:
  - `*` označava kako se dozvoljava *multicast* s bilo koje IP adrese na mreži. Naime *IGMP v.3* nudi mogućnost ograničavanja s kojih IP adresa se dozvoljava promet na pojedini *multicast*, prema pojedinom računalu (***IGMPv3 join*** s uključenim ***source filteringom***).
  - `224.1.2.3` je konkretna *multicast* grupa odnosno njena IP adresa.
- `0x924` je pozicija u *ASIC sklopu*, koja je zadužena za prosljeđivanje ove *multicast* grupe na ovom mrežnom sučelju tj. *interfaceu* oznake ***Gi 1/0/2***.
- `Gi 1/0/2` je mrežno sučelje preklopnika (*interface)* (Gi 1/0/2) na koji će se slati ovaj *multicast* promet.

**3.** Treći unos (Rb.**3.**) je specifičan za mrežno sučelje (Gi 1/0/10) na preklopniku, a unos je za *IGMP v.3* u kojemu je uključen *IGMP source filtering* po kojem se na ovo mrežno sučelje (Gi 1/0/10) na *multicast* grupu: `224.1.2.3`, dozvoljava primitak mrežnih paketa koji su došli samo s IP adrese: `172.17.15.78`. Pogledajmo primjer *multicast* komunikacije i to dio pridruživanja *multicast* grupi te potom slanja podataka na tu *multicast grupu;* kako je vidljivo na slici 192.

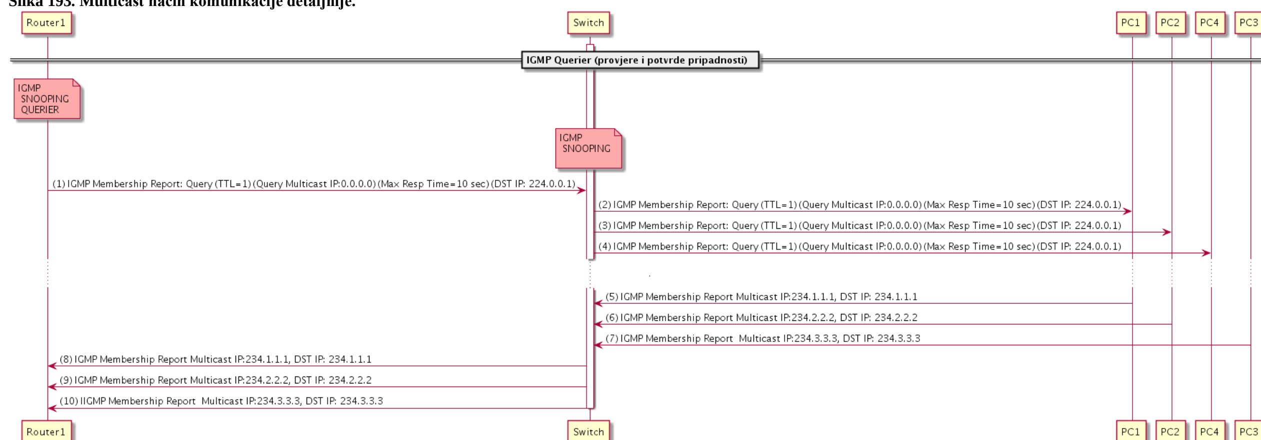
**Slika 192. Multicast način komunikacije i razmjena mrežnih paketa.**



U primjeru na slici **192.** računala: **PC1**, **PC2** i **PC4** su se pretplatila na istu *multicast* grupu (u prvom koraku) slanjem *IGMP* poruke (`IGMP Join`) na preklopnik (koji ih je u stanju obraditi odnosno koji podržava *multicast*). Naime preklopnik mora imati podršku za `IGMP snooping`. Preklopnik na osnovi pretplate (*IGMP Join*) klijenta/računala na određenu *multicast* grupu točnije *multicast* IP adresu, prosljeđuje *IGMP* poruku o pripadnosti klijenta *multicast* grupi, prema uređaju koji radi na OSI tri (OSI 3) sloju, a to je usmjerivač ili *multilayer* preklopnik koji podržava **IGMP Querier** i **PIM**.
Važno je razumjeti kako uređaj na OSI sloju dva (preklopnik) koji podržava `IGMP snooping` može korektno prosljeđivati **IGMP** poruke, između klijenta/računala i uređaja na OSI sloju tri (usmjerivača). Naime isti preklopnik prati na kojem svom mrežnom sučelju (*portu*), se koji klijent/računalo pretplatio ili odjavio iz koje *multicast* grupe. Potom na osnovu te spoznaje, kasnije može uredno prosljeđivati *multicast* mrežne pakete prema njima (klijentima/računalima).

*OSI* tri (OSI 3 [*router/usmjerivač*]) uređaj prima **IGMP** (*Internet Group Management Protocol*) poruke koje mu je proslijedio preklopnik, a koje mogu sadržavati ili zahtjev za pretplaćivanje na *multicast* grupu (*IGMP JOIN*) ili izlazak iz *multicast* grupe (*IGMP Leave Group*) i to za svakog klijenta/računalo zasebno. Usmjerivač za razliku od preklopnika ne gradi istu *multicast* tablicu kao preklopnik već sâmo tablicu u kojoj ima informacije koje sve *multicast* grupe (IP adrese) još uvijek postoje na ***LAN*** mreži (prema preklopniku) i na kojem su njegovom mrežnom sučelju, kako bi mogao na njih proslijediti *multicast* poruke.
S druge strane, ako u ***LAN*** mreži u nekom trenutku ne postoji niti jedan pretplatnik bilo koje *multicast* grupe, to znači kako on na tu *multicast* grupu neće slati nikakav promet. Kako bi se tablica odnosno lista aktivnih *multicast* grupa održavala obično *OSI tri* uređaj (usmjerivač) mora imati funkcionalnost znanu kao `IGMP querier`.
Ovoj funkcionalnosti je namjena slanje `IGMP Membership Queries` upita prema klijentima/računalima koja su se prijavila za pripadnost *multicast* grupama, odnosno pitanje žele li još uvijek pripadati toj grupi. Ako niti jedan klijent/računalo ne potvrdi pripadnost traženoj *multicast* grupi unutar zadanog vremenskog okvira, usmjerivač briše konkretnu *multicast* grupu. I napredniji preklopnici mogu preuzeti funkcionalnost `IGMP queriera` isto kao i *Linux* računala. Pogledajmo dijagram toka za ovaj slučaj, vidljiv na slici **193**.

**Slika 193. Multicast način komunikacije detaljnije.**




Konkretno `IGMP querier` šalje upit (`IGMP Membership Query`) na *multicast* (grupnu) IP adresu `224.0.0.1` koju primaju i na koju moraju odgovoriti svi klijenti/računala koji koriste *multicast.*
Pošto `TTL` vrijeme "života" paketa ima vrijednost jedan (*TTL=1*) to znači kako se ovaj paket ne može preko usmjerivača proslijediti na drugu IP mrežu, jer će paket već na prvom usmjerivaču biti odbačen.
`Max Resp Time` koji postavlja usmjerivač koji šalje *IGMP Query* upit postavljen je na maksimalno vrijeme unutar kojeg *IGMP klijent* mora poslati odgovor. Dakle *IGMP klijent* (računalo) pokreće svoj *brojač* prije nego odgovori, a odgovor mora poslati unutar vremena koje je postavio usmjerivač odnosno `IGMP Querier`. Ovaj *brojač* na strani *IGMP* klijenta postoji stoga kako svi *IGMP* klijenti ne bi u istom trenutku počeli slati odgovore prema `IGMP Querieru`.
Važno je znati kako se *IGMP Query* upit šalje na *multicast* IP adresu `224.0.0.1` kako bi ga primili svi *multicast* uređaji, ali u *IGMP* polju `igmp maddr` (*IGMP Multicast IP*) je navedena IP adresa: `0.0.0.0` što znači kako se upit šalje za sve *multicast* IP adrese odnosno grupe, a ne za po svaku od njih zasebno. Ovu poruku preklopnik kopira na sva mrežna sučelja prema računalima spojenim na njih, koja imaju aktivan *multicast* odnosno pretplaćeni su na neku *multicast* grupu.
U *IGMP v.3* protokolu `IGMP Querier` može slati i *IGMP Query* koji može sadržavati upite za točno određene *multicast* grupe (i više njih).
Kada su računala primila ovu poruku tada svako od njih koje još uvijek želi pripadati nekoj *multicast* grupi nakon isteka svog *timera* (`multicast_query_response_interval`), a unutar vremena (`Max Resp Time)` šalje odgovor odnosno *IGMP Membership Report* za sve one *multicast* grupe (IP adrese) kojima još uvijek želi pripadati. Ovaj odgovor se šalje isključivo na *multicast* IP adresu kojoj *IGMP* klijent želi pripadati. Tako ovu poruku vide svi, uključujući usmjerivač (`IGMP Querier`) i sve druge uređaje odnosno računala u mreži na konkretnoj *multicast* IP adresi. Zbog slanja paketa na ovu odredišnu adresu, bilo koje drugo računalo na mreži koje pripada istoj *multicast* IP adresi neće slati još jedan odgovor na `IGMP Queriera` jer njega (usmjerivač/`IGMP Querier`) sâmo zanima ima li ikoga tko želi ostati pretplaćen na tu *multicast* IP adresu.

*Usmjerivač/`IGMP Querier` ne želi biti preplavljen s istim porukama od svih pripadnika iste multicast grupe jer ga zanima sâmo postoji li barem i jedan pretplatnik multicast grupe.*

Dodatno (nije nacrtano na shemi) svi ostali klijenti koji žele i dalje ostati u svojim *multicast* grupama, i nakon isteka *brojača* (`multicast_membership_interval`) moraju sâmi periodički ponovno slati poruku `Membership Report JOIN` za konkretnu *multicast* grupu, kako ne bi bili izbačeni iz nje. S ovim periodičkim porukama osvježava se i *multicast* tablica na preklopniku tako da u njoj ostaju sâmo oni koji su osvježili podatak o tome da i dalje žele pripadati toj *multicast* grupi.

Pogledajmo neke od Linux varijabli (za *bridge sučelje* **`vmbr0`**) koje su vezane za navedene *multicast* brojače (*timere),* a neke ćemo opisati kasnije.

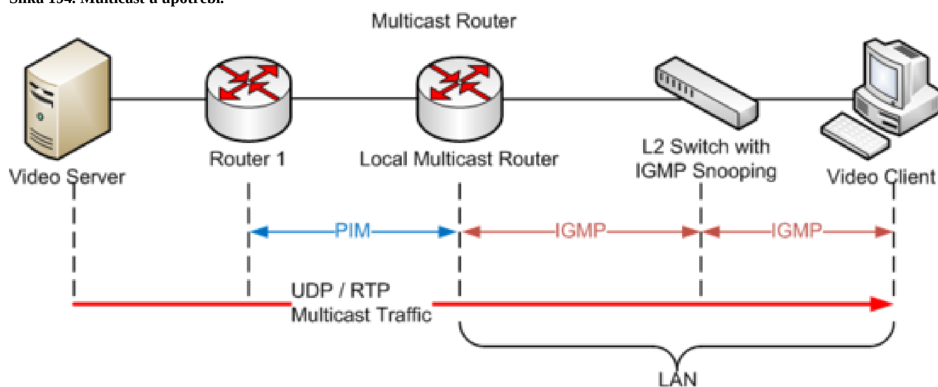
| /sys lokacija varijable | Standardna vrijednost | Opis |
|---|---|---|
| **/sys/devices/virtual/net/vmbr0/bridge/multicast_membership_interval** | 25996 ds (2599.6 sekundi) | Vrijeme u desetinkama sekunde, koliko će *IGMP* uređaj čekati, na potvrde od *IGMP* klijenta, prije nego ga izbaci iz *multicast* grupe i prestane slati *multicast* pakete na njega. |
| **/sys/devices/virtual/net/vmbr0/bridge/multicast_startup_query_count** | 2 | Broj *IGMP Query* poruka koje se šalju inicijalno, kako bi se saznali sve *multicast* pripadnosti (grupe). |
| **/sys/devices/virtual/net/vmbr0/bridge/multicast_startup_query_interval** | 3124 ds (312.4 sekundi) | Vrijeme u *deci* sekundama (desetinkama) između *IGMP query* upita koji se šalju na početku inicijalizacije *IGMP query* procedure. |

Vratimo se na dio priče o komunikaciji između dva usmjerivača iz predzadnje slike **192**. Dakle kada *OSI tri (OSI 3)* uređaj primi *IGMP* poruku pretvara ju u **PIM** (*Protocol-Independent Multicast*) format te ju preko *PIM* protokola prosljeđuje sljedećem *OSI tri* uređaju (usmjerivaču). Dakle usmjerivači razmjenjuju aktivne *multicast* grupe odnosno *multicast* IP adrese među sobom, kako bi znali na koji od usmjerivača uopće prosljeđivati koje *multicast* grupe.

Na kraju (na slici **192** s lijeve strane) nalazi se uređaj, u ovom slučaju *Video Streaming* poslužitelj, koji šalje video sadržaj (*stream*) sa svoje *multicast* IP adrese, na koju su se pretplatila pojedina računala, a u našem slučaju: **PC1**, **PC2** i **PC4**. Pošto su svi usmjerivači svjesni tko je sve pretplaćen na ovu *multicast* IP adresu, oni znaju i preko kojeg puta (*rute*) odnosno preko kojih drugih usmjerivača proslijediti ove mrežne pakete koje u konačnici prihvaćaju preklopnici na lokalnim mrežama, a koji su i prijavili pripadnost toj *multicast* grupi.

Oni na osnovi *IGMP* zahtjeva od svakog pojedinog klijenta znaju kojim klijentima proslijediti primljene pakete.
Video *Streaming* pošiljatelj šalje samo jedan tôk podataka (paketa) prema svima koji su pretplaćeni na *multicast* IP adresu na koju on šalje promet, tako da primjerice paket br. **1** prolazi sve do preklopnika (*Switch1*) koji ga kopira i šalje njegove kopije prema računalima **PC1**, **PC2** i **PC4**. Drugim riječima, ako netko želi kroz mrežu slati samo jedan tôk podataka na više odredišnih računala istovremeno, a ne otvarati i slati zaseban tôk (nîz) podataka prema svakom pojedinom odredišnom računalu, *multicast* je odgovor na to. Pogledajmo sve navedeno na malo drugačiji način, vidljiv na logičkoj shemi spajanja, na slici **194**.

**Slika 194. Multicast u upotrebi.**




Međutim, ako preklopnik ne podržava *IGMP* odnosno, ako nema mogućnost *IGMP Snoopinga* tada će preklopnik sve *multicast* poruke poslati na sva mrežna sučelja (*interface/portove*) osim onoga s kojega je paket došao, poput *broadcast* poruka. Stoga će ih primiti apsolutno sva računala spojena na preklopnik unutar iste ***VLAN*** mreže.

Za preklopnike koji nemaju niti podršku za *multicast* niti ***VLAN***ove to znači kako će *multicast* poruke biti definitivno poslane na sva mrežna sučelja preklopnika, naravno osim onoga s kojega je paket došao, pa će se *multicast* poruke također prosljeđivati kao da se radi o *broadcast* porukama.

Ovo se događa zbog toga što je odredišna **MAC** adresa svakog *multicast* paketa neka od adresa iz opsega od **`01-00-5E`**`-00-00-00` do **`01-00-5E`**`-7F-FF-FF`. Pošto preklopnik koji ne zna "raditi" s *multicast* porukama ne može imati tu odredišnu *MAC* adresu u svojoj *CAM* tablici preklapanja, on stoga *multicast* paket mora poslati na sva svoja mreža sučelja (*interface/portove*) osim onoga s kojeg je paket došao, ali unutar istog *VLANa* (ako uopće ima podršku za *VLANove)* i ako oni postoje i definirani su. Dakle isto što bi napravio i s drugim paketima za koje ne zna gdje im se nalazi odredišna *MAC* adresa.
Takvi mrežni paketi (u tom slučaju) su u statistikama preklopnika vidljivi kao ***unknown unicast*** paketi (ako preklopnik vodi statistike).



Vezano za **MAC** adrese, pogledajte poglavlje:
**19.3.1. Što su MAC adrese.**

Inače u normalnoj komunikaciji s nekog od računala kojem pripada konkretna odredišna *MAC* adresa bi bio poslan odgovor preklopniku pa bi preklopnik tu *MAC* adresu koja bi u odgovoru bila u polju izvorišne MAC adrese (*Source MAC*), preklopnik spremio u svoju *CAM* tablicu i povezao ju s mrežnim sučeljem na kojem se nalazi to računalo.

Stoga bi ubuduće znao paket proslijediti na to mrežno sučelje; prema *MAC* adresi tog računala.
Što se *multicasta* na "glupim" preklopnicima tiče, oni će sa svakim *multicast* paketom napraviti istu stvar dakle *multicast* paketi/poruke će se slati kao *broadcast* poruke na sva svoja mreža sučelja (portove) i sâmim time nepotrebno zagušivati cijelu lokalnu (***LAN***) mrežu.

### 22.3.3. *Multicast* u upotrebi

Svaki mrežni uređaj (pr. računalo) koji podržava *multicast* može se pretplatiti na željenu *IP multicast* adresu odnosno logički gledano, komunikacijski kanal. U prvom koraku računalo kojemu je omogućen *multicast* postavlja svom mrežnom sučelju (*mrežnoj kartici*) *multicast* IP adresu **224.0.0.1** jer je to prva definirana *multicast* IP adresa na koju i svako računalo/uređaj koji želi postati aktivan pretplatnik bilo koje *multicast* IP adrese (*multicast grupe*) mora moći odgovoriti.

Za Linux pogledajmo ima li naša mrežna kartica aktiviran *multicast*, što ćemo provjeriti pomoć naredbe `ifconfig`:

```
ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 00:08:60:00:79:20
          inet addr:192.168.1.254  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
          RX packets:2459 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3184 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1369224 (1.3 MiB)  TX bytes:455856 (445.1 KiB)
```

U trećem retku od gore vidimo kako je *multicast* omogućen s riječi: `MULTICAST`.
Isto smo mogli vidjeti i s `ip` naredbom na sljedeći način:

```
ip addr show eth0
```

**Kako aktivirati i deaktivirati podršku za *multicast* za mrežno sučelje *eth0*:**
Aktivacija pomoću naredbe `ifconfig` (isto možemo postići i s: `ip link set multicast on dev eth0`):

```
ifconfig eth0 multicast
```

Deaktivacija (isto možemo postići i s: `ip link set multicast off dev eth0`):

```
ifconfig eth0 -multicast
```



***Standardno je multicast omogućen na svim mrežnim sučeljima unutar Linuxa!***

Listu mrežnih sučelja koja imaju aktiviran *multicast* možemo dobiti i sa sljedećom naredbom:

```
cat /proc/net/dev_mcast
```

A popis *multicast* IP adresa (grupa), na koje smo se pretplatili, možemo dobiti s naredbom:

```
cat /proc/net/igmp
```

Isto ali malo čitljivije možemo dobiti s novijom naredbom `ip`:

```
ip maddress show
```

ili s naredbom `netstat` na sljedeći način:

```
netstat -gn
```

Prije nego krenemo dalje trebamo provjeriti podržava li uopće naš kernel *multicast* funkcionalnost:

```
cat /boot/config-`uname -r` | grep CONFIG_IP_MULTICAST
```

Ako dobijemo slijedeći odgovor (`=y`), sve je u redu:

```
CONFIG_IP_MULTICAST=y
```

Standardno je na operativnom sustavu Linux isključeno odgovaranje na *ICMP* poruke (koje šalje program `ping`) za *multicast* i *broadcast*) adrese za IPv4, dok je za IPv6 to uključeno. To možemo privremeno i isključiti (za IPv4), dok testiramo *multicast*.

| Lokacija varijable (*/proc*) | Sysctl varijabla | Standardna vrijednost |
|---|---|---|
| /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts | `net.ipv4.icmp_echo_ignore_broadcasts` | 1 |

**Prvo isključimo ovu zaštitu** ne odgovaranja na IPv4 ICMP *multicast* poruke:
```
sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
```



Za *IP* v.6 *ping* (naredba `ping6`) nije potrebno isključivati ovu varijablu odnosno funkcionalnost.

U našem slučaju, imamo dva Linux računala sa sljedećim IP adresama: `192.168.1.254` i `192.168.1.149` na kojima je standardno uključen *multicast*. S trećeg Linux računala probajmo *pingati multicast* adresu (`224.0.0.1`) na koju oni moraju moći odgovoriti, a pri tome šaljemo samo *3 ICMP* poruke (*ping*) na sljedeći način:
```
ping -c3 224.0.0.1

PING 224.0.0.1 (224.0.0.1) 56(84) bytes of data.
64 bytes from 192.168.1.149: icmp_seq=1 ttl=64 time=0.072 ms
64 bytes from 192.168.1.254: icmp_seq=1 ttl=64 time=0.398 ms (DUP!)
64 bytes from 192.168.1.149: icmp_seq=2 ttl=64 time=0.103 ms
64 bytes from 192.168.1.254: icmp_seq=2 ttl=64 time=0.463 ms (DUP!)
64 bytes from 192.168.1.149: icmp_seq=3 ttl=64 time=0.094 ms

--- 224.0.0.1 ping statistics ---
3 packets transmitted, 3 received, +2 duplicates, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.072/0.226/0.463/0.168 ms
```
Vidimo kako odgovor dobivamo s dvije različite IP adrese:

a) S prve IP adrese: `192.168.1.149`.
b) Te s druge IP adrese:`192.168.1.254`.

To je stoga jer oba računala odgovaraju na *ICMP* (*ping*) pa zato dobivamo i poruku kako su odgovori duplicirani: `(DUP!)`. To znači kako naša oba dva *multicast* računala odgovaraju na *ping*, na *multicast* grupu `224.0.0.1`.

Za ***RedHat/CentOS/Fedora*** i slične Linuxe, bazirane na *RedHatu* prvo instalirajmo poseban *ping* program za *multicast* jer nam on omogućava i pridruživanje željenoj *multicast* grupi odnosno *multicast* IP adresi automatski. Dakle instalirajmo ga ovako:
```
yum -y install omping
```

Na računalu `192.168.1.149` (**PC1**) pogledajmo pripadnost *multicast* grupama pomoću naredbe `netstat` na sljedeći način:
```
netstat -gn

IPv6/IPv4 Group Memberships
Interface       RefCnt Group
--------------- ------ ---------------------
lo              1      224.0.0.1
eth2            1      224.0.0.1
```
Odnosno isto možemo vidjeti i s novijom naredbom:
```
ip maddress show
```

Vidimo kako je naša mrežna kartica (`eth2`) prijavljena samo na *multicast* grupu `224.0.0.1` što je normalno jer je to adresa koju moraju imati svi uređaji koji podržavaju *multicast*. Krećemo s upotrebom novo instalirane naredbe `omping`.
Sada ćemo se na istom računalu: `192.168.1.149` (**PC1**) pretplatiti na *multicast* grupu: `234.1.2.3` na **port:** `12345`
```
omping -p 12345 -m 234.1.2.3 192.168.1.149
```

Sve dok je ova naredba/program (`omping`) pokrenuta, računalo je registrirano na *multicast* IP `234.1.2.3` te naredba `omping` sluša na UDP portu `12345`, a istovremeno šalje *UDP* pakete na taj port kako bi se potvrdila komunikacija preko *UDP* transportnog protokola. Dakle u ovom slučaju se ne koristi se *ICMP* protokol za *ping* funkcionalnost već se šalju *UDP* paketi na port `12345` i to:

- Jedan *UDP* paket na *unicast* IP adresu (na direktnu adresu računala).
- Jedan *UDP* paket na *multicast* IP adresu.

Na istom računalu sada ponovno pogledajmo pripadnost *multicast* grupama:
```
netstat -gn

IPv6/IPv4 Group Memberships
Interface       RefCnt Group
--------------- ------ ---------------------
lo              1      224.0.0.1
eth2            1      234.1.2.3
eth2            1      224.0.0.1
```

Dakle, isti ispis bi dobili i s novijom `ip` naredbom, ako ju pozovemo na sljedeći način:

```
ip maddress show
```

Vidimo da je našoj mrežnoj kartici (`eth2`) dodijeljena *multicast* IP adresa, *multicast* grupe u koju smo dodani (`234.1.2.3`).

Provjerimo i potvrdimo otvoreni UDP port `12345` na sljedeći način:

```
netstat -unap | grep -i 12345

udp          0       0 234.1.2.3:12345         0.0.0.0:*                    1644/omping
udp          0       0 192.168.1.149:12345     0.0.0.0:*                    1644/omping
```

Vidimo kako stvarno program `omping` sluša i prima UDP pakete na portu `12345`. Sada bi ovu *multicast* IP adresu mogli *pingati* s bilo kojeg drugog računala, čija *ping* naredba podržava *ping* na *multicast* IP adrese.



Sve implementacije naredbe ***ping*** pod linuxom podržavaju *ping* na *multicast* IP adrese.

Ako pak pogledamo pakete na mreži kod pokretanja programa `omping` te njegovog zaustavljanja nakon par trenutaka, vidljivo je sljedeće.

Paketi na mreži su popisani redoslijedom kako su dolazili te smo ih označili rednim brojevima:

```
1.   192.168.1.149 :waiting for response msg
2.   192.168.1.149 :joined (S,G) = (*, 234.1.2.3), pinging
3.   192.168.1.149 :unicast, seq=1, size=69 bytes, dist=0, time=0.004ms
4.   192.168.1.149 :multicast, seq=1, size=69 bytes, dist=0, time=0.007ms
5.   192.168.1.149 :unicast, seq=2, size=69 bytes, dist=0, time=0.020ms
6.   192.168.1.149 :multicast, seq=2, size=69 bytes, dist=0, time=0.022ms
7.   192.168.1.149 :unicast,xmt/rcv/%loss=2/2/0%,min/avg/max/std-dev=0.004/0.012/0.020/0.011
8.   192.168.1.149 :multicast,xmt/rcv/%loss=2/2/0%,min/avg/max/std-dev=0.007/0.015/0.022/0.011
```

Pogledajmo ove poruke, redom kako su dolazile:
(`1.`) Prva poruka je slanje probnog paketa na *multicast* IP adresu.
(`2.`) Druga poruka je pretplata na *multicast* grupu `234.1.2.3`.
(`3.`) Treća poruka je slanje prve *unicast* UDP poruke (*SEQ=1*) direktno na IP adresu računala koje provjeravamo, jer se može raditi i o udaljenom računalu.
(`4.`) Četvrta poruka je slanje prve *multicast* UDP poruke (*SEQ=1*).
(`5.`) Peta poruka je ponovno *unicast* (*SEQ=2*), a šesta (`6.`) *multicast* (*SEQ=2*).
Sedma (`7.`) i osma (`8.`) poruka su izvještaji o primljenim i izgubljenim paketima.

Pogledajmo sve i s logičke razine (snimljeno s programom **Wireshark**) te pojednostavljeno vidljivo na slici 195.

**Slika 195. Multicast komunikacija: razmjena mrežnih paketa.**




Kao što je vidljivo, prva poruka (**1**) odgovara već objašnjenoj prvoj *multicast* poruci nakon koje slijedi druga poruka (**2**) koja je *IGMP v3* poruka, a koju generira program `omping` i s kojom se prijavljujemo preklopniku (*switchu*) na *multicast* adresu `234.1.2.3`. Naime u ovoj poruci, pošto se radi o *IGMP v.3* poruka se šalje na *multicast* IP adresu: `224.0.0.22` koju primaju i na koju odgovaraju samo uređaji koji podržavaju *IGMP* protokol inačice 3, a u našem slučaju se radi naravno u preklopniku (*switchu*). Vrsta *IGMP* poruke je: `JOIN Group: 234.1.2.3`.

Nakon ove poruke kreće *UDP pinganje*, prvo na *unicast* IP (`192.168.1.149`), a potom na *multicast* IP adresu `234.1.2.3`. I na kraju kada smo zatražili gašenje programa `omping` on nas odjavljuje s *multicast* IP adrese `234.1.2.3` slanjem *IGMP v.3.* poruke prema *IGMP v.3.* uređaju (preklopniku). S time kako se sada šalje *IGMP* poruka: `LEAVE Group: 234.1.2.3`.

Kasnije nakon testiranja dostupnosti multicast IP adrese uključit ćemo zaštitu na odgovaranje na IPv4 *multicast*, ponovno:

```
sysctl net.ipv4.icmp_echo_ignore_broadcasts=1
```

Dakle ovo računalo više neće odgovarati na *ping* (*ICMP*) za IPv4 protokol, na *multicast* adrese.



**IGMP Querier** može biti i preklopnik koji podržava ovu funkcionalnost, mada je to obično usmjerivač (*router*). U posebnim situacijama u kojima koristimo Linux poslužitelj koji koristi Linux *bridge* mrežno sučelje, moguće je konfigurirati i Linux da preuzme funkcionalnost **IGMP Querier**a. Ako imamo čisto Linux *bridge* mrežno sučelje (ne *Open vSwitch*) primjerice imena: `vmbr0`, tada za njega možemo uključiti **IGMP Querier** postavljanjem (*/sys/*) varijable: **`multicast_querier`** na vrijednost **`1`**.

Pogledajmo varijable, a koje ovise o mrežnom sučelju, vezane za *IGMP querier*. Primjer je za *bridge* mrežno *sučelje* `vmbr0`:

| Varijabla ili Sysctl varijabla | /proc **ili** */sys* **lokacija** | **Standardna vrijednost** | **Opis** |
|---|---|---|---|
| **`multicast_querier`** | /sys/devices/virtual/net/**vmbr0**/bridge/**multicast_querier** | 0 | Uključeno ili isključeno (0/1). |
| **net.ipv4.conf.vmbr0.force_igmp_version** | /proc/sys/net/ipv4/conf/**vmbr0**/**force_igmp_version** | 0 | Forsiranje IGMP inačice (1,2,3). |
| | /proc/sys/net/ipv4/conf/default/**force_igmp_version** | 0 | Forsiranje IGMP inačice za sva *mrežna sučelja (1,2,3)*. |
| **`multicast_querier_interval`** | /sys/devices/virtual/net/**vmbr0**/bridge/ **multicast_querier_interval** | 25496 ds (2549.6 sekundi) | Vrijednost u decisekundama (desetinkama) između zadnjeg *multicast host membership query* zaprimljenog kako bi se osiguralo da je *host* aktivan u *multicast* grupi. |

Moguća je i optimizacija *IGMPa* u Linuxu, stoga pogledajmo samo neke od parametara koji su ovdje dostupni:

| Varijabla ili Sysctl varijabla | /proc **ili** */sys* **lokacija** | **Stand. vrij.** | **Opis** |
|---|---|---|---|
| **`net.ipv4.conf.vmbr0.force_igmp_version`** | /proc/sys/net/ipv4/conf/**vmbr0**/**force_igmp_version** | 0 | Forsiranje inačice IGMP-a (1/2/3) |
| | /proc/sys/net/ipv4/conf/default/**force_igmp_version** | 0 | Vrijedi za sva mrežna sučelja. |
| **`multicast_query_interval`** | /sys/devices/virtual/net/**vmbr0**/bridge/ **multicast_query_interval** | 12498ds (1249.8 sekundi) | Postavlja vrijeme, u decisekundama, između poruka upita poslanih od strane *bridge*a kako bi se osigurala valjanost članova koji pripadaju u više *multicast* grupa. |
| **`multicast_query_response_interval`** | /sys/devices/virtual/net/**vmbr0**/bridge/**multicast_query_response_interval** | 999ds (99.9 sekundi) | Duljina vremena, u decisekundama, unutar kojega je računalu dopušteno da odgovori na upit nakon što je poslan. Mora biti manja ili jednaka vrijednosti `multicast_query_interval` |

Sve navedene parametre, moguće je mijenjati i s Linux naredbom `brctl`, `bridge` ili s novijom naredbom `ip` koje su namijenjene za rad s *bridge* funkcionalnostima, pa tako imao cijeli nîz parametara vezanih za *multicast*.



Vezano za ***bridge*** mrežno sučelje, pogledajte poglavlje:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**

**Dodatni izvori informacija: (642),(697),(698),(699),(700),(K-6),(K-10),** **`man sysctl`**,**`man brctl`**,**`man ip`**.

## 22.4. Anycast komunikacija

***Anycast*** je mrežna metodologija adresiranja i usmjeravanja u kojoj jedna odredišna adresa ima više putanja usmjeravanja do dva ili više odredišta krajnjih točaka. Usmjerivači (*routeri*) će odabrati željeni put na temelju broja usmjerivača u prolazu, udaljenosti, najniže „*cijene*“ odnosno koeficijenta, mjerenja kašnjenja ili na temelju najmanje zagušene rute. ***Anycast*** mreže široko se koriste kod proizvoda odnosno usluga s mrežama za isporuku sadržaja (**CDN**) [Engl. *Content delivery network*] kako bi svoj sadržaj što više (geografski) približili krajnjem korisniku.

***Anycast*** adresiranje asocijacija je vrste: jedan-na-jedan-od-mnogih, gdje se *datagrami* (mrežni paketi) usmjeravaju na bilo kojeg člana skupine potencijalnih prijamnika koji su svi identificirani istom odredišnom IP adresom. Algoritam usmjeravanja odabire pojedinačnu odredišnu *anycast* IP adresu, na temelju najmanje skupe metrike usmjeravanja. U praksi to znači da se paketi usmjeravaju do topološki (i često geografski) najbližeg člana bilo koje odredišne *anycast* IP adrese.
*Anycast* je prvi put opisan u RFC1546 te potom u RFC4786.

***DNS sustavi***
Svi korijenski domenski internetski poslužitelji odnosno DNS poslužitelji implementirani su kao klasteri poslužitelja koji koriste ***anycast*** adresiranje. Naime svih 13 korijenskih poslužitelja A–M postoji na više lokacija, s njih 11 na više kontinenata. Pri tome korijenski DNS poslužitelji B i H postoje na dvije lokacije u SAD-u. DNS poslužitelji primarno koriste *anycast* adrese za pružanje decentralizirane usluge. RFC 3258 dokumentira upotrebu *anycast* adresiranja za pružanje autoritativnih DNS usluga. Mnogi komercijalni davatelji DNS usluga također su prešli na *IP anycast* okruženje kako bi povećali performanse upita i redundantnost te implementirali balansiranje opterećenja.

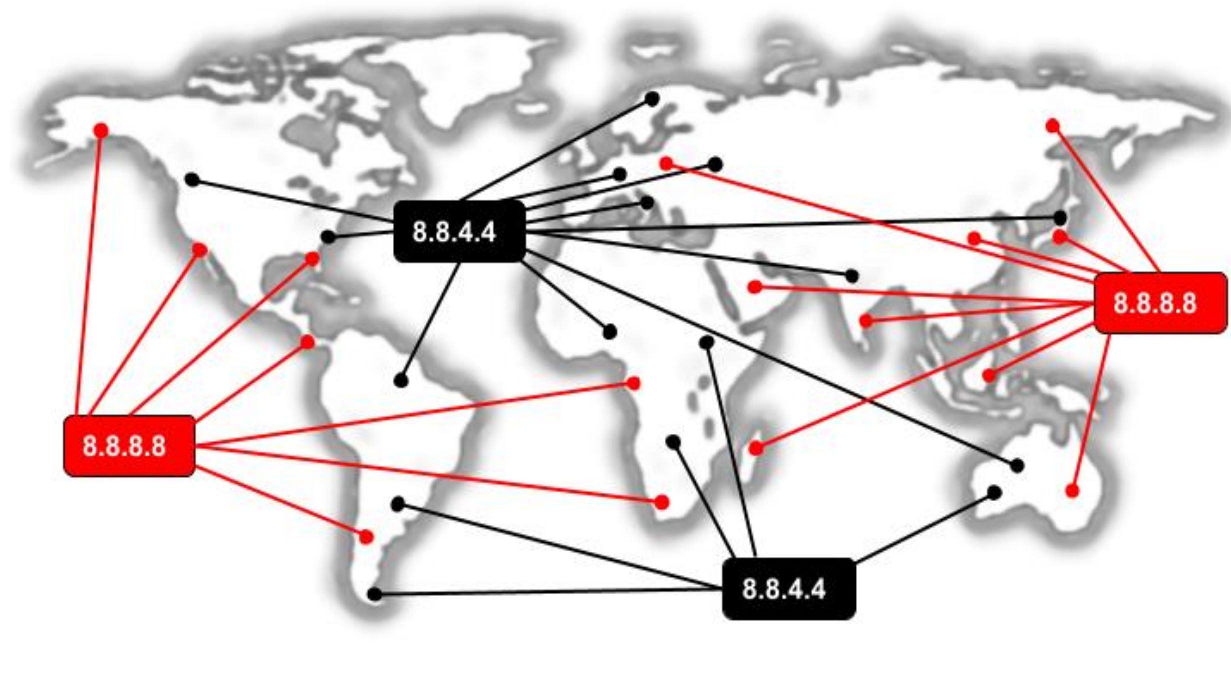


Vezano za **DNS** protokol i detalje oko njega, pogledajte primjere u poglavljima:
**25.8.5. DNS protokol.**
**25.8.5.4. Vrste DNS upita..**

Dakle ***anycast*** se koristi za (javne) ***DNS*** poslužitelje. Primjerice kada DNS klijenti šalju upite Googleovom[957] javnom DNS-u, oni se preusmjeravaju na geografski i topološki najbliže odredište, koristeći bilo koju *anycast* adresu koja se koristi; primjerice 8.8.8.8, 8.8.4.4 ili neku od IPv6 adresa.
Međutim stvarni odabrani fizički DNS poslužitelj na koji će se DNS klijent spojiti, a koji oglašava određenu **anycast** adresu, može se mijenjati zbog mrežnih uvjeta i prometnog opterećenja, kako bi DNS klijenti dobili odgovor s najbližeg DNS poslužitelja, kako geo lokacijski, tako i s najmanjim kašnjenjem (latencijom).

Pogledajmo sliku 195.A. koja nam logički opisuje način raspodjele *anycast* adresa (njihove geo lokacije na slici su simbolične).

**Slika 195.A. Anycast komunikacija.**



## CDN sustavi

Mreže za isporuku sadržaja (engl. *content delivery networks*) mogu koristiti ***anycast*** za trenutne HTTP veze sa svojim distribucijskim centrima ili za DNS protokol. Budući da većina HTTP veza s takvim mrežama zahtijeva statički sadržaj kao što su slike, dokumenti, multimedija i slično, one su općenito kratkog vijeka i bez održavanja stanja tijekom novih TCP sesija.
Opća stabilnost ruta i statusa veza čini *anycast* prikladnim za ovakvu primjenu, iako se koristi TCP protokol.
Naime mreža za isporuku sadržaja ili mreža za distribuciju sadržaja (***CDN***) zemljopisno je distribuirana mreža posredničkih (engl. *proxy*) poslužitelja i njihovih podatkovnih centara. Cilj CDN-a je osigurati visoku dostupnost i performanse distribucijom usluge prostorno (geografski) u odnosu na krajnje korisnike. CDN-ovi su nastali kasnih 1990-ih kao sredstvo za ublažavanje uskih grla interneta jer je Internet počeo postajati kritičan medij za pojedince i tvrtke. Od tada su CDN-ovi narasli do te mjere da služe velikom dijelu internetskog sadržaja danas, uključujući web objekte (tekst, grafiku i skripte), objekte za preuzimanje (medijske datoteke, softver, dokumenti), aplikacije (e-trgovina, portali), live streaming mediji, streaming mediji na zahtjev i stranice društvenih medija.

***Anycast*** se na internetu primarno može implementirati pomoću *Border Gateway Protokola* (***BGP***). Naime višestrukim odredišnim računalima (poslužiteljima), obično u različitim zemljopisnim područjima, dodjeljuju se iste IP anycast adrese, (obično) sekundarnog *loopback* sučelja, a različite rute do tih adresa najavljuju se putem **BGP**-a preko primarnog mrežnog sučelja odnosno njegove javne IP adrese.

Usmjerivači smatraju da su to alternativni pravci do istog odredišta, iako su zapravo rute do različitih odredišta s istom IP adresom. *Anycast* se obično koristi s transportnim protokolima koji ne prate stanje komunikacijske veze, pa se temelje na UDP-u, kao načinu da se osigura visoka dostupnost usluge i uravnoteženje opterećenja. Jedna od poznatijih primjena IPv4 *anycast*-a je navedeni domenski sustav (DNS).

Ovo je vrlo pogodno za bilo koju komunikaciju ove vrste jer je to usluga utemeljena na UDP-u koja pruža pristup podacima o imenima domena bez održavanja komunikacijskog (pouzdanog) kanala poput TCP-a, koja se replicira na višestruke, geografski raspršene poslužitelje, što nam daje visoku dostupnost i skalabilnost DNS servisa. Anycast se osim na internetu, može koristiti i u lokalnim mrežama, kako bi se dobila visoka dostupnost i skalabilnost željenih servisa.

Njegova primjena nije ograničena samo na DNS već i na druge protokole. U manjim i lokalnim mrežama za *anycast* se osim kompleksnog ***BGP-a***, mogu koristiti i drugi protokoli za usmjeravanje koji podržavaju ovakav način rada; poput:

- ***RIPv.2***[958] protokola za usmjeravanje.
- ***OSPF***[959] protokola za usmjeravanje.
- i drugih protokola za usmjeravanje.

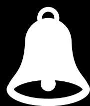
Vezano za anycast, pojednostavljeno možemo reći sljedeće:



*Anycast* je napredna tehnika umrežavanja koja omogućava da više računala na mreži dijeli istu (jedinstvenu) IP adresu. Ovisno o lokaciji (zemljopisnoj) korisničkog zahtjeva, usmjerivači ga šalju onom računalu (poslužitelju) koji se nalazi najbliži korisniku koji je inicirao zahtjev.



Vezano za **DNS** protokol i detalje oko njega, pogledajte primjere u poglavljima:
**25.8.5. DNS protokol.**
**25.8.5.4. Vrste DNS upita.**



***Anycast komunikacija je moguća upotrebom UDP, ali ne i TCP protokola (o njima kasnije)!***

**Izvori informacija: (954),(955),(956),(957), (958),(959),(K-6), (K-10), RFC1546, RFC 3258, RFC4786.**

# 23. Internet (IP) sloj (OSI 3)

**Slijedi napredno poglavlje (23.x)!**
U ovom poglavlju detaljnije ćemo obraditi OSI sloj tri odnosno **IP** protokol koji je definiran u standardu **RFC 791**.
**IP** (*Internet Protocol*) je zadužen za adresiranje krajnjih točaka u komunikaciji (računala/mrežnih uređaja,…) te ugniježđenje (*enkapsulaciju*) podataka s viših slojeva poput *TCP* ili *UDP* u *datagrame** odnosno ono što nazivamo mrežnim paketima.
A potom za slanje tih paketa na mrežu, od izvorišta do odredišta, eventualno preko više IP mreža što je također važna funkcionalnost IP protokola. Zbog svega navedenog *IP* protokol mora imati definiran format paketa unutar kojega postoje svi elementi koji su potrebni za ovakav rad. Svaki *TCP* ili *UDP datagram** se sastoji od dvije logičke cjeline:

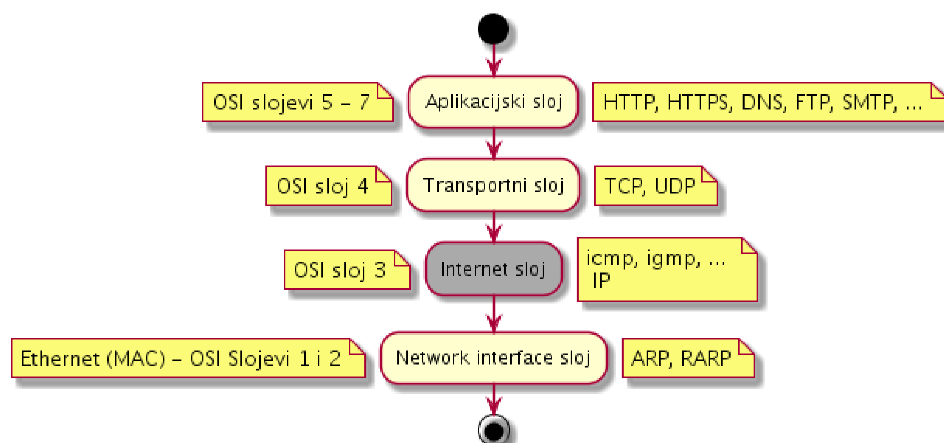
- IP zaglavlja koje sadrži izvorišnu i odredišnu IP adresu, ali i druga polja važna za dostavljanje *datagrama**.
- *Takozvanog payload* dijela u kojemu se nalaze podaci koji se šalju, a oni sadrže sve iz svih viših slojeva; primjerice *TCP* ili *UDP* transportnog sloja, a u konačnici i aplikacijske (korisne) podatke.

Naime dizajn *IP* skupa protokola (*TCP/IP*) radi prema "*end-to-end*" principu, odnosno osiguravanju komunikacije između dvije krajnje točke u komunikaciji. Činjenica je kako se cijela mrežna infrastruktura može smatrati nepouzdanom, počevši od svakog mrežnog elementa, preko njihovih međuveza odnosno sâmih medija preko kojih su povezani. Dodatni faktor koji se uzeo u obzir pri dizajnu je i činjenica da je mreža dinamična i stalno se mijenja. Dakle u bilo kojem trenutku je moguć ispad nekog mrežnog elementa (uređaja) ili međuveze, a tada bi drugi mrežni element morao preuzeti usmjeravanje mrežnih paketa, pri čemu se put do odredišta mijenja. Činjenica je i to kako ne postoji centralno mjesto za praćenje performansi mrežnih elemenata ili stanja međuveza između njih, na osnovu kojih bi se moglo pratiti i održavati stanje cijele (globalne) mreže, jer bi to velikim rastom mreža postalo nemoguće. Stoga je zaključeno kako se sva logika i mehanizmi moraju ugraditi u sâm protokol (sada govorimo o *IP* dijelu) i to u komunikaciji krajnjih točaka komunikacije. Ne zaboravimo kako su se ideje i principi koji su utjecali na razvoj TCP/IP odnosno IP dijela protokola razvijali ranih 1970-tih godina, ali su se u obzir uzeli svi mogući uvjeti na mreži, kao i njeno veliko širenje, koje se i dogodilo razvojem interneta.
Kao posljedica ovakvog dizajna vezanih za navedene pretpostavke, Internet protokol (*IP*) jedino nudi dostavu paketa, prema principu "*best-effort*" odnosno najbolje što se može, a što čini komunikaciju potencijalno nepouzdanom. To znači kako se sâm *IP* protokol smatra protokolom koji se ne brine o sâmoj konekciji i urednom dostavljanju paketa na odredište. Odnosno on se naziva tzv. *connectionless* protokolom. To znači kako može doći do:

- Korupcije (neispravnosti) paketa odnosno podataka unutar paketa.
- Gubitka podataka (paketa).
- Duplikacije primitka paketa ili zaprimanja paketa (podataka) krivim redoslijedom.

Spominjali smo kako su mreže dinamične i stalno im se može mijenjati put kojim podaci prolaze: zbog ispada mrežnih uređaja, međuveza ili slično. Pri tome podaci putuju u paketima, pri čemu se svaki paket obrađuje i šalje, ali i prima zasebno.
Zbog ove dinamične prirode ovakvih veza, događa se da paketi mogu doći do odredišta kroz različite puteve i sâmim time u krivom redoslijedu. Kako bi se riješili ovi problemi oni prvo moraju biti otkriveni, a potom riješeni sâmo i isključivo od krajnjih strana uključenih u komunikaciju. Dakle ovaj posao u potpunosti moraju odraditi krajnja računala koja komuniciraju, odnosno pošiljatelj i primatelj. Pošto *IP* protokol nije namijenjen za detekciju ili rješavanje ovih problema, o tome se brinu transportni protokoli više razine, poput ***TCP*** ili ***UDP*** protokola. Pogledajmo pojednostavljenu sliku (196) ugniježđenja (*enkapsulacije*).

**Slika 196. OSI i TCP/IP modeli.**




Jedina sigurnost koju **IP** sloj nudi je provjera integriteta IP zaglavlja sa svim poljima u njemu. Naime za svaki paket na *IP* sloju se izračunava *provjerni* zbroj (*checksum*), ali ne za podatke (*payload*) već sâmo za sva polja u *IP* zaglavlju. Svaki usmjerivač koji zaprimi svaki pojedini paket, ponovno izračunava *provjerni* zbroj te, ako je neispravan, paket se odbacuje bez slanja upozorenja pošiljaocu. Možemo reći kako Internet protokol (*IP*), a pri tome govorimo samo o ***IP*** sloju:

- Definira *IP* adresnu shemu odnosno adresiranje s *IP* adresama.
- Brine se o životnom vijeku paketa pomoću takozvanog *TTL (Time to Live)* polja.
- Omogućava komunikaciju između krajnjih točaka, koja je nesigurna, bez praćenja njenog stanja (*connectionless*).
- Brine se o fragmentiranju (s jedne strane) i ponovnom sastavljanju fragmenata paketa (s druge strane).
- *Rutabilan* je što znači kako može prolaziti kroz usmjerivače (*routere)* odnosno kroz više povezanih IP mreža.

***datagram** *je naziv za mrežni paket na OSI sloju tri (OSI 3)*

**Izvori informacija: (766), (1044), (K-6), (K-10), `man 7 ip`, RFC 791.**

# 23.1. Oblik IP poruke (paketa)

Pogledajmo strukturu IP paketa. Naime **IP** paket sastoji se od sljedećih dijelova, kako je vidljivo u tablici:

| IP v4 Zaglavlje | | | | | |
|---|---|---|---|---|---|
| **Version** | **IHL** | **DSCP (ToS)** | **ECN** | **Total Length** | |
| Identification | | | | Flags | Fragment Offset |
| Time to Live | Protocol | | | Header Checksum | |
| Source IP Address | | | | | |
| Destination IP Address | | | | | |
| Options | | | | | |
| DATA | | | | | |

Slijedi opis polja unutar IP paketa:

- `Version` – označava inačicu protokola: za IPv4 je to 4 dok je za IPv6 to 6.
- `Internet Header Length (`**`IHL`**`)` – ovo je polje od 4 bita, koji označavaju broj 32 bitnih riječi (*IHL* x 32 bita). U njemu se zapisuje veličina cijelog IP zaglavlja bez *payload* dijela. Ova veličina ovisi i o broju opcija koje su uključene (pogledajte malo niže, pod polje *Options*).
- `Differentiated Services Code Point (`**`DSCP`**`)` – izvorno se koristilo kao *Type Of Service* (**ToS**). Danas je definirano prema RFC 2474 a potom RFC 3168 te nadograđeno sa RFC 3260 kao *Differentiated services* (**DiffServ**) za potrebe aplikacija koje traže prijenos podataka u stvarnom vremenu, poput *VoIP* komunikacije.


Za detalje pogledajte poglavlje: **23.1.1. Osiguranje kvalitete (*QoS*) i klasifikacija prometa (*ToS i DSCP*).**

- `Explicit Congestion Notification (`**`ECN`**`)` – polje je definirano u RFC 3168 i koristi se za notifikaciju zagušenja (*network congetion*). Ovo polje je opcionalno i može se koristiti samo, ako obje strane podržavaju ovu (***ECN***) funkcionalnost.


Za detalje pogledajte poglavlje: **24.2.9.3. Explicit Congestion Notification (*ECN*).**

- `Total Length` – je 16 bitno polje u koje se zapisuje veličina cijelog paketa; i *IP* zaglavlja i podataka (*payload*). To znači da IP paket može biti maksimalne veličine 65.535 bajta (64 kilobajta) [**$2^{16}$**].
- `Identification` – je polje za identifikaciju fragmenata jednog *datagrama*, u slučaju da se paket fragmentira na manje dijelove (fragmente). Definirano je u **RFC 6864.**
- `Flags` polje od tri (3) bita sa zastavicama, od kojih:
  - `bit0` – rezerviran je i mora biti **0**
  - `bit1` – označava: *Don't Fragment* (**DF**), a ovaj bit može biti u dva stanja:
    - `0` – Ova opcija je isključena (standardno), pa je fragmentacija omogućena.
    - `1` – Ako se izričito zabranjuje fragmentiranje *datagrama* onda je ova vrijednost **1**. Nadalje, ako je ova zastavica uključena, a fragmentacija se zahtjeva zbog usmjeravanja, tada će ovakav paket biti odbačen. Dakle ako je uključena ova zastavica, a šalje se paket veći od ***MTUa*** koji može podnijeti mreža, pa bi se morao fragmentirati (razlomiti na više manjih paketa), paket će biti odbačen. Ovo se može koristiti za *ICMP* u slučaju testiranja maksimalne veličine ***MTUa***, kada će paketi koji prelaze određenu veličinu (***MTU***) biti odbačeni, te se može utvrditi koji od njih je zadnji uspješno prošao (s kojim maksimalnim ***MTU-om***). Ovakav rad se koristi i za *Path MTU Discovery* protokol.
  - `bit2` – označava: *More Fragments* (**MF**), a ovaj bit može biti:
    - `0` – Zadnji fragment ima postavljenu vrijednost **0**, što znači kako više nema fragmenata. Za ne fragmentirane pakete ova zastavica je stalno isključena odnosno također u vrijednosti **0**.
    - `1` – Za fragmentirane pakete ova vrijednost je postavljena u **1** ako slijedi još fragmenata.


Za detalje pogledajte poglavlje: **23.2. IP fragmentacija.**

- `Fragment Offset` – je veličine 13 bitova, a pokazuje vrijednost 8 *bajtnih* blokova. Ovo polje specificira pomak odnosno *offset* pojedinog fragmenta (razlomljenog paketa) od početka originalnog ne fragmentiranog datagrama. Prvi fragment uvijek ima vrijednost *offseta* 0. To znači kako se može označiti pomak do: $(2^{13} – 1) \times 8 = 65{,}528$ *bajta*. Računajući od nule (početak pomaka fragmenta) to znači kako bi maksimalna veličina paketa koji bi se spojio od svih fragmenata, bila 65,528 *bajta* što je i u teoriji više nego je moguća veličina najvećeg paketa (65,535 *bajta*) jer uz to sve ide i zaglavlje od 20 *bajta* pa je to onda : 65,528 + 20 = 65,548 *bajta*.

- `Time To Live (TTL)` – je 8 bitna vrijednost koja označava vrijeme života paketa. Nakon što ovo vrijeme paketa na mreži istekne, zadnji usmjerivač koji ga zaprimi, odbaciti će ga. Ovo je zaštitni mehanizam od vječnog lutanja paketa po mreži. U praksi se koristi Tzv. *hop count* odnosno odbrojavanje pređenih usmjerivača. Pri tome se prvo postavlja inicijalna vrijednost (obično 64), a prolaskom kroz svaki sljedeći usmjerivač smanjuje se za jedan. Kada se dođe do vrijednosti jedan (1), taj usmjerivač kod kojega dođe, odbaci ga te obično na izvorište šalje poruku o grešci.


Za detalje pogledajte poglavlje: **23.4.1. Time to live (*TTL*).**

- `Protocol` – ovdje se zapisuje koji je vršni protokol zadužen za ovaj paket. Dakle ovdje se definira koji viši (*transportni*) protokol će se koristiti za prijenos. Primjerice: **TCP** ili **UDP** ili neki drugi specifični transportni protokol. Popis ovih vršnih protokola je definiran u **RFC 790** i vidljiv u tablici dolje, te na izvoru informacija **(1452)**.
- `Header Checksum` – je 16 bitna vrijednost *provjernog* zbroja (*checksum*), koja se izračunava samo za IP zaglavlje, kako govori i ime. Svaki paket koji dođe do bilo kojeg usmjerivača, prvo se od strane usmjerivača ponovno izračunava provjerni zbroj, te ako nije točan paket se odbacuje. Ako je paket ispravan, usmjerivač prvo smanjuje *TTL* vrijednost za jedan (1), upisuje ga u ovaj (sada novi) paket, te ponovno izračunava `Header Checksum`, a tek potom paket prosljeđuje dalje (pročitajte **RFC 791**).
- `Source address` – je izvorišna IP adresa; onoga tko šalje paket.
- `Destination address` – je odredišna IP adresa; onoga tko prima paket.
- `Options` – ovo su opcije (koje se vrlo često ne koriste). Prije nego se ove opcije uopće mogu koristiti u *IHL* polju je potrebno alocirati dovoljno 32 *bitnih* polja (riječi) kako bi se opcije mogle zapisati.
  Dakle moguće je dodavati više opcija u nizu, a nakon zadnje mora slijediti *End of Options List (0x00)*.
- `Data` – ovaj dio sadrži *payload* odnosno sve s viših OSI slojeva. Dakle sadrži sve s **TCP** ili **UDP** sloja kao i podatke.

Pod poljem `Protocol` definiraju se vršni (*transportni*) protokoli, stoga pogledajmo samo neke od njih:

| Broj protokola | Protokol |
|---|---|
| 1 | *ICMP (Internet Control Message Protocol) za IPv4* |
| 2 | *IGMP (Internet Group Management Protocol)* |
| 4 | *IP in IP (enkapsulacija)* |
| 6 | ***TCP** (Transmission Control Protocol)* |
| 8 | *EGP (Exterior Gateway Protocol)* |
| 9 | *IGP (Interior Gateway Protocol)* |
| 17 | ***UDP** (User Datagram Protocol)* |
| 27 | *Reliable Data Protocol* |
| 47 | *GRE (Generic Routing Encapsulation)* |
| 58 | *ICMP za IPv6* |
| 88 | *EIGRP* |
| 89 | *OSPF (Open Shortest Path First)* |
| 103 | *PIM (Protocol Independent Multicast)* |
| 112 | *VRRP (Virtual Router Redundancy Protocol), CARP (Common Address Redundancy Protocol)* |
| 115 | *L2TP (Layer Two Tunneling Protocol Version 3)* |
| 124 | *IS-IS over IPv4 (Intermediate System to Intermediate System (IS-IS))* |
| 132 | ***SCTP** (Stream Control Transmission Protocol)* |
| 133 | *FC (Fibre Channel)* |
| 136 | *UDP lite (Lightweight User Datagram Protocol)* |
| 137 | *Multiprotocol Label Switching Encapsulated in IP* |

Format poja s opcijama (`Options`) se sastoji od tri dijela i često je kombinacija više polja, kako je vidljivo u tablici:

<table>
<tr><th colspan="3">Option Type</th><th>Option Lenght</th><th>Option Data</th></tr>
<tr><td>Flags</td><td>Class</td><td>Number</td><td>Option Length</td><td>Option Data</td></tr>
</table>

Pogledajte i njihov opis:

- `Option Type`:
  - `Flags` ili *Copied Flag*
    - `0` – indicira kako se opcija koja slijedi **NE** treba kopirati u svaki mrežni fragment.
    - `1` – indicira kako se opcija koja slijedi **MORA** kopirati u svaki mrežni fragment.
  - `Class`
    - `0` – je standardno postavljena (*Control*).
    - `2` – koristi se za mjerenje (za opciju *Internet Timestamp*) ili za *debugiranje*.
  - `Number`
    - `0` – indicira kako se radi o kraju opcija (*end of the option list*).
    - `1` – indicira kako nema opcija (*No Operation*).
    - `2` – indicira kako se radi o sigurnosnim opcijama, definiranim u **RFC 791.**
    - `3` – indicira opciju *Loose Source Routing*.
    - `4` – indicira opciju *Internet Timestamp*.
    - `7` – indicira opciju *Record Route*, koja ako je uključena, svaki usmjerivač kroz koji prođe mreži paket zapisuje svoju IP adresu. Ovu funkcionalnost koriste programi `traceroute` (standardno) ili `ping` (ako mu se naredi) za praćenje kroz koje sve usmjerivače prolazi paket do odredišta.
    - `8` – označava *stream ID*.
    - `9` – indicira opciju *Strict Source Routing* kod kojega pošiljatelj mrežnog paketa naređuje usmjerivačima kojim točnim putem paket mora prolaziti do odredišta. Dakle navodi se lista usmjerivača kroz koje paket mora proći, ne bi li došao do odredišta. Slijedi cijeli niz brojeva.
- `Option Lenght` - označava veličinu polja opcija.
- `Option-Data` - ovo je dio u kojem se nalaze podaci od opcija.

Opcije navedene u tablici su vidljive i brojčano; navodimo samo one često korištene.

Sve opcije su definirane u: **https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml.**

Pogledajmo sâmo neke od njih:

| Broj opcije | Opcija | Opisano u standardu | | Broj opcije | Opcija | Opisano u standardu |
|---|---|---|---|---|---|---|
| **0** | *End of Options List* | RFC 791 | | **11** | *MTU Probe* | RFC 1063 i RFC 1191 |
| **1** | *No Operation* | RFC 791 | | **12** | *MTU Reply* | RFC 1063 i RFC 1191 |
| **2** | *Security* | RFC 1108 | | **25** | *Quick start* | RFC 4782 |
| **3** | *Loose Source Route (**LSR**)* | RFC 791 | | **82** | *Traceroute* | RFC 1393 i RFC 6814 |
| **4** | *Time Stamp* | RFC 791 | | **137** | *Strict Source Route (**SSR**)* | RFC 791 |
| **7** | *Record Route* | RFC 791 | | **147** | *Address extension* | RFC 6814 |
| **8** | *Stream ID* | RFC 791 i RFC 6814 | | **148** | *Router Alert* | RFC 2113 |



***Cijelo IP zaglavlje je minimalne veličine 20 bajta.***

## Source route značajka

*Source route* (**SSR**) značajka je IP protokola koja omogućuje pošiljatelju paketa da odredi kojom rutom bi paket trebao ići na putu do odredišta (i na povratku). Ona je izvorno osmišljena za korištenje kada računalo (ili mrežni uređaj) nema odgovarajuće zadane rute u svojoj tablici usmjeravanja. Međutim njegova upotreba potencijalni je sigurnosni problem, stoga je najbolje onemogućiti ovu značajku, ako nije potrebna. Za prihvaćanje paketa sa **Source route** opcijom (**SSR** [*Strict Source Route*] opcija) može se koristiti *sysctl* varijabla: `net.ipv4.conf.all.accept_source_route` (moguća je definicija i za pojedina mrežna sučelja). Druga opcija je prihvaćanje paketa s lokalnim izvornim adresama. U kombinaciji sa prikladnim usmjeravanjem, može se koristiti za usmjeravanje paketa između dva lokalna mrežna sučelja. Ova opcija je dostupna kao *sysctl* varijabla: `net.ipv4.conf.all.route_localnet` (i kod nje je moguća definicija za svako mrežno sučelje).

## Aplikacijski protokoli i IP protokol

U velikoj većini slučajeva **IP** (*Internet protokol*) samo je jedna od komponenti i slojeva unutar TCP/IP skupa protokola, jer se na njega nadovezuju *transportni* protokoli poput **TCP** ili **UDP** protokola, koje potom koriste aplikacijski protokoli, poput: ***HTTP***, ***HTTPS***, ***DNS***; ***TELNET***, ***SSH***; ***LDAP***, ***SMTP***, ***SNMP*** i drugih, za prijenos aplikacijskih podataka.
Međutim, postoji i nekoliko protokola koji rade samo na **IP** sloju te ne trebaju **TCP** ili **UDP** protokole za transport.
Takvi protokoli su primjerice:

- **ICMP** *(Internet Control Message Protocol) i* **IGMP** *(Internet Group Management Protocol).*
- **NDP** (*Neighbor Discovery Protocol*) i **IPSec** *(Internet Protocol Security)* i još samo nekolicina protokola za posebne namjene.



*Definicija viših (transportnih) protokola koji će se koristiti iznad **IP** protokola definira se u polju:* **`Protocol`**.

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na IP sloju. Primjerice, ako nas zanima brojač (*metrika*) imena `IpInReceives`, možemo ju vidjeti sa:

```
nstat -az IpInReceives
```

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za IP sloj:

| Naziv brojača | Opis |
|---|---|
| **`IpInReceives`** | Broj paketa koje je primio IP sloj. Povećava se na početku funkcije **ip_rcv**, a uvijek se ažurira zajedno s `IpExtInOctets`. Povećat će se čak i ako se paket kasnije odbaci (npr. jer je IP zaglavlje nevažeće ili je kontrolni zbroj pogrešan i tako dalje). Označava i broj agregiranih segmenata nakon **GRO** ili **LRO**. (**RFC 1213**) |
| **`IpInDelivers`** | Broj paketa isporučenih protokolima višeg sloja. npr. TCP, UDP, ICMP i tako dalje. Ako nitko ne sluša na toj mrežnoj utičnici (*socketu*), isporučit će se samo protokoli podržani kernelom, ako netko sluša na *socketu*, isporučit će se svi važeći IP paketi. (**RFC 1213**) |
| **`IpOutRequests`** | Broj paketa poslanih putem IP sloja, i za pojedinačne i za višestruke pakete, uvijek se ažurira zajedno s `IpExtOutOctets`. |
| **`IpExtInNoECTPkts`**<br>**`IpExtInECT1Pkts`**<br>**`IpExtInECT0Pkts`**<br>**`IpExtInCEPkts`** | Ova četiri brojača izračunavaju koliko je paketa primljeno po ECN statusu. Broji se pravi broj paketa bez obzira na **LRO/GRO**. Dakle, za isti paket možete otkriti da `IpInReceives` broji 1, ali `IpExtInNoECTPkts` broji 2 ili više. |
| **`IpInHdrErrors`** | Označava da je paket odbačen zbog pogreške IP zaglavlja. To se može dogoditi i u dolazu ili izlazu paketa. (**RFC 1213**) |
| **`IpInAddrErrors`** | Povećat će se u dva scenarija: (1) IP adresa nije važeća. (2) Odredišna IP adresa nije lokalna adresa i IP prosljeđivanje (*forwarding*) nije omogućeno. (**RFC 1213**) |
| **`IpExtInNoRoutes`** | Ovaj brojač znači da je paket odbačen kada IP stog primi paket i ne može pronaći rutu za njega iz tablice ruta. To se može dogoditi kada je IP prosljeđivanje omogućeno, a odredišna IP adresa nije lokalna adresa i ne postoji ruta za odredišnu IP adresu. |
| **`IpInUnknownProtos`** | Povećat će se ako kernel ne podržava protokol sloja 4. Ako aplikacija koristi mrežnu utičnicu (*socket*), kernel će uvijek isporučiti paket u *socket* i ovaj brojač se neće povećati. |
| **`IpExtInTruncatedPkts`** | Za IPv4 paket to znači da je stvarna veličina podataka manja od polja "**Total Length**" u IPv4 zaglavlju. To može biti znak problema na mreži ili da se neki datagrami gube usput. U tom slučaju možete razmotriti povećanje **MTU** (*Maximum Transmission Unit*) vrijednosti na mrežnom sučelju. |
| **`IpInDiscards`** | Označava da je paket odbačen u IP prijemnoj stazi zbog internih razloga kernela (npr. jer nema dovoljno memorije). (**RFC 1213**) |
| **`IpOutDiscards`** | Označava da je paket odbačen u IP stazi slanja zbog internih razloga kernela. |
| **`IpOutNoRoutes`** | Označava da je paket odbačen u IP stazi slanja i da za njega nije pronađena ruta. |
| **`IpReasmReqds`** | Broj primljenih IP fragmenata koje je trebalo ponovno sastaviti u ovom entitetu. |
| **`IpReasmOKs`** | Broj uspješno ponovno sastavljenih IP *datagrama*. |
| **`IpReasmTimeout`** | Maksimalan broj sekundi u kojima se primljeni fragmenti zadržavaju dok čekaju ponovno sastavljanje u ovom entitetu. |
| **`IpReasmFails`** | Broj kvarova koje je otkrio algoritam ponovnog sastavljanja IP-a (iz bilo kojeg razloga: isteklo vrijeme, pogreške itd.). Imajte na umu da ovo nije nužno broj odbačenih IP fragmenata budući da neki algoritmi (osobito algoritam u RFC 815) mogu izgubiti trag broja fragmenata kombinirajući ih kako su primljeni. |
| **`IpFragOKs`** | Broj IP datagrama koji su uspješno fragmentirani u ovom entitetu. |
| **`IpFragFails`** | Broj IP datagrama koji su odbačeni jer su morali biti fragmentirani u ovom entitetu, ali nisu mogli biti, npr. jer je postavljena njihova zastavica **Dont Fragment** (**`DF`**). |
| **`IpFragCreates`** | Broj fragmenata IP datagrama koji su generirani kao rezultat fragmentacije u ovom entitetu. |
| **`IpForwDatagrams`** | Broj ulaznih datagrama za koje ovaj entitet nije bio konačno IP odredište, zbog čega je pokušano pronaći rutu za njihovo prosljeđivanje na to konačno odredište. Kada paket stigne na mrežni uređaj, a njegova odredišna adresa nije lokalna adresa na tom uređaju, uređaj može odlučiti proslijediti taj paket na drugi uređaj koji je odgovoran za daljnje preusmjeravanje prema odredištu. Ovaj proces se naziva proslijeđivanje (***forwarding***). |

Za: **IpReasmReqds**, **IpReasmOKs**, **IpReasmTimeout**, **IpReasmFails**: razmotriti promjene vrijednosti *sysctl* varijabli: `net.ipv4.ipfrag_max_dist` i `net.ipv4.ipfrag_secret_interval` (povećanje zadnje).

Za: **IpFragOKs**, **IpFragFails**, **IpFragCreates**: razmotriti promjene vrijednosti *sysctl* varijabli: `net.ipv4.ipfrag_high_thresh`, `net.ipv4.ipfrag_low_thresh` i `net.ipv4.ipfrag_time`.

**Izvori informacija: (1044),(1135),(1402),(1452),(1650),(1651),(K-6),(K-10), `man nstat`, RFC 790, RFC 791, RFC 1213, RFC 2474, RFC 3168, RFC 3260, RFC 6864.**

## 23.1.1. Osiguranje kvalitete (*QoS*) i klasifikacija prometa (*ToS i DSCP*)

Kvaliteta usluge to jest **QoS** (engl. *Quality of Service*) odnosi se na opis ili mjerenje ukupne izvedbe određene usluge, kao što su telefonija ili računalna mreža. Za kvantitativno mjerenje kvalitete usluge često se razmatra nekoliko povezanih aspekata mrežne usluge, kao što su gubitak paketa, brzina prijenosa, propusnost, kašnjenje prijenosa, dostupnost serisa ili usluge i slično. U području računalnih mreža i ostalih paketno komutiranih telekomunikacijskih mreža, kvaliteta usluge odnosi se na prioritizaciju prometa i mehanizme kontrole rezervacija resursa, a ne na postignutu kvalitetu usluge.
Kvaliteta usluge je mogućnost pružanja različitih prioriteta različitim aplikacijama, korisnicima ili protoku podataka ili jamčenja određene razine performansi protoka podataka.
Kvaliteta usluge posebno je važna za prijenos mrežnog prometa s posebnim zahtjevima. Kvaliteta usluge je važna za multimedijske aplikacije za prijenos u stvarnom vremenu kao što su protokoli za prijenos glasa preko IP-a (***VoIP***), online igre za više igrača i IPTV, jer one često zahtijevaju fiksnu brzinu prijenosa i osjetljive su na kašnjenje. Kvaliteta usluge posebno je važna i u mrežama gdje je kapacitet mreže ograničen resurs, na primjer u mobilnoj podatkovnoj komunikaciji.
Mreža ili protokol koji podržava **QoS** može dogovoriti parametre klasifikacije i rada s aplikacijskim softverom i rezervirati kapacitet u mrežnim čvorovima, na primjer tijekom faze uspostavljanja sesije. On tijekom sesije može pratiti postignutu razinu performansi, na primjer brzinu prijenosa podataka i kašnjenje, te dinamički kontrolirati prioritete raspoređivanja u mrežnim čvorovima. I na kraju može osloboditi rezervirani kapacitet tijekom faze zatvaranje sesije (veze).



Sumirajmo: kvaliteta usluge (ili skraćeno ***QoS***) odnosi se na skup tehnika koje jamče ili poboljšavaju kvalitetu usluge koja se pruža aplikacijama. Najpopularnija takva tehnika uključuje klasificiranje mrežnog prometa u kategorije te razlikovanje načina rukovanja prometom, prema kategoriji kojoj pripada.

***Svaka aplikacija za sebe može preko socketa s kojim se spaja na mrežni stôg Linuxa postavljati ToS ili DSCP vrijednosti***!

## QoS, ToS i DSCP

Današnje podatkovne mreže nose mnoge različite vrste usluga, uključujući glas, video, glazbu, web stranice, e-poštu i drugo. Mnogi od predloženih mehanizama za osiguranje kvalitete prijenosa (**QoS**) koji su omogućili tim uslugama da koegzistiraju bili su i složeni i nisu se uspjeli prilagoditi zahtjevima javnog interneta.
Naime unutar TCP/IP mrežnog paketa u IP zaglavlju rezerviran je jedan bajt (8 bitova) koji definira koju vrstu usluge (i prioritet) paket eventualno sadrži. **ToS** (engl. *Type of Service*) bajt unutar IP zaglavlja može se koristiti za određivanje prioriteta paketa unutar mreže. Ovo polje unutar IP paketa je definirano u **RFC 791**. *Type of Service* (ToS) polje unutar IP zaglavlja sastoji se od tri polja, kako je definirano u **RFC 1349**. Međutim u prosincu 1998. IETF je objavio **RFC 2474** odnosno definiciju polja diferenciranih usluga (**DS** polja) u zaglavljima IPv4 i IPv6, a koja je zamijenila polja **IPv4 TOS** i **IP precedence** s jednim poljem **DS**. To je slučaj u većini implementacija.
U polju ***DS*** koristi se raspon od osam vrijednosti (selektori klasa) za kompatibilnost unatrag s prethodnim poljem IP **ToS**. Danas je *DiffServ* (***DSCP***) uvelike zamijenio ***ToS*** i druge ***QoS*** mehanizme na osi sloju tri (OSI 3), kao primarnu arhitekturu koju usmjerivači koriste za pružanje usluge osiguravanja kvalitete usluge globalno znane kao ***QoS*** (engl. *Quality of Service*).

Dakle **Differentiated services** ili **DiffServ** koristi 6-bitnu kodnu točku diferenciranih usluga to jest *DSCP* (engl. *differentiated services code point*) u 8-bitnom polju diferenciranih usluga (**DS** polje) u IP zaglavlju za potrebe klasifikacije paketa.
Možemo reći i ovako: *DiffServ* je mehanizam za upravljanje prometom temeljen na klasama odnosno kategorijama.
Tako primjerice usmjerivači koji podržavaju *DiffServ* odnosno *DSCP* implementiraju način rada koje definiraju svojstva prosljeđivanja paketa povezana s klasom prometa.
Usmjerivači koji podržavaju *DiffServ* (*DSCP*) implementiraju takozvano per-hop ponašanja (***PHB***), koje definira svojstva prosljeđivanja paketa povezana s klasom prometa. Pri tome različite klase mogu se definirati da ponude, na primjer, usluge s malim gubicima ili s niskom latencijom (kašnjenjem). Umjesto razlikovanja mrežnog prometa na temelju zahtjeva pojedinačnog toka, *DiffServ* radi na principu klasifikacije prometa, smještajući svaki paket u jednu od ograničenog broja klasa prometa. Svaki usmjerivač na mreži tada je konfiguriran za razlikovanje prometa na temelju svoje klase. Svakom klasom prometa može se upravljati drugačije, osiguravajući preferencijalni tretman za promet višeg prioriteta na mreži.

Iako *DiffServ* preporučuje standardizirani skup klasa, *DiffServ* arhitektura ne uključuje unaprijed definirane prosudbe o tome kojim vrstama prometa treba dati prioritet. *DiffServ* jednostavno pruža okvir koji omogućuje klasifikaciju i diferencirano postupanje. Mrežni promet koji ulazi u *DiffServ* podliježe klasifikaciji. Klasifikator prometa može pregledati mnogo različitih parametara u dolaznim paketima, kao što su izvorišna adresa, odredišna adresa ili vrsta prometa i dodijeliti pojedinačne pakete određenoj klasi prometa. Klasifikatori prometa mogu poštovati bilo koje *DiffServ* oznake u primljenim paketima ili mogu odlučiti ignorirati ili nadjačati te oznake. Za čvrstu kontrolu nad količinama i vrstom prometa u određenoj klasi, mrežni operater može odlučiti da ne poštuje oznake na ulazu u *DiffServ* domenu.
Naime skupina usmjerivača koji implementiraju zajedničke, administrativno definirane *DiffServ* politike naziva se *DiffServ* domena. Nadalje, promet u svakoj klasi može se dodatno podvrgavati drugim komponentama sustava koji mogu utjecati na njega.



Za detalje, kako rade klasifikatori mrežnog prometa, pogledajte poglavlje:
**25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa.**



**QoS** se odrađuje unutar **queueing** mehanizma (***qdisc***) koji je znan i kao: ***network scheduler***, ***packet scheduler***, ***queueing discipline*** ili ***qdisc***. Pogledajte gore navedeno poglavlje.

U teoriji, moguće je imati do 64 različite klase prometa koristeći 64 dostupne DSCP vrijednosti. DiffServ RFC-ovi preporučuju, ali ne zahtijevaju točno određena kodiranja. To mrežnom operateru daje veliku fleksibilnost u definiranju klasa prometa. U praksi, međutim, većina mreža koristi sljedeća uobičajeno definirana ponašanja:

- Zadano prosljeđivanje to jest *default forwarding* (**DF**) [PHB] — što označava metodu "*najbolje moguće*". Promet koji ne zadovoljava zahtjeve bilo koje druge definirane klase koristi **DF**.
- Ubrzano prosljeđivanje to jest *expedited forwarding*(**EF**) [PHB] — posvećeno prometu s malim gubicima i malim kašnjenjem. **EF** ima karakteristike niskog kašnjenja, gubitaka i općenito male varijacije u radu. Ove su karakteristike prikladne za glasovne, video i druge usluge u stvarnom vremenu. **EF** prometu često se daje strogi (visoki) prioritet čekanja iznad svih ostalih klasa prometa. Definiran je u **RFC 3246**.
- Osigurano prosljeđivanje to jest *assured forwarding* (**AF**) [PGB] — daje jamstvo isporuke pod predodređenim uvjetima. Definiran je u **RFC 2597** i **RFC 3260**. On omogućuje operateru osiguranje isporuke sve dok promet ne premašuje određenu pretplaćenu cijenu. Promet koji prelazi definiranu stopu pretplate suočava se s većom vjerojatnošću da će biti prekinut (odbačen), ako dođe do zagušenja. Grupa ponašanja **AF**-a definira četiri zasebne klase *AF*-a sa svim prometom unutar jedne klase s istim prioritetom. Unutar svake klase, paketima se daje prednost odbacivanja (visoka, srednja ili niska, gdje veća prednost znači više odbacivanja). Kombinacija klasa i prioriteta odbacivanja daje dvanaest odvojenih DSCP kodiranja od AF11 do AF43, kako je vidljivo u tablici:

| | Class 1 | Class 2 | Class 3 | Class 4 |
|---|---|---|---|---|
| **Low drop probability** | **AF11** (DSCP 10)<br>001010 | **AF21** (DSCP 18)<br>010010 | **AF31** (DSCP 26)<br>011010 | **AF41** (DSCP 34)<br>100010 |
| **Med drop probability** | **AF12** (DSCP 12)<br>001100 | **AF22** (DSCP 20)<br>010100 | **AF32** (DSCP 28)<br>011100 | **AF42** (DSCP 36)<br>100100 |
| **High drop probability** | **AF13** (DSCP 14)<br>001110 | **AF23** (DSCP 22)<br>010110 | **AF33** (DSCP 30)<br>011110 | **AF43** (DSCP 38)<br>100110 |

- *Class Selector* **PHB**-ovi — koji održavaju kompatibilnost unatrag s IP poljem "*precedence*" (ToS polje).

**Upotreba ToS (za starije implementacije i naredbe koje ga još koriste)**
Primjerice, `iptables` i neki (stariji) usmjerivači, koriste staru stariju shemu kodiranja prema **ToS**, kako je definirano u **RFC 1349** gdje su bitovi 0 do 2 "*precedence*" bitovi, a 3 do 6 definiraju **ToS** to jest vrstu usluge (4 bita). Pogledajmo tablicu s popisom pet unaprijed definiranih **ToS** pseudonima za postavljanje ***ToS*** bajta.
Osim pseudonima možete koristiti heksadecimalne vrijednosti 0x00-0xFF ili decimalne brojeve između 0-255.
Dakle naredba `iptables` podržava **ToS**, korištenjem sljedećeg prekidača:

```
-j TOS --set-tos ToS-VRIJEDNOST
```

Dok su pseudonimi koje možete koristiti s naredbom `iptables` sljedeći:

| **Ime** ToS **pseudonima (*ToS-VRIJEDNOST*)** | **Decimalno** | **Hexadicimalno** | **Binarno** |
|---|---|---|---|
| Minimize-Delay | 16 | `0x10` | 1000 |
| Maximize-Throughput | 8 | `0x08` | 0100 |
| Maximize-Reliability | 4 | `0x04` | 0010 |
| Minimize-Cost | 2 | `0x02` | 0001 |
| Normal-Service | 0 | `0x00` | 0000 |

Pogledajmo kako pomoću naredbe `iptables` koristiti ***ToS*** za klasifikaciju *SSH* protokola, kao "*Minimize-Delay*" [*0x10*] :

```
iptables -A PREROUTING -t mangle -p tcp --sport ssh -j TOS --set-tos Minimize-Delay
```

Međutim `iptables` podržava i ***DSCP***. Odnosno može postaviti ***DSCP*** polje unutar IP zaglavlja.
To se može postići na dva načina:

```
-j DSCP --set-dscp DSCP-VRIJEDNOST
-j DSCP --set-dscp-class DSCP-VRIJEDNOST
```

Pri tome ***`DSCP-VRIJEDNOST`*** može biti: decimalna, heksadecimalna ili simbolička oznaka (pr. **BE**, **EF** ili bilo koja **CSxx** i **AFxx** klasa). Pogledajmo primjer sličan gornjem, dakle promjenu klase za SSH protokol (gdje je izvorišni port 22/*SSH*):

```
iptables -A PREROUTING -t mangle -p tcp --sport ssh -j DSCP --set-dscp-class AF13
```

Isto je naravno moguće i s novijom naredbom `nft` upotrebom pravila poput: `nft ... ip dscp` ***`DSCP-VRIJEDNOST`***`...`
Pri tome ***`DSCP-VRIJEDNOST`*** može biti: decimalna, heksadecimalna ili simbolička oznaka (pr. **ef** ili bilo koja **CSxx** ili **AFxx** klasa). Pogledajmo jedan primjer sličan onom s naredbom `iptables` sa ***SSH*** protokolom (dolazno/odlazno):

```
nft add table ip mangle
nft add rule ip mangle PREROUTING tcp sport 22 counter ip dscp set 0x04
nft add rule ip mangle PREROUTING tcp dport 22 counter ip dscp set 0x04
```

Pogledajmo kako brzo testirati ***ToS*** (modifikacija ***ToS*** polja u IP zaglavlju) upotrebom naredbe `ping`:(prema IP **10.0.0.2**)
Primjerice za postavljanje **ToS** na vrijednost **2** [**m**oguće je definirati i heksadecimalne oznake (pr. `-Q 0x03`, `-Q 0x02`,]:

```
ping -Q 2 10.0.2.2
```

Pogledajmo i kako snimiti *ICMP* pakete s programom `tcpdump` u ***pcap*** formatu, za kasniju analizu s programom ***Wireshark***.

```
tcpdump -i enp0s3 -s0 -n -vvv  icmp -w tcpdump-ping-ToS.pcap
```

Pogledajmo i kako filtrirati samo *ICMP* pakete koji imaju postavljeno ***ToS*** polje **`0x02`**, decimalno **`2`** (za gornji ***ping*** primjer):

```
tcpdump -i enp0s3 -n -vvv 'icmp and ip[1] & 0x02 == 2'
```

Pogledajmo maksimalno skraćenu tablicu s korisnim ***ToS*** i ***DSCP*** klasifikacijama te nazivima u **iptables** naredbi:

| **DSCP** ime | **DS** hex | **ToS** hex | Iptables **ToS** ime | Naziv klase |
|---|---|---|---|---|
| **DF / CS0** | 0x00 | 0x00 | *Normal-Service* | *Standard* |
| - | – | 0x02 | *Minimize-Cost* | - |
| - | 0x01 | 0x04 | *Maximize-Reliability* | - |
| **LE** | 0x01 | 0x04 | - | *Lower-Effort* |
| - | 0x02 | 0x08 | *Maximize-Throughput* | - |
| | 0x04 | 0x10 | *Minimize-Delay* | - |
| **CS1** | 0x08 | 0x20 | - | *Low-Priority Data* |
| **AF11** | 0x0a | 0x28 | - | *High-Throughput Data* |
| **CS2** | 0x10 | 0x40 | - | *OAM* |
| **AF21** | 0x12 | 0x48 | - | *Low-Latency Data* |
| **CS3** | 0x18 | 0x60 | - | *Broadcast Video* |
| **AF31** | 0x1a | 0x68 | - | *Multimedia Streaming* |
| **CS4** | 0x20 | 0x80 | - | *Real-Time Interactive* |
| **AF41** | 0x22 | 0x88 | - | *Multimedia Conferencing* |
| **EF** | 0x2e | 0xb8 | - | *Telephony* |
| **CS6** | 0x30 | 0xc0 | - | *Network Routing Control* |

Nazivi klasa usluga definirani su u: **RFC 4594**, **RFC 5865** i **RFC 8622**.

Pogledajmo i tablicu s usporednim preporukama za odabir klase ovisno o vrsti mrežnog prometa:

| **Vrsta mrežnog prometa** | **DSCP klase** |
|---|---|
| **TCP: SYN ili ACK**, druga ekstremno brza komunikacija s malim kašnjenjem | **AF21** |
| **SSH, VoIP, DNS** | **AF22** |
| Web komunikacija (**HTTP**, **HTTPS**), E-mail, …<br>[Ovo *je zadana (standardna) klasa*] | **AF23** |
| Aplikacije koje zahtijevaju visoku propusnost (pr. **Tor** mreža) | **AF11** |
| Video streaming | **AF31** |
| Aplikacije koje ne zahtijevaju posebne klasifikacije (pr. **Torrent**, ***Bitcoin***, …) | **LE** |

**Izvori informacija: (1357),(1358),(1359),(1360),(1361),(1362),(1363),(1364),(1365),(1367),** `man iptables`, `man nft`, `man ping`, `man tcpdump`, **RFC 791**, **RFC 1349**, **RFC 2474**, **RFC 2597**, **RFC 3260**, **RFC 4594**, **RFC 5865** i **RFC 8622**,

## 23.2. IP fragmentacija

IP fragmentacija je mehanizam ugrađen u *IP* protokol, koji omogućava razlamanje mrežnih *datagrama (paketa)*, kako bi se mogli kreirati mrežni paketi koji mogu proći kroz mrežu s manjom maksimalnom veličinom paketa (***MTU***) od one izvorne.
S druge strane primatelj zaprima ove fragmente te ih povezuje u cjeline od kojih kreira izvorni *datagram*, kao da nikada nije niti bio razlomljen. Ova procedura razlamanja paketa u fragmente definirana je u standardu **RFC 791**, a pojednostavljena procedura sastavljanja na strani primatelja fragmenata u standardu **RFC 815**.

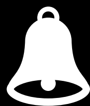
Kako bi se fragmenti ispravno identificirali i poslali, te s druge strane ispravno zaprimili te ponovno povezali u cjelinu, potrebna su sljedeća polja unutar IP zaglavlja:

- ***Identification*** polje koje identificira pojedini fragment, naravno uz izvorišne i *odredišne* IP adrese.
- ***Protocol*** kao vršni protokol koji je zadužen za transport, primjerice: **TCP**, **UDP**, **SCTP**.
- Te konačno polje: ***More Fragments*** **(*MF*)** koje označava kako novi fragmenti slijede ili kako ih nema više.

Linux koristi nekoliko *sysctl* sistemskih varijabli za optimizaciju IP fragmentacije (ovo su napredne mogućnosti):

| Sysctl varijabla | Vrijednost | Opis |
|---|---|---|
| `net.ipv4.ipfrag_high_thresh` | 4194304 | Maksimalna količina memorije koja se može koristiti za rastavljanje IP fragmenata. Vrijednost je LONG INT |
| `net.ipv4.ipfrag_time` | 30 | Maksimalno vrijeme u sekundama koliko će se fragmenti čuvati (držati) u međumemoriji. |
| `net.ipv4.ipfrag_max_dist` | 64 | Ovo je cjelobrojna vrijednost koja definira maksimalni "poremećaj" koji je dopušten među fragmentima koji dijele zajedničku IP adresu. Premaleni broj (pr.1-5) može uzrokovati odbacivanje fragmenata, a preveliki (pr. 10.000) nepravilno povezivanje fragmenata koji dolaze s različitih IP adresa. |

**Normalna komunikacija (fragmentacija je omogućena)**
Pogledajmo primjer u kojemu je dozvoljeno fragmentiranje paketa, što je normalan način ponašanja u mreži.
Naime u primjeru koji slijedi na mreži čiji *MTU* je 1500 *bajta*, kreirat ćemo mrežne pakete koji su veličine 2000 *bajta* na što će IP sloj reagirati tako što će morati fragmentirati (razlomiti) ove, za ovu mrežu i njen ***MTU***, prevelike pakete.
Koristit ćemo naredbu `ping` za slanje *ICMP* poruka na drugo računalo (192.168.1.254).



***ICMP*** ***koristi IP kao niži komunikacijski protokol. Dakle ne koristi TCP ili UDP kao transportne protokole***.

Pogledajmo kako izgleda slanje ICMP poruka s naredbom `ping`:

```
ping -c3 -s 2000 192.168.1.254

PING 192.168.1.254 (192.168.1.254) 2000(2028) bytes of data.
2008 bytes from 192.168.1.254: icmp_seq=1 ttl=255 time=2.72 ms
2008 bytes from 192.168.1.254: icmp_seq=2 ttl=255 time=5.08 ms
2008 bytes from 192.168.1.254: icmp_seq=3 ttl=255 time=2.29 ms
--- 192.168.1.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 3478ms
rtt min/avg/max/mdev = 2.298/3.101/5.085/1.159 ms
```

Vidljivo je nekoliko očekivanih stvari:

- Šaljemo 2000 *bajta* podataka od kojih IP sloj dodaje još 20 *bajta* te ICMP sloj dodaje 8 *bajta*, stoga je konačna veličina paketa 2028 *bajta*; vidljivo u drugom redu `2000(2028)`.
- U sljedećim redovima (`icmp_seq=1` do `icmp_seq=3`) vidimo kako se prikazuje slanje 2008 *bajta* paketa, jer se ovdje gleda samo na *ICMP* razinu, koja dodaje tih 8 *bajta* na naših 2000 *bajta* podataka.
- I na kraju vidimo kako su svi paketi uredno primljeni te da smo na njih dobili uredan odgovor (`3 packets transmitted, 3 received, 0% packet loss`).

Pogledajmo kako to izgleda na mreži, snimljeno programom ***Wireshark*** ili programom `tcpdump`.
Dakle šaljemo `ping` tj. ***ICMP*** upit s IP adrese:192.168.1.100 na IP adresu: 192.168.1.254.



Vezano za detalje oko naredbi: **`tcpdump`**, **`ping`** i **ICMP** protokola, pogledajte i poglavlja:
**25.7.8. Naredba *tcpdump*** ← za naredbu **tcpdump.**
**25.7.1. *Naredba ping*** ← za naredbu **ping.**
**25.7.1.1. ICMP poruka** ← za **ICMP** protokol.

**U konkretnom primjeru pogledajmo mrežni okvir odnosno paket broj 51:**

```
1.   Frame 51: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
2.   Ethernet II, Src: 08:00:27:93:e8:86, Dst: 00:12:00:85:4c:c6
3.   Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.254
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
      Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 1500
      Identification: 0x51d1 (20945)
      Flags: 0x01 (More Fragments)
          0... .... = Reserved bit: Not set
          .0.. .... = Don't fragment: Not set
          ..1. .... = More fragments: Set
      Fragment offset: 0
      Time to live: 64
4.     Protocol: ICMP (1)
          Header checksum: 0x5e87 [validation disabled]
          [Header checksum status: Unverified]
          Source: 192.168.1.100
          Destination: 192.168.1.254
          Reassembled IPv4 in frame: 52
 Data (1480 bytes)
```

Na izlistanju je vidljiv prvi paket koji šaljemo, a jasno je da se radi o *IP* paketu koji ima postavljen `Flag: More fragments: Set (1)` što znači kako slijedi još fragmenata (paketa) koji zajedno tvore jednu cjelinu.

Dodatno je vidljivo kako se navodi da se slijedeći fragment nalazi u paketu br. `52` (koji slijedi). U polju `Protocol` je vrijednost `ICMP` što indicira kako se radi o *ICMP* paketu. Sada naše računalo šalje i drugi fragment (paket) prema odredištu:

**Sada pogledajmo okvir odnosno paket broj 52:**

```
1.   Frame 52: 562 bytes on wire (4496 bits), 562 bytes captured (4496 bits) on
interface 0
2.   Ethernet II, Src: 08:00:27:93:e8:86, Dst: 00:12:00:85:4c:c6
3.   Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.254
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
      Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 548
      Identification: 0x51d1 (20945)
      Flags: 0x00
          0... .... = Reserved bit: Not set
          .0.. .... = Don't fragment: Not set
          ..0. .... = More fragments: Not set
      Fragment offset: 1480
      Time to live: 64
      Protocol: ICMP (1)
      Header checksum: 0x8186 [validation disabled]
      [Header checksum status: Unverified]
      Source: 192.168.1.100
      Destination: 192.168.1.254

+     [2 IPv4 Fragments (2008 bytes): #51(1480), #52(528)]
       [Frame: 51, payload: 0-1479 (1480 bytes)]
       [Frame: 52, payload: 1480-2007 (528 bytes)]
       [Fragment count: 2]
       [Reassembled IPv4 length: 2008]
       [Reassembled IPv4 data: 08007735d40500011cfa1f5a3857000008090a0b0c0d0e0f...]
```

Nastavak ispisa sadržaja mrežnog okvira broj **52**:

```
4. ++ Internet Control Message Protocol
       Type: 8 (Echo (ping) request)
       Code: 0
       Checksum: 0x7735 [correct]
       [Checksum Status: Good]
       Identifier (BE): 54277 (0xd405)
       Identifier (LE): 1492 (0x05d4)
       Sequence number (BE): 1 (0x0001)
       Sequence number (LE): 256 (0x0100)
       [Response frame: 54]
       Timestamp from icmp data: Nov 30, 2017 13:31:24.022328000 Central European Standard Time
       Timestamp from icmp data (relative): 2.668212000 seconds]
```

Pošto je on (**paket broj 52**) zadnji fragment, u njemu se vidi kako zastavica (*FLAG): `More fragments`* više nije postavljena. To definitivno znači kako je on zadnji fragment. Ponovno vidimo da se radi o *ICMP* poruci (u polju *Protocol*).
U ovom zadnjem fragmentu je sada jasno vidljiv (*ICMP*) kao viši protokol (`++`) dio.
Potom dolazi zanimljivi dio; pogledajte redak: *`+: 2 IPv4 Fragments (2008 bytes): #51(1480), #52(528)`*.

To znači kako se indicira da postoje dva fragmenta koji zajedno čine jednu logičku cjelinu i to točno:

- Fragment broj **51** koji nosi 1480 *bajta payloada* (podaci + dio s viših OSI slojeva).
- Fragment broj **52** koji nosi 528 *bajta payloada* (podaci + dio s viših OSI slojeva).

Dakle sve bez *ICMP* sloja:

- Prvi fragment (br.**51**): 1480 - 8 (ICMP dio) = 1472 *bajta.*
- Drugi fragment (br.**52**): 528 - 8 (ICMP dio) = 520 *bajta.*

Ukupno 1472 + 520 = 1992 *bajta.*
Odnosno 1480 + 528 = 2008 *bajta* na ICMP sloju, koji su vidljivi u `ping` naredbi na početku testa.

Kako bi u programu **Wireshark** vidjeli fragmente na jednostavan način, potrebno je uključili sljedeći filter (u polju za filtere):

```
ip.flags.mf == 1 or ip.frag_offset > 0
```

## Zabrana IP fragmentacije

Ako je u *IP* zaglavlju paketa, zastavica: ***Don't Fragment (DF)*** postavljena, paket NE SMIJE biti fragmentiran te će, ako njegova veličina prelazi [***MTU***]() vrijednost mrežnog sučelja on biti odbačen. U slučaju kada pokušamo poslati paket koji je veće veličine od ***MTU*** našeg mrežnog sučelja (mrežne kartice), a zastavica *Don't Fragment (**DF**)* je postavljena, naše računalo neće niti poslati ovakav (preveliki) paket na mrežu.
U slučaju kada je ***DF*** zastavica postavljena, a naš je paket veličine koja zadovoljava osnovni uvjet slanja na mrežu odnosno, ako je jednak ili manji od ***MTU*** vrijednosti našeg mrežnog sučelja, ali već prvi (ili bilo koji) usmjerivač koji ima na svom vanjskom mrežnom sučelju manji ***MTU*** od našeg, on će odbaciti naš (njemu preveliki) paket. Zatim će nam (obično) poslati *ICMP error* poruku.



Zabranu fragmentacije odnosno postavljen ***DF*** bit, standardno imaju protokoli poput ***HTTP***, ***HTTPS*** i nekih drugih.



Za osnovni primjer u kojemu izričito zabranjujemo IP fragmentaciju paketa, pogledajte primjer iz poglavlja o ***MTU***: **23.4.2.1 MTU primjeri.**

Pogledajte i osnove korištenja naredbe ***ping*** te oblik **ICMP** poruke:
**25.7.1. *Naredba ping*.**
**25.7.1.1. ICMP poruka.**

Podsjetite se **IP** protokola, a posebno IP zaglavlja (polje **"Flags"**):
**23.1. Oblik IP poruke (paketa).**

Pogledajmo i kako na mreži pronaći mrežne pakete koji imaju postavljen ***DF*** bit.
Prisjetimo se da se ***DF*** bit nalazi u IP zaglavlju u takozvanom ***Flags*** polju. ***IP flags*** polje je veličine tri bita:

- Prvi bit mora biti `0`,
- Drugi bit ako je `1` označava da je **DF** aktivan.
- Treći bit ako je `1` označava da je ***More fragment*** ("***MF***") aktivan.

Međutim prvo moramo shvatiti kako filtrirati željeno ***DF*** polje unutar IP zaglavlja. Pogledajmo skraćenu tablicu IP zaglavlja:

| Bit/Oktet | 0 | | 1 | | 2 | 3 |
|---|---|---|---|---|---|---|
| 0 | Version | IHL | DSCP | ECN | Total length | |
| 32 | Identification | | | | Flags | Fragment Offset |
| 64 | Time to Live | | Protocol | | … | … |

Dakle vidimo da je polje ***Flags*** na poziciji 7-mog bajta:

*Version (4bita) + IHL (4bita) + DSCP (6bita) + ECN (2bita) + Total length (16 bitova) + Identification (16bitova)*

U 7-mom bajtu, na drugoj poziciji od lijevo, gledano binarno je `01000000` to jest decimalno je to vrijednost **`64 (DF bit)`**.

U 7-mom bajtu, na trećoj poziciji od lijevo, gledano binarno je `00100000` to jest decimalno je to vrijednost **`32 (MF bit)`**.

Za dohvaćanje ovakvih paketa, koristit ćemo naredbu `tcpdump` na sljedeći način:

```
tcpdump -n -vv -i eth0 'ip[6] = 64'
```

U ispisu ćemo uz svaki ovakav paket vidjeti i poruke poput (**`flags [DF]`**)

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na IP sloju. Sintaksa ove naredbe koju ćemo korisiti jest:

***`nstat -opcije ImeBrojača`***

Primjerice, ako nas zanima brojač imena `IpInReceives`:

```
nstat -az IpInReceives
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `IpInReceives` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za IP fragmentaciju:

| Naziv brojača | Opis |
|---|---|
| **`IpReasmReqds`** | Broj primljenih IP fragmenata koje je trebalo ponovno sastaviti u ovom entitetu. |
| **`IpReasmOKs`** | Broj uspješno ponovno sastavljenih IP *datagrama*. Optimizacijom vrijednosti *sysctl* varijable: `net.ipv4.ipfrag_low_thresh` moguće je optimizirati mehanizam sastavljanja datagrama. To se može postiči s postavljanjem navedene donje granice za količinu fragmentiranog prometa prije nego što kernel počne sastavljati fragmentirane pakete. |
| **`IpReasmTimeout`** | Maksimalan broj sekundi u kojima se primljeni fragmenti zadržavaju dok čekaju ponovno sastavljanje u ovom entitetu. Optimizacijom vrijednosti *sysctl* varijable: `net.ipv4.ipfrag_time` moguće je optimizirati vremenski mehanizam sastavljanja datagrama postavljanjem vremena u sekundama nakon kojeg će se fragmentirani paketi automatski odbaciti ako nisu sastavljeni (*reassemblirani*). |
| **`IpReasmFails`** | Broj kvarova koje je otkrio algoritam ponovnog sastavljanja IP-a (iz bilo kojeg razloga: isteklo vrijeme, pogreške itd.). Imajte na umu da ovo nije nužno broj odbačenih IP fragmenata budući da neki algoritmi (osobito algoritam u **RFC 815**) mogu izgubiti trag broja fragmenata kombinirajući ih kako su primljeni. |
| **`IpFragOKs`** | Broj IP datagrama koji su uspješno fragmentirani u ovom entitetu. |
| **`IpFragFails`** | Broj IP datagrama koji su odbačeni jer su morali biti fragmentirani u ovom entitetu, ali nisu mogli biti, npr. jer je postavljena njihova zastavica **Dont Fragment (`DF`)**. Drugi razlog može biti preopterećenost sustava. Optimizacijom vrijednosti sysctl varijable: `net.ipv4.ipfrag_high_thresh` moguće je ublažiti ovaj problem, postavljanjem gornje granice za količinu fragmentiranog prometa prije nego što kernel počne odbacivati fragmentirane pakete. |
| **`IpFragCreates`** | Broj fragmenata IP datagrama koji su generirani kao rezultat fragmentacije u ovom entitetu. To može ukazivati na situacije kada se paketi fragmentiraju zbog postavljene **MTU** vrijednosti. |



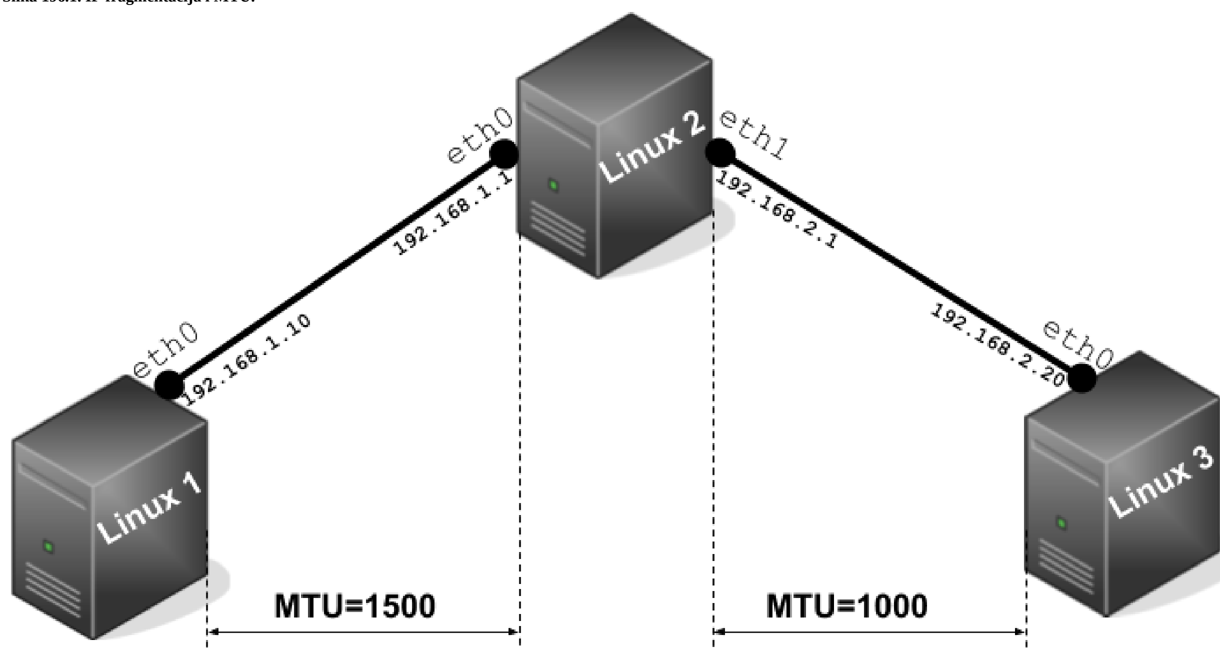
Za detalje oko **MTU** pogledajte poglavlja:
**23.2.1. Optimizacije vezane za fragmentaciju i MTU.**
**23.4.2. .Maximum Transmission Unit (*MTU*).**

**Izvor informacija: (701),(1044),(1366),(1651),(K-6), (K-10), RFC 791, RFC 815, RFC 1213,**
**`man ping`**,**`man tcpdump`**,**`man 7 ip`**,**`man nstat`**.

## 23.2.1. Optimizacije vezane za fragmentaciju i MTU

**Slijedi napredna cjelina!**
Kako ćete vidjeti u poglavlju: **23.4.2. Maximum Transmission Unit (*MTU*)** svako računalo, a pogotovo usmjerivač na svakoj svojoj mrežnoj kartici odnosno mrežnom sučelju ima postavljenu takozvanu ***MTU***[62] veličinu (Engl. *Maximum Transmission Unit*). ***MTU*** definira maksimalnu veličinu podataka (bez zaglavlja) koji mogu stati u mrežni paket na OSI mrežnom sloju tri (OSI 3) odnosno IP sloju prema TCP/IP protokolu. ***MTU*** veličina za lokalne ***ethernet*** mreže je obično 1500 bajta, ali za neke druge mreže može biti i drugačija. Poput primjerice: A/S/HDSL, ADSL2+, ATM i druge, a pogotovo za neke vrste dodatnih ugnježđivanja (*enkapsulacija*), poput raznih VPN tunela. Zamislimo (slika **196.1**) da je na naše mrežno sučelje (***eth0***) s ***MTU*** vrijednosti od 1500 bajta došao mrežni paket, a naše računalo je usmjerivač ili mrežni uređaj koji taj paket mora proslijediti na svoje drugo mrežno sučelje, prema drugoj mreži. Međutim drugo mrežno sučelje (***eth1***) je spojeno na drugačiju mrežu čiji ***MTU*** je samo 1000 bajta. Ovdje govorimo u slučaju kada je naše računalo o kojem govorimo: „***Linux 2***" na slici **196.1**.
Tada naše računalo mora razlomiti paket čiji ***MTU*** je 1500 bajta, na dva manja paketa od kojih niti jedan ne može biti veći od ***MTU*** izlaznog mrežnog sučelja (***eth1***) od 1000 bajta. Ovo razlamanje paketa na manje pakete se zove fragmentacija.
Ovakvo razlamanje paketa unosi određenu latenciju odnosno kašnjenje jer je potrebno određeno vrijeme, ali i resursi (CPU i RAM) kako bi se to odradilo. Stoga primarno uređaj koji odrađuje fragmentaciju paketa, biva nešto više opterećen nego da to ne radi. Stoga u određenim sustavima, poput pojedinih mreža u velikim podatkovnim centrima za virtualizaciju, obično na poslužiteljima za virtualizaciju (Tzv. ***Hipervizorima***) imamo konfiguriranu zabranu fragmentacije mrežnih paketa, kako bi se resursi tih poslužitelja što bolje iskoristiti. Odnosno da oni ne bi trošili resurse na fragmentaciju i defragmentaciju. To se obično konfigurira unutar *Hipervizorovog* virtualnog preklopnika (tzv. *Virtual Switch*), kao što je primjerice ***Open vSwitch***. On konkretno ima opcije[(A),(B)] za fragmentaciju: `options: df_default` ili `ipf-set-enabled/ipf-set-disabled`.

**Slika 196.1. IP fragmentacija i MTU.**




S aplikacijske razine, moguće je unutar IP dijela paketa koji šaljemo na odredišnu IP adresu, postaviti takozvani *Don't Fragment* (**DF**) bit. On se u IP zaglavlju paketa nalazi u polju sa zastavicama (Engl. ***Flags***). Ako je on postavljen to naznačuje svim tranzitnim uređajima da, ako zaprime ovaj paket, a on ima veći ***MTU*** od sučelja na koje ga tranzitni uređaj mora proslijediti, da ga jednostavno odbaci. Dakle ako bi s računala ***Linux 1*** poslali paket veličine 1500 bajta prema računalu
***Linux 3*** s postavljenim *Don't Fragment* (**DF**) bitom, računalo ***Linux 2*** bi ga odmah odbacilo jer bi ga za slanje prema računalu ***Linux 3*** moralo razlomiti na dva manja paketa unutar granice od ***MTU*** 1000, a mi smo mu to s ovom IP opcijom zabranili.



Pogledajte primjer kreiranja paketa s postavljenim ***DF*** bitom, u poglavlju:
**23.4.2.1. MTU primjeri**.

Ne zaboravimo, isto kako se troše resursi za fragmentiranje, na tranzitnim uređajima, a u ovom slučaju bi to bilo računalo ***Linux 2***, troše se resursi i za ponovno sastavljanje tih paketa u izvorni oblik; obično na odredišnom računalu. To uglavnom nije problem, ali kod visoko performantnih sustava i onih koji zahtijevaju što manje kašnjenje (latenciju), to može biti problem.

Druga opcija, ako ne želimo da itko u mreži radi fragmentiranje paketa je (obično) za posebne namjene, a to je da na svim uređajima na mreži smanjimo ***MTU*** na najmanju ***MTU*** vrijednost koja postoji u mreži; tada niti jedan tranzitni uređaj na mreži neće morati raditi fragmentaciju odnosno odredišni uređaj neće morati raditi defragmentaciju tj. ponovno povezivanje paketa.

**Izvori informacija: (62), *MTU (poglavlje 23.4.2.)*, RFC 791.**

### 23.2.1.1. Path MTU Discovery

Srećom imamo i treću opciju, a ona je takozvano otkrivanje puta ***MTU*** znana kao ***PMTUD*** (Engl. *Path MTU Discovery*).
Ona je standardizirana tehnika za određivanje maksimalne veličine prijenosne jedinice (***MTU***) na mrežnoj putanji između dvije krajnje strane u komunikaciji, pomoću internetskog protokola (***IP***), s ciljem izbjegavanja fragmentacije IP paketa.
***PMTUD*** je izvorno bio namijenjen usmjerivačima koji su koristili IP protokol inačice 4 (***IPv4***) pa je standardiziran za IPv4 u RFC 1191 te za IPv6 u RFC 8201. Potom je u RFC 4821 opisano proširenje tehnika koje funkcioniraju bez podrške putem ***ICMP*** protokola (Engl. *Internet Control Message Protocol*).

Za IPv4 pakete, svaki uređaj na putu čiji je ***MTU*** manji od ovog (poslanog) paketa, odbacit će taj prvi paket te poslati povratnu ***ICMP*** poruku. Ova poruka je takozvani *Path MTU Discovery* koja radi tako da se šalje *ICMP* paket u kojem se postavlja ***Don't Fragment*** (**DF**) bit u IP zaglavlju, te se u *ICMP* dijelu postavlja: *Destination unreachable* te *Fragmentation Needed* (Type 3, Code 4). Ovaj paket će sadržavati ***MTU*** vrijednost uređaja koji šalje ovu poruku. Primitkom ove *ICMP* poruke pošiljatelj može smanjiti svoju ***MTU*** veličinu za odredišnu IP adresu. Postupak se ponavlja sve dok ***MTU*** veličina pošiljatelja nije dovoljno malena da pređe čitav put do odredišta bez fragmentacije, jer će i sljedeći uređaj napraviti isto, odnosno poslati ovakvu *ICMP* poruku; ako ima odredišno sučelje s manjom ***MTU*** vrijednosti.

IPv6 usmjerivači ne podržavaju fragmentaciju i stoga ne podržavaju opciju *Don't Fragment* (**DF**). Za IPv6, *Path MTU Discovery* djeluje tako što u početku pretpostavlja da je ***MTU*** svugdje jednak ***MTU***-u na (našem) mrežnom sučelju odakle potječe promet. Tada će slično kao kod IPv4, bilo koji uređaj duž puta čiji je ***MTU*** manji od paketa, odbaciti taj paket i poslati natrag *ICMPv6* poruku*: Packet Too Big (Type 2)* koja će sadržavati svoj ***MTU***, također omogućavajući izvornom pošiljatelju da prikladno smanji svoju ***MTU*** vrijednost. Postupak se ponavlja sve dok ***MTU*** nije dovoljno malen da pređe čitav put bez fragmentacije. Ne zaboravite da se *ICMPv4* i *ICMPv6* protokoli malo razlikuju.

Ako se veličina ***MTU***-a promijeni nakon uspostavljanja veze, odnosno u bilo kojem trenutku i niža je od prethodno određene ***MTU*** vrijednosti, prvi veći paket uzrokovat će *ICMP* pogrešku i pronaći će se nova, niža *MTU* vrijednost. Suprotno tome, ako *PMTUD* utvrdi da putanja do odredišta omogućuje veći ***MTU*** nego što je to trenutno postavljeno odnosno odabrano, Linux će povremeno provjeravati je li se put promijenio i sada dopušta veće pakete odnosno veće *MTU* vrijednosti.
Međutim zbog sigurnosnih razloga, na javnim poslužiteljima i uređajima na internetu, često je slanje ICMP poruka onemogućeno, tako da standardne implementacije ***PMTUD*** neće raditi (osim onih baziranih na: RFC 4821), što ne mora važiti i za lokalne (***LAN***) mreže.



I na *Linuxu* i na *Windows* sustavu postoji i brojač***** koji je prema zadanim postavkama postavljen na deset minuta, nakon kojih provjerava, je li došlo do nekih promjena vezanih za veličinu ***MTU-a*** unutar mreže.

Na Linuxu postoji nekoliko sistemskih (kernel) varijabli odnosno postavki koje se tiču ovih postavki:

| Sysctl varijabla | Postavljena vrijednost | Opis |
|---|---|---|
| `net.ipv4.ip_no_pmtu_disc` | 0 | `0` - Omogućena je upotreba *Path MTU Discovery-a* ***(PMTUD).***<br>→ ***Stoga se ne šalje Don't Fragment (DF).***<br>`1` - Isključuje se upotreba ***PMTUD,*** ali ako se zaprimi *ICMP* poruka za ***PMTU***, tada će se za to odredište koristiti ***MTU*** veličina****** koja je definirana u **`net.ipv4.route.min_pmtu`**.<br>`2` - Isto kao (**1)** ali se dolazne ***PMTUD*** ne koriste.<br>`3` - Poput (**0**) uz značajne dodatne sigurnosne postavke.<br>***Ne preporučuje se!.*** |
| `net.ipv4.ip_forward_use_pmtu` | 0 | `0` – Isključeno jer se ne želi vjerovati PMTUD kod prosljeđivanje paketa (*forwarding*-a).<br>`1` – Uključeno (**ne preporučuje se**). |
| `net.ipv4.route.min_pmtu` | 552 | Minimalna** ***MTU*** vrijednost koja će se postaviti. |
| `net.ipv4.route.mtu_expires` | 600 | Brojač (s) koji definira vrijeme nakon kojeg će se provjeravati ***MTU*** veličina*****. Standardno je to 600s.=10min. |

S trenutnim standardnim postavkama (`net.ipv4.ip_no_pmtu_disc=0`), prema shemi na slici 196.1., probajmo s našeg računala **Linux 1** (192.168.1.10), preko ***Linux 2*** računala koje je sada u funkciji usmjerivača (a isto ponašanje je i za *Proxy*, *Vatrozid*, *Load Balancer* i slične funkcionalnosti) poslati mrežni paket veličine 1450 bajta.
Zbog jednostavnosti šaljemo ***ICMP*** paket. Dakle želimo biti sigurni da šaljemo mrežni paket čija konačna veličina je manja od 1500 bajta, koliki je ***MTU*** na prvom mrežnom sučelju (**`eth0`**) i mreži prema računalu ***Linux 2***.

Međutim računalo (sada usmjerivač prema funkciji koju obavlja): ***Linux 2***, na drugom odnosno odredišnom mrežnom sučelju (**`eth1`**) prema konačnom odredištu ***Linux 3*** (192.168.2.20) ima postavljen ***MTU*** na 1000 bajta. Stoga bi nam on kao pošiljatelju koji bi trebao proslijediti veći paket (***MTU*** vrijednost) od one *MTU* vrijednosti na odredišnoj mreži, trebao poslati već navedenu posebnu *ICMP* poruku (*Destination unreachable* te *Fragmentation Needed* [Type 3, Code 4]) na koju bi trebali reagirati smanjivanjem *MTU* vrijednosti za tu odredišnu IP adresu (192.168.2.20).

Ovakvo ponašanje je normalno s navedenim postavkama (`net.ipv4.ip_no_pmtu_disc=0`), a u suprotnom (pr. da imamo `net.ipv4.ip_no_pmtu_disc=2`) ne bi dobili *ICMP* poruku o korekciji ***MTU*** vrijednosti za tu konkretnu odredišnu IP adresu, te bi računalo koje radi kao usmjerivač: ***Linux 2***, normalno odradio fragmentiranje paketa.

Sada konačno pošaljimo *ICMP* paket veličine 1450 bajta (ne računajući *ICMP* zaglavlje od 8 bajta) i to samo četiri paketa:

```
ping -s 1450 192.168.2.20 -c4

PING 192.168.2.20 (192.168.2.20) 1450(1478) bytes of data.
From 192.168.1.1 icmp_seq=1 Frag needed and DF set (mtu = 1000)
1458 bytes from 192.168.2.20: icmp_seq=2 ttl=63 time=2.10 ms
1458 bytes from 192.168.2.20: icmp_seq=3 ttl=63 time=2.04 ms
1458 bytes from 192.168.2.20: icmp_seq=4 ttl=63 time=2.10 ms
--- 192.168.2.20 ping statistics ---
4 packets transmitted, 3 received, 1 errors, 25% packet loss, time 42ms
rtt min/avg/max/mdev = 1.409/1.913/2.225/0.276 ms
```

Dakle vidimo da smo već nakon prvog poslanog paketa dobili poruku: `icmp_seq=1 Frag needed and DF set (mtu = 1000)` koja nam naznačava da smanjimo *MTU* na 1000 ako šaljemo pakete na odredišnu IP adresu: 192.168.2.20.

Pogledajmo što se dogodilo u dijelu tablice usmjeravanja za odredišnu IP adresu: 192.168.2.20.

```
ip -d route get 192.168.2.20
unicast 192.168.2.20 via 192.168.1.28 dev enp0s3 table main src 192.168.1.231 uid 0
    cache expires 382sec mtu 1000
```



Naredba `ip -d` nam daje detaljniji prikaz, nakon kojega možemo nastaviti nîzati naredbe i/ili druge prekidače naredbe `ip`.

Vidimo da je u dijelu tablice usmjeravanja zaduženom za pojedine IP adrese (unose) vidljivo da je sada naš sustav postavio *MTU* vrijednost na 1000 (`mtu 1000`), iako je za naše fizičko mrežno sučelje (za računalo ***Linux 1***) i dalje vidljivo:

```
ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 08:00:27:a1:d5:0c brd ff:ff:ff:ff:ff:ff
```

Dakle *MTU* na mrežnom sučelju je i dalje 1500 bajta i on se primjenjuje za slanje paketa na mrežu.
Međutim to znači i da se *MTU* postavljen za točno određeno računalo (192.168.2.20) primjenjuje sâmo za slanje mrežnih paketa na tu IP adresu (192.168.2.20).



Ako želimo isključiti optimizaciju *MTU* veličine upotrebom ***Path MTU Discovery*** mogućnosti, potrebno je ***sysctl*** varijablu `net.ipv4.ip_no_pmtu_disc` postaviti na vrijednost **2**.

To možemo postići s naredbom `sysctl` na sljedeći način:

```
sysctl net.ipv4.ip_no_pmtu_disc=2
```

Ili ako želimo napraviti trajnu promjenu možemo u datoteku `/etc/sysctl.conf` dodati sljedeći unos odnosno redak:

```
net.ipv4.ip_no_pmtu_disc = 2
```

Dodatno, ako koristimo ***Path MTU Discovery***, možemo i povećati minimalnu *MTU* vrijednost. Naime ako koristimo postavku koja nije standardna: `sysctl net.ipv4.ip_no_pmtu_disc=1`, tada kada se zaprimi *ICMP* poruka *Path MTU Discovery*, **PMTU** sustav će za ovo odredište (konkretnu odredišnu IP adresu) postaviti *MTU* na vrijednost varijable: `net.ipv4.route.min_pmtu`. Stoga bi u tom slučaju bilo dobro nešto ju povećati jer je ona standardno postavljena na **552**.

Dakle možemo ju primjerice povećati na **1100**, što je dovoljno veliko, ali i dovoljno malo za veliku većinu mreža:

```
net.ipv4.route.min_pmtu = 1100
```

Tu promjenu možete i trajno snimiti u datoteku `/etc/sysctl.conf`.

Ako želite provjeriti ***minmtu*** vrijednost; to možemo napraviti za **`eth0`** mrežno sučelje, na sljedeći način:

```
ip -d link show eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT
group default qlen 1000
link/ether 02:cc:bc:05:c7:f6 brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 560 maxmtu
3712 addrgenmode none numtxqueues 8 numrxqueues 8 gso_max_size 65536 gso_max_segs
65535
```

Dakle vidimo da je minimalni *MTU* postavljen na 560 (`minmtu 560`), te vidimo maksimalni dostupan (za ovo sučelje) na: `maxmtu 3712`. Osim *MTU* vrijednosti ovdje vidimo i efektivne vrijednosti TX/RX nîzova (*Queues*) i ***GSO*** postavke.



Pogledajte poglavlje vezano za ***GSO***:
**25.5.2. LSO, TSO i GSO**

Pogledajte i poglavlje:
**25.2.2. Multiqueue funkcionalnost.**

## *TCP Packetization-Layer Path MTU discovery*

Moguće je koristiti i mehanizme za dogovaranje ***MTU*** veličine i preko TCP protokola; tzv. *TCP Packetization-Layer Path MTU discovery*, kako je definirano u **RFC 4821**.

Neke od *sysctl* varijabli zaduženih za dogovaranje ***MTU*** veličine preko TCP protokola su: `net.ipv4.tcp_mtu_probing` (za aktiviranje) te `net.ipv4.tcp_base_mss` za definiranje početne ***MSS*** veličine.
Prema navedenom ***RFC*** standardu, dostupne su i sljedeće ***sysctl*** varijable odnosno opcije:

- `net.ipv4.tcp_probe_interval` – ovdje se definira svakih koliko minuta će se slati *TCP path MTU discovery* poruke. Standardno je 600 sekundi (10 minuta).
- `net.ipv4.tcp_probe_threshold` – nakon koliko bajta se zaustavlja TCP *Path MTU Discovery probing*.



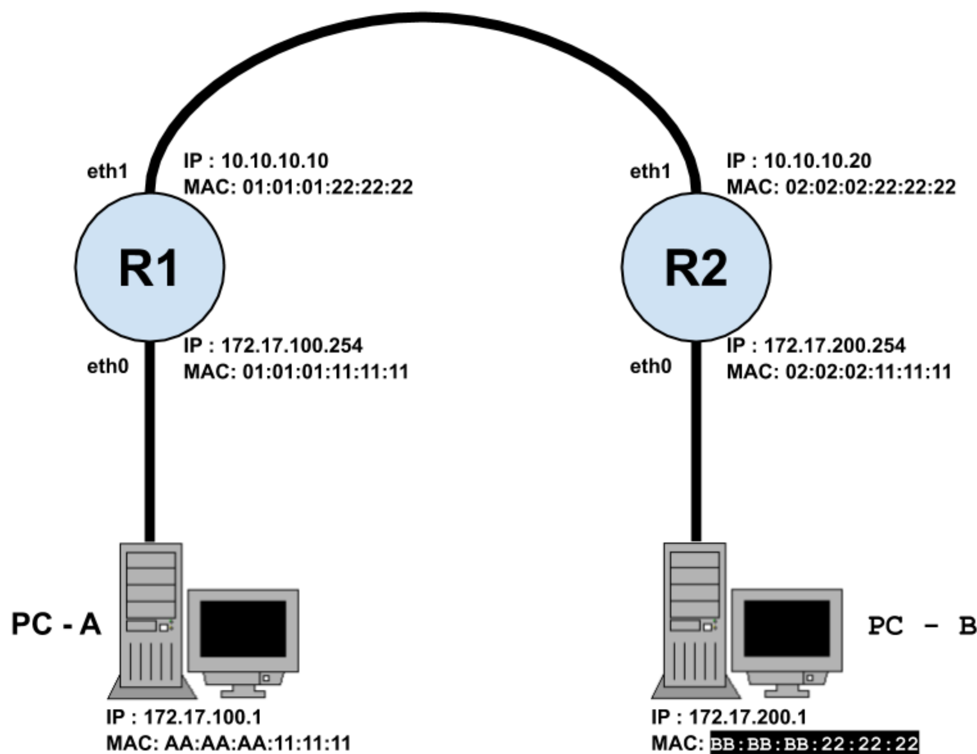
Pogledajte i poglavlja vezana za ***MTU:***
**23.4.2. Maximum Transmission Unit (*MTU*).**
**23.4.2.1. MTU primjeri.**

**Izvori informacija: (325),(642),(936),(937),(1022), Implementacija IP protokola (sekcija „`IP_MTU_DISCOVER`“), `man ip link`,`man 7 ip`,`man 7 tcp`, RFC 1191, RFC 8201 i RFC 4821.**

# 23.3. Kako radi mreža (OSI 2 + OSI 3)

Sada kada smo naučili ponešto o drugom i trećem OSI sloju mreže (OSI 2 i OSI 3) povezat ćemo naše znanje u konkretan primjer. Stoga zamislimo sljedeću topologiju mreže (slika 197) u kojemu računalo: **PC - A** želi poslati mrežni paket na računalo: **PC - B**.

**Slika 197. Topologija mreže.**



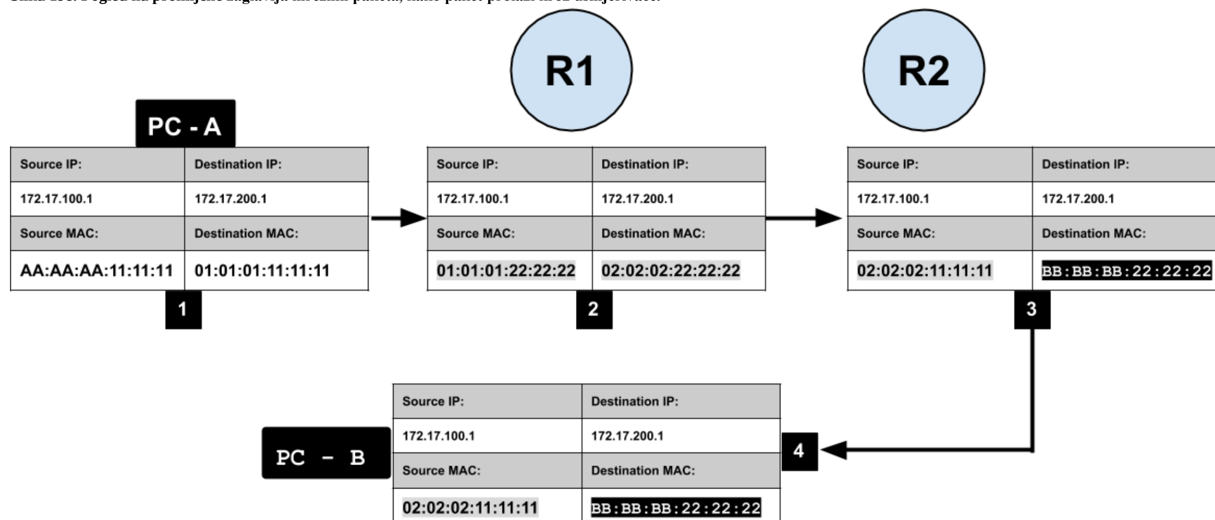

Pogledajmo sâmo dio paketa koji sadrži:

- Izvorišnu **IP** (*Source* IP) i odredišnu **IP** (*Destination* IP) adresu.
- Izvorišnu **MAC** (*Source MAC*) i odredišnu **MAC** (*Destination* MAC) adresu.

Za sve IP adrese pretpostavljamo masku mreže: `255.255.255.0` pa ju više nećemo spominjati.

Sada pogledajmo što će se dogoditi s mrežnim paketom poslanim s računala **PC-A** prema računalu **PC-B**, prolaskom kroz mrežu odnosno kroz usmjerivače **R1** i **R2** kako je vidljivo na slici 198.

**Slika 198. Pogled na promijene zaglavlja mrežnih paketa, kako paket prolazi kroz usmjerivače.**




**1.** Računalo **PC - A** želi poslati paket na IP adresu: `172.17.200.1` (na računalo **PC - B**), a pošto to nije IP adresa iz njegove mreže ono prvo gleda svoju tablicu usmjeravanja (*routing tablicu*) i traži ima li neku *rutu* do te mreže. Pošto je nema (pretpostavimo da je tako u ovom slučaju) on paket mora poslati na jedino preostalo mjesto odnosno na svoj *Default Gateway (to je njegov usmjerivač **R1**)* s IP adresom `172.17.100.254`, za koji prvo mora saznati njegovu MAC adresu.

Ta MAC adresa će biti upisana kao odredišna (*destination)* MAC adresa paketa koji šalje:

- Ako ju već ima (MAC adresu), formira paket i šalje ga na mrežu.
- Ako ju nema (MAC adresu), on šalje ***ARP*** zahtjev/upit na mrežu kako bi saznao MAC adresu od IP adrese: `172.17.100.254` te formira paket s njom (kada ju dobije s ARP upitom) i šalje ga na mrežu.

Dakle naš konačni paket sadrži IP adresu ovog računala koje ga šalje (*Source IP*) i također MAC adresu ovog računala (*Source MAC*) te odredišnu krajnju IP adresu: `172.17.200.1`.

Međutim on kao odredišnu MAC adresu stavlja MAC adresu *Default Gateway-a*, kako bi bio siguran da će podrazumijevani usmjerivač tj. *Default gateway* (usmjerivač **R1**) preuzeti ovaj mrežni paket.



***Nikada**, a niti u ovom procesu navedenom u primjeru, ne mijenjaju se izvorišne i odredišne **IP** adrese paketa, već samo izvorišne i odredišne **MAC** adrese.*

**2.** Sada je ***Default Gateway*** (usmjerivač **R1**) preuzeo paket i on gleda svoju tablicu usmjeravanja te na osnovu nje priprema paket tako da zadržava izvorišne i odredišne IP adrese, ali on kao *izvorišnu* MAC adresu stavlja svoju MAC adresu, a kao *odredišnu MAC* adresu stavlja MAC adresu od usmjerivača na koji on šalje paket odnosno od usmjerivača **R2.** Potom šalje tako pripremljeni paket na usmjerivač **R2**. Važno je razumjeti da i usmjerivač **R1** odluku o slanju ovog i svakog drugog paketa donosi tako da prvo traži ima li direktnu *rutu* do odredišta, a tek ako ju nema on paket šalje na ono svoje mrežno sučelje na kojem je njemu podešen njegov *default gateway* odnosno koristi se *podrazumijevana ruta*, koja u našem slučaju pokazuje na usmjerivač **R2**.

**3.** U ovom trenutku usmjerivač **R2** preuzima paket te i on gleda svoju tablicu usmjeravanja i na osnovu nje, a koja u našem slučaju pokazuje na krajnje (odredišno) računalo **PC – B**, priprema mrežni paket za slanje na ono svoje mrežno sučelje koje je spojeno na traženu odredišnu mrežu prema računalu **PC – B**. Potom priprema paket tako da zadržava istu *izvorišnu* i *odredišnu* IP adresu, ali sada kao *izvorišnu* MAC adresu stavlja svoju MAC adresu, a kao *odredišnu* MAC adresu, MAC adresu od krajnjeg računala **PC - B** tj. od njegove mrežne kartice čiju MAC adresu poznaje. U slučaju da ne poznaje MAC adresu od računala **PC – B**, traži ju pomoću ARP zahtjeva. Kada su sve MAC adrese poznate šalje se paket na odredište prema računalu **PC - B**. Važno za napomenuti je da svaki paket sadrži i neke konkretne podatke i viši protokol, ali to smo u ovom primjeru zanemarili zbog fokusa na IP i MAC adrese i onoga što se s njima događa u prijenosu kroz mrežu.

**Izvori informacija: (K-6).(K-10),(K-11).**

# 23.4. Usmjerivači (*routeri*) i OSI slojevi 3 i 4

U ovoj cjelini upoznati ćemo se s OSI slojevima tri i četiri (OSI 3 i OSI 4) čije funkcionalnosti koriste usmjerivači (*routeri*), ali i *multilayer* preklopnici. Usmjerivači su specifični po tome što je kod njih:

- *Broadcast* domena unutar jednog mrežnog sučelja (*porta/interface-a*). Dakle *broadcast* poruke koje su poslane na mrežu koja završava na jednom mrežnom sučelju usmjerivača, usmjerivač ne propušta na druga mrežna sučelja (*portove)*.
- Obrada IP paketa odnosno OSI sloj tri (OSI 3) procesiranje odrađuje CPU uređaja (usmjerivača) i sâmim time unosi veće kašnjenje od obrade na OSI sloju dva (OSI 2) koje odrađuju preklopnici (čipovi). Naime usmjerivači mrežne pakete uspijevaju obraditi i proslijediti unutar milisekundi, dok preklopnici obrađuju i prosljeđuju mrežne pakete odnosno mrežne okvire (jer je to OSI sloj dva), unutar granica mikrosekundi što je 1.000 puta brže od usmjerivača
- Imaju visoku cijenu prema broj mrežnih sučelja (*portova/interface-a*).

U slučaju usmjeravanja (engl. *routing*), koje se događa na **OSI** sloju tri (IP adrese prema TCP/IP modelu), uređaji odluku za usmjeravanje paketa donose na osnovu IP adresa. Tablica na osnovu koje se odrađuje usmjeravanje se zove tablica usmjeravanja odnosno *routing tablica*. Dakle sličan mehanizam kao za preklopnike, ali na drugom OSI sloju i još malo kompleksniji zbog protokola usmjeravanja i mogućnosti dinamičkih promjena tablica usmjeravanja pomoću protokola za usmjeravanje poput: ***RIP***, ***OSPF***, ***BGP***, … Uređaji koji se bave usmjeravanjem zovemo usmjerivači (*routeri*), ali postoje i preklopnici (*switchevi*) koji mogu odrađivati i taj dio posla. Ovakve preklopnike zovemo preklopnici koji rade na više OSI slojeva (2, 3 i 4) odnosno nazivaju se **Multilayer** preklopnici. Ovakvi preklopnici vrlo su važni u današnjim mrežama jer nam daju mogućnost da uredno segmentiramo lokalnu mrežu, a da to ne unese usporavanja, koja bi uveli klasični usmjerivači.

Kako smo rekli, kod usmjeravanja, *usmjerivač* ili *Multilayer* preklopnik zaprima svaki pojedini mrežni paket te gleda samo OSI sloj tri (OSI 3) odnosno sloj na kojemu je prema TCP/IP modelu IP protokol.

IP protokol na svom sloju između ostalih polja sadrži i polja u kojima se nalaze:

- Izvorišna IP adresa (engl. *Source IP*).
- Odredišna IP adresa (engl. *Destination IP*).
- Provjerni zbroj (engl. *Checksum*).
- Ukupna duljina paketa (i IP zaglavlja i podataka).
- TTL (*Time to Live*) odnosno vrijeme dozvoljenog života mrežnog paketa, kao i deseci drugih opcija i zastavica.

Usmjerivač za svaki primljeni mrežni paket pristigao na bilo kojem njegovom mrežnom sučelju (*portu*), prvo provjerava *IP* zaglavlje i to dio s izvorišnom i odredišnom IP adresom.

On potom uspoređuje odredišnu IP adresu sa svojom tablicom usmjeravanja. Na osnovi ove tablice koja se može i dinamički osvježavati pomoću protokola usmjeravanja (*routing protokoli*) usmjerivač usmjerava paket preko onog mrežnog sučelja preko kojeg će paket moći doći do svog odredišta (sve ovisno o topologiji mreže):

- Ili direktno.
- ili preko drugih usmjerivača.

**Izvori informacija: (K-6),(K-8),(K-10).**

### 23.4.1. Time to live (*TTL*)

Pošto svaki mrežni paket u IP zaglavlju sadrži i ***TTL*** polje, svaki usmjerivač, mora pri preuzimanju svakog paketa, ovo polje, koje je zapravo brojač vremena života paketa, smanjiti. Prema osnovnom standardu u ovo polje se inicijalno upisivao maksimalan broj sekundi koje paket smije biti "na životu" (engl. *Time to live*).

Međutim u praksi se u ***TTL*** polje upisuje maksimalan broj prolaznih uređaja do odredišta. Svaki usmjerivač tijekom prihvaćanja paketa, mora ovaj brojač smanjiti za jedan. Maksimalna vrijednost ovog polja je 255, ali se ovisno o operativnom sustavu koristi neka vrijednost između 64 i 128. Ovaj mehanizam osigurava da paketi koji ne bi mogli stići do odredišta, ne bi vječno lutali mrežom, već se nakon isteka postavljenog brojača, odnosno broja prolazaka kroz određeni maksimalni broj usmjerivača, paket odbacuje na zadnjem usmjerivaču. S obzirom na to kako svaki usmjerivač smanjuje ovaj broj, te se paket zbog toga promijenio, usmjerivač stoga također mora ponovno izračunati novi *provjerni* zbroj (*checksum*) za *IP* zaglavlje paketa (kao i novi ***TTL***) te ga zapisati u novi paket.

Ova operacija ponovnog preračunavanja i zapisivanja *provjernog* zbroja mora se raditi za svaki pojedini paket. Naravno ova operacija je procesorski (CPU) zahtjevna.

Standardno postavljen ***TTL*** na operativnom sustavu Linux je obično **64**.
On je definiran i može se mijenjati u datoteci: `/proc/sys/net/ipv4/ip_default_ttl` odnosno u ***sysctl*** varijabli: `net.ipv4.ip_default_ttl`.



***Svaka aplikacija za sebe može preko socketa s kojim se spaja na mrežni stôg Linuxa postavljati svoju TTL vrijednost!***

Pogledajmo kako pomoću naredbe `tcpdump` dohvatiti sve mrežne pakete koji imaju TTL veći od primjerice 200.

Prvo ćemo i sami kreirati ovakve pakete pomoću naredbe `ping`, recimo s ***TTL*** postavljenim na 201.

```
ping -t 201 192.168.1.1
```

Sada ćemo u drugom terminalu s naredbom `tcpdump` dohvatiti sve mrežne pakete koji imaju TTL veći od 200:

```
tcpdump -n -v 'ip[8] > 200'
```

Pogledajmo kako smo filtrirali TTL polje iz IP zaglavlja. Naime TTL polje se nalazi u IP zaglavlju u 9. bajtu i veličine je jedan bajt. To znači da može imati najveću vrijednost 255 ($2^8$) iako se tolika vrijednost ne koristi. Stoga pretražujemo 8 bajt (**`ip[8]`**).



Vezano za detalje oko naredbe ***tcpdump***, pogledajte poglavlje:
**25.7.8. Naredba *tcpdump*.**

Pogledajte i detalje oko rada naredbe **ping (**i *ICMP* poruka**):**
**25.7.1. Naredba ping.**

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na IP sloju. Sintaksa ove naredbe koju ćemo korisiti jest:

```
nstat -opcije ImeBrojača
```

Primjerice, ako nas zanima brojač imena `IpInReceives`:

```
nstat -az IpInReceives
```

← U ispisu vidimo statistike za navedeni brojač `IpInReceives` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za **IP TTL** polje:

| Naziv brojača | Opis |
|---|---|
| **`TcpExtTCPMinTTLDrop`** | Iako je ovo TCP brojač on se odnosi na **IP TTL** polje. Ovaj brojač bilježi ukupan broj TCP segmenata koji su odbačeni jer je TTL vrijednost manja od minimalne TTL vrijednosti postavljene na sustavu. Ovaj brojač je koristan za praćenje segmenata (paketa) koji su odbačeni zbog problema s TTL-om, što može ukazivati na pogrešno konfigurirane mrežne postavke ili neželjenu mrežnu aktivnost. Naime kada IP paket putuje kroz mrežu, svaki *Layer 3* uređaj (*hop*) smanjuje **TTL** vrijednost za 1. Ako TTL vrijednost pada na ili ispod 0, paket se odbacuje, a obično se generira i ICMP poruka o isteku TTL-a. |

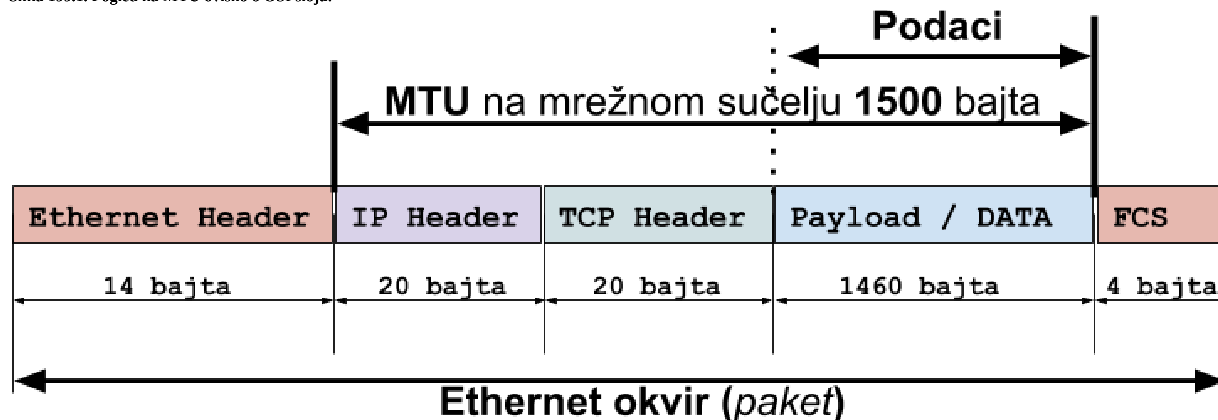
**Izvori informacija: (1022),(1044),(1366),(1367),(K-6), `man tcpdump`,`man nstat`,`man ping`,`man 5 procfs`, RFC791.**

## 23.4.2. Maximum Transmission Unit (*MTU*)

Svaki usmjerivač na svakoj svojoj mrežnoj kartici odnosno mrežnom sučelju ima postavljenu **MTU** veličinu (*Maximum Transmission Unit*). *MTU* definira maksimalnu količinu (veličinu) podataka bez zaglavlja, koji mogu stati u mrežni paket na OSI mrežnom sloju tri (OSI 3) odnosno IP sloju prema TCP/IP protokolu. Stoga se ***MTU*** nekada naziva i ***IP MTU***.

*MTU* veličina za lokalne *Ethernet* mreže je standardno 1500 *bajta* (RFC 894). Međutim za neke druge vrste mrežnih tehnologija on može biti drugačiji, poput primjerice za: A/S/HDSL, ADSL2+, ATM i druge, a pogotovo za neke vrste dodatnih ugnježđivanja (*enkapsulacija*), koje koriste VPN tuneli poput: *IPSec*, *OpenVPN* i sličnih. Pogledajmo *Ethernet* mrežni okvir na OSI slojevima 2 i 3. Uočite da se u standardno navedenih 1500 bajta nalaze: IP zaglavlje (20 bajta), TCP ili UDP zaglavlje (20 bajta) te je preostalo do maksimalno 1460 bajta za ostale informacije od viših aplikacijskih protokola te na kraju i sâme podatke (DATA).

**Slika 199.1. Pogled na MTU ovisno o OSI sloju.**






**MTU** vrijednosti postavljaju se obavezno na sva mrežna sučelja, kako fizička (pr. `eth0`, `eno1`), tako i logička (`eth0:0`, `eno1.10`, `veth0`, `br0`, …).
To znači da svako mrežno sučelje koje postoji na sustavu ima postavljenu **MTU** vrijednost!

Pogledajmo neke od ***MTU*** vrijednosti (**61**) ovisno gdje se koriste:

| MTU (bajta) | Vrsta konekcije |
|---|---|
| 1501- 9198 (ili više) | *Ethernet* v.2 - standardni *Ethernet* uz upotrebu "*Jumbo frame*" okvira. |
| 1500 | *Ethernet* v.2 - standardni *Ethernet MTU.* |
| 1480 | ***PPPoE*** *(Point-to-Point over Ethernet).* |
| 1460 | ***L2TP*** *(Layer 2 Tunneling Protocol).* |
| 1454 | ***PPPoE-over-DSL*** *(*najčešće*).* |
| 1372 | ***PPTP*** *(Point-to-Point Tunneling Protocol).* |

Pogledajmo i veličinu standardnog *ethernet* mrežnog okvira (veličine 1500 *bajta*) na OSI sloju dva (OSI 2):

| Veličina okvira (bajta) | Opis |
|---|---|
| 1518 | Veličina *Ethernet II* okvira.<br>Pogledajte poglavlje: **20.1. Oblik mrežnih okvira na OSI slojevima 2 i 1.** |
| 1522 | Prema *Ethernet* standardu **802.3ac** (**802.1Q**), s podrškom za **VLAN mreže**. |

To isto tako znači kako je na OSI sloju tri (OSI 3) odnosno na **IP** sloju prema TCP/IP protokolu, *MTU* veličina:

| Veličina okvira (bajta) | Opis |
|---|---|
| 1500 | Veličina okvira koji dolazi do *IP* (*OSI 3*) sloja [***MTU*** ili ***IP MTU***]. |
| 1480 | Veličina okvira na *IP* (*OSI 3*) sloju. |

Ako pogledamo primjerice *ICMP* paket, koji se oslanja na IP sloj, tada to izgleda ovako:

| Veličina okvira (bajta) | Opis |
|---|---|
| 1480 | Veličina okvira koji dolazi od *IP* (*OSI 3*) sloja, prema *ICMP* sloju. |
| 1472 | Veličina okvira koji koristi *ICMP*<br>(minimalna veličina *ICMP* zaglavlja je prema tome 8 *bajta*). |

Ili na još višem odnosno *TCP* sloju (*OSI 4*) onda imamo sljedeće stanje:

| Veličina okvira (bajta) | Opis |
|---|---|
| 1480 | Veličina okvira koji dolazi od *IP* (*OSI 3*) sloja. |
| 1460 | Veličina okvira koji dolazi sa *TCP* (*OSI 4*) sloja<br>(minimalna veličina *TCP* zaglavlja je prema tome 20 *bajta).* |

Postavljenu **MTU** vrijednosti na svim mrežnim sučeljima (prvi stupac) možemo vidjeti s naredbama: `ifconfig -s`, `netstat -i` i `ip -s link`, a za svako mrežno sučelje zasebno s naredbama: `ifconfig` ili `ip addr show`.

Pogledajmo MTU vrijednost s naredbom `ifconfig` (ispis smo skratili):

```
ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.125  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::a00:27ff:fe6c:caef  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:6c:ca:ef  txqueuelen 1000  (Ethernet)

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
```

← Vidimo da fizička Ethernet mrežna kartica `enp0s3` ima **MTU 1500**, a posebno sućelje povratne veze (*loopback*) `lo` ima **MTU 65536** bajta što je za ovo posebno sučelje zadana vrijednost.



**MTU** veličina se može promijeniti u konfiguracijskoj datoteci mrežnog sučelja.
Pogledajte kako to trajno napraviti (ako imate potrebe):
**25.6.5.1.2. Trajna (permanentna) ručna konfiguracija mreže.**
**26.1. Network Manager.**

## Zbog čega je važan MTU?

Računalo, ali i usmjerivač mogu na različitim mrežnim sučeljima, koja mogu biti spojena na različite vrste mreža, imati i različito postavljen ***MTU***. Kada mrežni paket (*datagram*) dođe na jedno mrežno sučelje usmjerivača, usmjerivač prvo provjerava svoju ***MTU*** veličinu postavljenu na tom mrežnom sučelju. Potom na osnovu tablice usmjeravanja odluči na koje mrežno sučelje će usmjeriti taj *paket*, te provjerava ***MTU*** vrijednost tog mrežnog sučelja.

Ako postoji razlika u ***MTU*** vrijednostima između mrežnog sučelja na kojem je *paket* zaprimljen i onoga na koje se preusmjerava, usmjerivač mora raditi takozvanu **IP fragmentaciju** odnosno rastavljanje paketa na manje *fragmente*. Ako je primjerice naš paket došao s mrežnog sučelja koje je na našoj *LAN* mreži, te ima *MTU* 1500 *bajta*, a odredište mu je na mreži na kojoj je mrežno sučelje za *VDSL*, koji koristi *PPPoE*, a koji ima *MTU* 1454 *bajta*, očito da svi podaci od izvornog paketa (*datagrama)* neće stati u novi *paket* jer je on manji za 46 bajta u odnosu na onaj u našoj mreži. Stoga usmjerivač mora razlomiti izvorni *paket* na dva nova paketa, od kojih će prvi sadržavati 1454 *bajta payload*, a drugi samo 45 *bajta payload* dijela (dio s viših slojeva).

Ova operacija se mora odrađivati za svaki pojedini mrežni paket kod kojega se mijenja ***MTU*** veličina vanjskog mrežnog sučelja usmjerivača. S druge strane na odredištu se radi suprotan proces sastavljanja. Ovaj proces je procesorski i memorijski zahtjevan za usmjerivač odnosno uređaj koji ga odrađuje.
**IP fragmentacija** je definirana u standardu **RFC 791**, dok je suprotan proces defragmentacije definiran u **RFC 815**.



Pogledajte i poglavlja:
**23.2. IP fragmentacija**
**23.2.1. Optimizacije vezane za fragmentaciju i MTU.**

Pogledajte sljedeće poglavlje te se prisjetite i zaglavlja IP paketa, u poglavlju:
**23.1. Oblik IP poruke (paketa).**

**Izvori informacija: (61),(62), (K-6), (K-10),** `man netstat`, `man ifconfig`, `man ip`, **RFC 791**, **RFC 815**, **RFC 894.**

### 23.4.2.1. MTU primjeri

Kako smo vidjeli iz gornjih tablica, veličina ***MTU*** se mijenja ovisno o *OSI* sloju, pa se tako standardni ***MTU*** od 1500 *bajta*, odnosi na ***MTU*** koji dolazi do OSI sloja tri. Na OSI sloju tri se nalazi *IP* protokol, koji dodaje minimalno 20 *bajta* svog zaglavlja, te dolazimo do maksimalno 1480 *bajta* dostupnih za više protokole. Najjednostavniji lako upotrebljivi primjer bi bio upotreba *ICMP* paketa, koje možemo slati naredbom `ping` za provjeru dostupnosti drugog računala na mreži. *ICMP* se oslanja na *IP* protokol, dakle nalazi se logički iznad njega, a on ima minimalno 8 *bajta* zaglavlja.

Tako dolazimo do sljedeće računice:

| **MTU prema IP sloju** (***MTU***$_{\text{koji vidimo na mrežnom sučelju}}$) | **MTU na IP sloju (IPv4)** | **MTU na ICMP sloju** |
|---|---|---|
| **1500 bajta** | 1480 *bajta* | 1472 *bajta* |

Pogledajmo i kako to izgleda, ako se koristi TCP protokol:

| **MTU prema IP sloju** (***MTU***$_{\text{koji vidimo na mrežnom sučelju}}$) | **MTU na IP sloju (IPv4)** | **MTU na TCP sloju** |
|---|---|---|
| **1500 bajta** | 1480 *bajta* | 1460 *bajta* |

Dokažimo kako su teorija i praksa jednake za *ICMP* poruke. Naredba `ping` na OSI sloju tri (**IP**) nudi nam mogućnost postavljanja *Don't Fragment (**DF**)* bita. To znači kako za *ICMP* (*ping*) paket koji šaljemo, zabranjujemo da bude *fragmentiran* na *IP* sloju. Dakle ovakav paket neće biti razlomljen na manje pakete (fragmente), ako mu je ***MTU*** paketa, odnosno njegova veličina veća od maksimalne ***MTU*** vrijednosti koja može biti poslana na mrežu.
Na mrežnoj kartici `eth0` (što je zapravo OSI sloj tri) postavljen je standardni **MTU** 1500 *bajta* pa pogledajmo kako to izgleda:

```
ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 08:00:27:93:E8:86
          inet addr:10.50.200.128  Bcast:10.50.200.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:26306 errors:0 dropped:0 overruns:0 frame:0
            TX packets:3259 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:8449333 (8.0 MiB)  TX bytes:854760 (834.7 KiB)
```

Vidimo kako je `MTU:1500` što je očekivano, kako smo i rekli. Isti ispis bi dobili i s novijom naredbom: `ip address`.



Prethodno vidljiva vrijednost ***MTU**a* se odnosi na **MTU** vrijednost **prema IP sloju**, kako je vidljivo u tablicama gore.

Sada ćemo probati poslati *ICMP* paket prema poslužitelju (192.168.100.254) i to veličine 1500 *bajta*, direktno na mrežu:

```
ping -c3 -M do -s 1500 192.168.100.254

PING 192.168.100.254 (192.168.100.254) 1500(1528) bytes of data.
ping: local error: Message too long, mtu=1500
ping: local error: Message too long, mtu=1500
ping: local error: Message too long, mtu=1500
--- 192.168.100.254 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2621ms
```

Vidljivo je nekoliko stvari:

- U drugom retku *ping* poruke se vidi: *`1500(1528)` bytes of data*, što znači kako je na 1500 *bajta* dodano još 28 *bajta* kako smo i očekivali.
- Nakon toga dobivamo greške (*`Message too long`*) jer se na našu mrežu ne može poslati više od 1500 *bajta* jer nam je ***MTU*** mrežne kartice s koje šaljemo pakete na mrežu ograničen na 1500 *bajta*.

Sada ćemo smanjiti veličinu paketa za 28 *bajta:* 20 *bajta* IP (IPv4) + 8 *bajta* ICMP, kako bi sve zajedno bilo u granici od 1500 *bajta*. Dakle šaljemo ICMP paket od 1472 *bajta* na sljedeći način te šaljemo ukupno tri *ICMP* paketa:

```
ping -M do -s 1472 192.168.100.254 -c3

PING 192.168.100.254 (192.168.100.254) 1472(1500) bytes of data.
1480 bytes from 192.168.100.254: icmp_seq=1 ttl=255 time=1.77 ms
1480 bytes from 192.168.100.254: icmp_seq=2 ttl=255 time=1.75 ms
1480 bytes from 192.168.100.254: icmp_seq=3 ttl=255 time=1.73 ms
--- 192.168.100.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2350ms
rtt min/avg/max/mdev = 1.735/1.755/1.773/0.015 ms
```

Na prethodnom ispisu su vidljive dvije stvari:

- Sada se šalje paket koji je na mreži ukupne veličine 1500 *bajta* - pogledajte drugi red:
  - `1472(1500) bytes of data` - to znači kako šaljemo 1472 *bajta*, a paket koji završava na mreži je veličine 1500 *bajta*.
- Nakon toga vidimo `1480 bytes from`, što znači da nam *ICMP* sloj govori kako on prema *IP* sloju šalje 1480 *bajta*, a u konačnici *IP (IPv4)* sloj dodaje još 20 *bajta*, što ukupno čini 1500 *bajta* na mreži, koji su prikazani u poruci iznad. ***Važno je znati da, ako koristimo IPv6 protokol, da IPv6 sloj dodaje 40 bajta u odnosu na 20 bajta kod IPv4***.

Pogledajmo kako izgledaju ove poruke (paketi) na mreži (skratili smo ispis).
Sada ćemo se fokusirati sâmo na poruku u kojoj naše računalo šalje *ICMP* upit (*ICMP request*):

```
1.  Frame 73: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
2.  Ethernet II, Src: (08:00:27:93:e8:86), Dst: (00:12:00:85:4c:c6)
3.  Internet Protocol Version 4, Src: 192.168.100.1, Dst: 192.168.100.254
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
      Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 1500
      Identification: 0x0000 (0)
  +   Flags: 0x02 (Don't Fragment)
          0... .... = Reserved bit: Not set
          .1.. .... = Don't fragment: Set
          ..0. .... = More fragments: Not set
      Fragment offset: 0
      Time to live: 64
      Protocol: ICMP (1)
      Header checksum: 0x903b [validation disabled]
      [Header checksum status: Unverified]
      Source: 192.168.100.1
      Destination: 192.168.100.254

4.   Internet Control Message Protocol
      Type: 8 (Echo (ping) request)
      Code: 0
      Checksum: 0x11bd [correct]
      [Checksum Status: Good]
```

Na OSI sloju dva (OSI 2 [`2.`]) vidimo izvorišnu i odredišnu MAC adresu to jest: `Src: (08:00:27:93:e8:86)` i `Dst: (00:12:00:85:4c:c6)`. Na OSI sloju tri (`3.`) odnosno na IP sloju su vidljive izvorišna IP adresa (`Src: 192.168.100.1`) te IP adresa odredišta (`Dst: 192.168.100.254`) to jest IP adresa na koju se ovaj mreži paket šalje.
Nadalje unutar OSI sloja tri (`3.`) to jest unutar IP zaglavlja vidimo polje `+ Flags`. Ovdje je vidljivo kako je bît: `Don't fragment` postavljen (odnosno u stanju `1`) te vidimo kako je naznačeno da je viši protokol ***ICMP***, prema poruci: `Protocol: ICMP`. Na sljedećem OSI sloju četiri (`4.`) je vidljiv upravo ***ICMP*** protokol, a u njegovom zaglavlju prema `Type 8` znamo kako se radi o *ICMP* poruci: ***ping request***. Nakon ove naše poruke, uslijedio bi odgovor od poslužitelja (192.168.100.254), a poruka bi bila: ICMP ***ping reply*** (`Type 0`).



Za primjere konfiguracije odnosno postavljanja nove **MTU** vrijednosti pogledajte sljedeće poglavlje.



Za upotrebu standardnog mehanizama za dogovaranje **MTU** veličine, pogledajte poglavlje:
**23.2.1.1. Path MTU Discovery.**

Prema određenim postavkama moguće je aktivirati mehanizme za dogovaranje ***MTU*** veličine i preko TCP protokola (tzv. *TCP Packetization-Layer Path MTU discovery* [**RFC 4821**]). Neke od *sysctl* varijabli zaduženih za dogovaranje ***MTU*** veličine preko TCP protokola su: `net.ipv4.tcp_mtu_probing` (za aktiviranje) te `net.ipv4.tcp_base_mss` za definiranje početne ***MSS*** veličine. Prema navedenom ***RFC*** standardu, dostupne su i sljedeće1 *sysctl* varijable odnosno opcije:

- `net.ipv4.tcp_probe_interval` – ovdje se definira svakih koliko minuta će se slati *TCP path MTU discovery* poruke. Standardno je 600 sekundi (10 minuta).
- `net.ipv4.tcp_probe_threshold` – nakon koliko bajta se zaustavlja TCP *Path MTU Discovery probing*.

**Izvor informacija:** **`man ping`**,**`man ifconfig`**,**`man ip`**,**`man 7 ip`**,**`man 7 tcp`**,**RFC 791**, **RFC 894**, **RFC 4821**

### 23.4.3. Veliki mrežni okviri (*Jumbo frames*)

**Slijedi napredno poglavlje!**
Govorili smo kako je u LAN mrežama, veličina ***MTU***-a prema OSI sloju tri (**IP**) standardno 1500 bajta. Međutim moguće je definirati i koristiti i znatno veće mrežne okvire koji se nazivaju ***Jumbo frames*** (*jumbo* okviri). Naime svaki mrežni okvir s više od 1500 bajta, smatra se *jumbo* okvirom, od engleske riječi *Jumbo*: koja označava vrlo veliku osobu ili pojam odnosno stvar.

Konvencionalno jumbo mrežni okviri mogu nositi do 9000 bajta korisnog sadržaja, ali postoje varijacije, stoga je potrebno paziti na njegovu maksimalnu veličinu. Naime mnogi *Gigabit Ethernet (1.000 Mbps* odnosno 1*Gbps)* preklopnici i *Gigabit Ethernet* mrežne kartice mogu podržati *jumbo* okvire. Naravno i većina bržih mrežnih kartica i mrežnih uređaja (1/2.5/10/25/100+ *Gbps*) također podržava jumbo mrežne okvire. Isto tako i neki *Fast Ethernet* (100*Mbps*) preklopnici i *Fast Ethernet* mrežne kartice također mogu podržavati *jumbo* okvire. Naime svaki *Ethernet* okvir koji se prima ili koji šaljemo na mrežu mora se i obraditi te pripremiti za primanje ili za slanje. Isto tako svaki mrežni okvir koji prolazi mrežom mora biti obrađen dok prolazi kroz mrežu za što su naravno potrebni resursi.

Dakle svaki mrežni okvir je potrebno zaprimiti, pročitati mu zaglavlje, ponovno izračunati *provjerni* zbroj (*checksum*) te usporediti s izvornim *provjernim* zbrojem, kao i odraditi dodatne provjere unutar paketa. Sve u svemu potrebno je procesiranje, koje zahtjeva resurse računala (uglavnom CPU i RAM) i mrežnih uređaja. Iz ovoga je jasno kako je slanje velikog broja mrežnih okvira veličina 1500 bajta (prema OSI sloju tri) zahtjevno (CPU+RAM) te, ako bi se veličina mrežnog okvira povećala, da bi se time smanjio i broj mrežnih okvira koje bi bilo potrebno poslati na mrežu ili primiti s nje. Upravo zbog toga su i izmišljeni veliki mrežni okviri (*jumbo* okviri).

Zamislimo da sâmo trebamo prenijeti 9000 bajta prema OSI sloju tri (OSI 3) preko mreže.
- Za to bi standardnim mrežnim okvirima veličine 1500 bajta trebalo šest (6) mrežnih okvira, a za svaki od njih treba izračunati provjerni zbroj, napraviti zaglavlje i poslati ih na mrežu, prolaskom kroz mrežu istu stvar mora napraviti i mrežni preklopnik kao i drugi uređaji na mreži (barem što se tiče procesiranja, u ovom slučaju većeg broja paketa).
- Za to bi nam trebao samo jedan (1) veliki odnosno *jumbo* okvir.

Ova veličina od 9000 bajta se odnosi na veličinu mrežnog okvira prema OSI sloju tri (OSI 3), što znači sljedeće:

| **Na sloju mreže (OSI 1)** | **Na MAC sloju (OSI 2)** | **Prema IP sloju (prema OSI 3) (*MTU* *koji vidimo na mrežnom sučelju*)** | **Na IP sloju (OSI 3)** | **Na TCP sloju (OSI 4)** |
|---|---|---|---|---|
| *9038 bajta* | *9018 bajta* | *9000 bajta* | *8980 bajta* | *8960 bajta* |

Dakle, kako smo već govorili;
- Mrežni okvir prema OSI sloju tri, veličine je 9000 bajta, o ovdje govorimo o **MTU** veličini.
- Mrežni okvir na IP sloju (OSI sloj tri) ima zaglavlje od 20 bajta (9000-20=8980) pa imamo 8980 bajta za korisne podatke prema višem sloju (koji se već koristi).
- Ako primjerice koristimo **TCP** protokol kao transportni protokol, on dodaje svoje zaglavlje od 20 bajta (8980-20=8960) pa imamo 8960 bajta za korisne podatke prema aplikacijskim slojevima.

U konačnici ako gledamo niže OSI slojeve, na OSI sloju dva (OSI 2) dodaje se MAC zaglavlje od 18 bajta, pa naš mrežni okvir veličine 9000 bajta raste na 9018 bajta. Kada ovakav mrežni okvir dođe na najniži sloj (sloj mreže) odnosno OSI sloj jedan ovdje se dodaje još jedno zaglavlje veličine 20 bajta pa mrežni okvir raste na 9039 bajta te se takav i konačno šalje na mrežu.

I ovdje je moguće koristit VLAN-ove pa, ako ih koristimo, dodaju se još 4 bajta za VLAN-ove pa sada imamo sljedeće stanje:

| **Na sloju mreže (OSI 1)** | **Na MAC sloju (OSI 2)** + VLAN 802.1Q tag | **Prema IP sloju (prema OSI 3) (*MTU* *koji vidimo na mrežnom sučelju*)** | **Na IP sloju (OSI 3)** | **Na TCP sloju (OSI 4)** |
|---|---|---|---|---|
| *9042 bajta* | *9022 bajta* | *9000 bajta* | *8980 bajta* | *8960 bajta* |

Došli smo do toga da, ako koristimo i VLAN-ove, konačni mrežni okvir koji se šalje na mrežu raste na veličinu od 9042 bajta. Kako ručno postaviti ***MTU*** na veličinu 9000 bajta, odnosno kako uključiti *jumbo* okvire na mrežnoj kartici, ako to uopće mrežna kartica hardverski podržava. Prvo pogledajmo trenutno postavljeni ***MTU*** na mrežnoj kartici `eth0` s naredbom `ip`:

```
ip link show eth0 | grep mtu
2: eth0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT qlen 1000
```

Vidimo kako je postavljen na `1500`. Postavljene *MTU* vrijednost za mrežne kartice možemo vidjeti i s `netstat -i`.
Sada postavimo mrežnoj kartici `eth0` *MTU* veličinu na `9000` na sljedeći način:

```
ip link set eth0 mtu 9000
```

Za trajnu konfiguraciju mrežne karticu `eth0` dodajte u datoteku: `/etc/sysconfig/network-scripts/ifcfg-eth0` sljedeću konfiguraciju odnosno sljedeći redak konfiguracije:

```
MTU=9000
```



Za trajnu konfiguraciju ***MTUa***, mrežnog sučelja pogledajte poglavlje:
**25.6.5.1.2. Trajna (permanentna) ručna konfiguracija mreže.**



Upotreba ***Jumbo MTU*** povećava propusnost i stoga jer se za istu količinu prenesenih podataka smanjuje broj mrežnih paketa. Vezano za ovu problematiku, pogledajte poglavlje: **20.4. *Gbps* i *Mpps* – u čemu je veza**, te poglavlja koja slijede.



Važno je razumjeti kako *jumbo* ***MTU*** (9000) mora biti jednako postavljen na svim mrežnim uređajima, a prvenstveno preklopnicima (*switchevima*) i računalima koja komuniciraju na istoj lokalnoj mreži. **Pri tome se na preklopnicima podrška za jumbo mrežne okvire obično konfigurira na razini cijelog preklopnika**. Dakle svi u komunikaciji bi morali imati isti ***MTU***; barem u lokalnoj mreži i to na mrežnim karticama za koje želimo koristiti *jumbo* mrežne okvire.

*Ako jumbo **MTU** (~9000) koristimo unutar virtualizacije, treba biti ekstremno oprezan da sva mrežna sučelja imaju postavljen ispravan MTU. Naime razne virtualizacijske platforme (pr. **OpenStack**) nerijetko imaju i desetak mrežnih slojeva; od virtualnih do raznih logičkih slojeva, od kojih potencijalno svaki od njih može imati drugačiji MTU. To može uzrokovati ili fragmentaciju (i više opterećenje na sustav) ili odbacivanje mrežnih paketa, ovisno kako su ovi slojevi virtualizacije konfigurirani!.*

Probajmo konfigurirati *jumbo* okvire i na računalu s druge strane s kojim želimo komunicirati, spojenom na isti preklopnik, na kojem smo uključili podršku za *jumbo* okvire. Test ćemo napraviti slanjem *ping* poruke (ICMP paketa) u kojoj ćemo u IP zaglavlju uključiti takozvani „***don't fragment bit***" koji označava kako niti jedan uređaj u prolazu ne smije raditi fragmentaciju mrežnih okvira koje upravo šaljemo s naredbom *ping*. To opet znači kako ćemo slati mrežne okvire određene veličine te želimo da mrežni okviri točno te veličine i budu dostavljeni drugoj strani, bez razlamanja na manje okvire, prolaskom kroz uređaje koji eventualno ne podržavaju *jumbo* okvire. Kako smo već naučili: *ICMP* se oslanja na *IP* protokol, dakle nalazi se logički iznad njega, a on ima minimalno 8 *bajta* zaglavlja. IPv4 zaglavlje je 20 bajta, a ako se koristi IPv6 tada je zaglavlje 40 bajta. *Dakle ICMP ne koristi TCP ili UDP kao transportne protokole*.
Vezano za veličinu mrežnog okvira, dolazimo do sljedeće računice:

| MTU prema IP sloju (***MTU****koji vidimo na mrežnom sučelju*) | MTU na IP sloju (IPv4) | MTU na ICMP sloju |
|---|---|---|
| 9000 *bajta* | 8980 *bajta* | 8972 *bajta* |

To znači kako *ICMP* paket smije biti maksimalno **8972** bajta veličine, ili će biti odbačen, prvo od sâme mrežne kartice, a ako bi i nekako prošao do preklopnika, preklopnik bi ga odbacio kao prevelik.
Sada pošaljimo *ping* poruku s *don't fragment bit*om prema drugom računalu: **192.168.1.100** (u primjeru šaljemo tri paketa):

```
ping -M do -s 8972 192.168.1.100 -c3
PING 192.168.1.100 (192.168.1.100) 8972(9000) bytes of data.
8980 bytes from 192.168.1.100: icmp_seq=1 ttl=64 time=0.118 ms
8980 bytes from 192.168.1.100: icmp_seq=2 ttl=64 time=0.146 ms
8980 bytes from 192.168.1.100: icmp_seq=3 ttl=64 time=0.118 ms
--- 192.168.1.100 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.118/0.138/0.173/0.027 ms
```

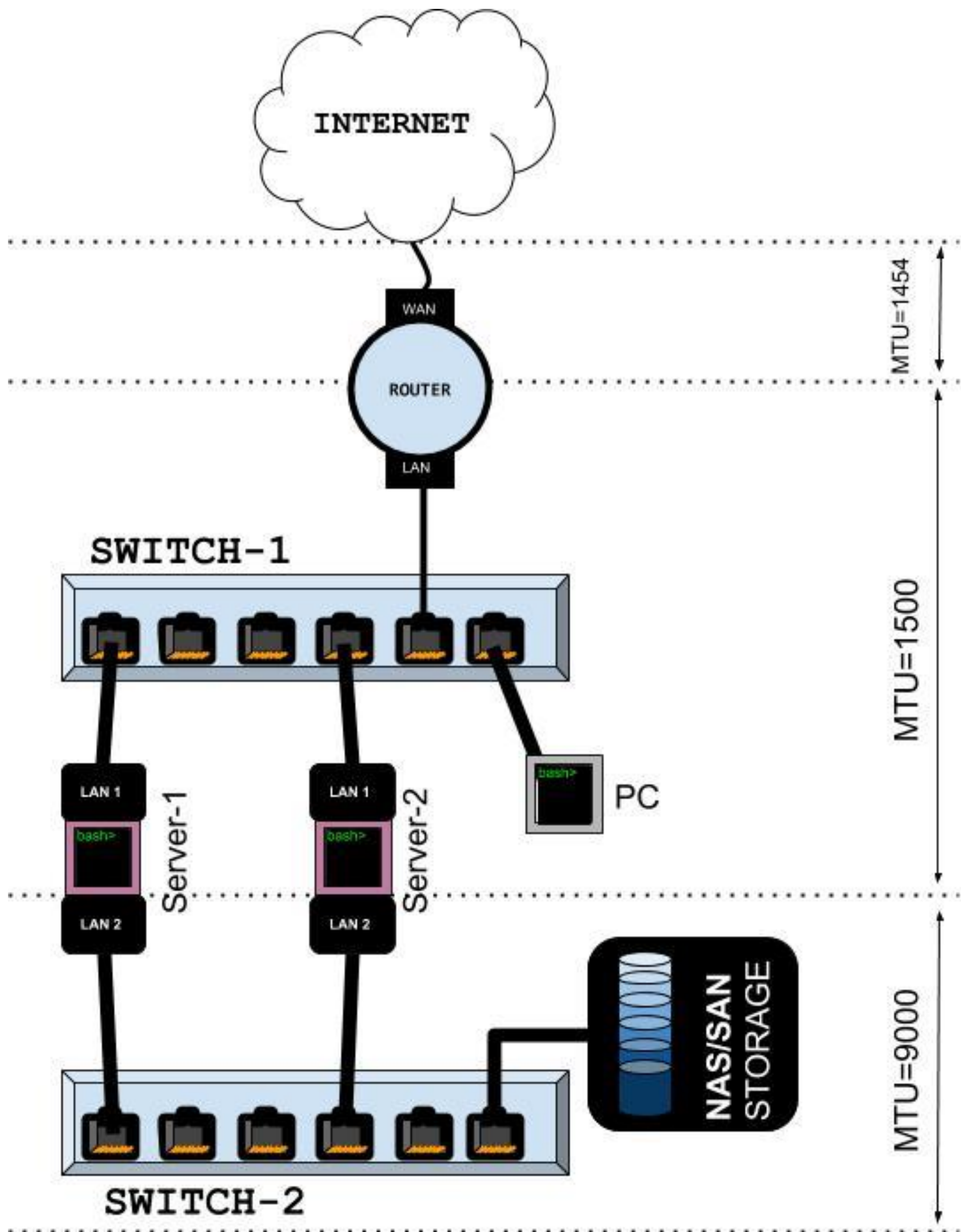
Vidimo kako je sve uredno prošlo bez problema. Kada bi međutim pokušali poslati preveliki *ICMP* paket, a to bi na razini *ICMP*-a u ovom slučaju bilo `8973` bajta, dobili bi:

```
ping -M do -s 8973 192.168.1.100 -c3
PING 192.168.1.100 (192.168.1.100) 8972(9000) bytes of data.
ping: local error: Message too long, mtu=9000
ping: local error: Message too long, mtu=9000
ping: local error: Message too long, mtu=9000
--- 192.168.1.100 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
```

Vidimo kako su u ovom slučaju svi paketi odbačeni kao preveliki (`Message too long`). S ovime smo dokazali kako svi mrežni uređaji između ova dva računala u komunikaciji uključujući i oba računala, podržavaju jumbo okvire veličine 9000 bajta.

Slika 199 prikazuje jednu od mogućih primjena *jumbo* mrežnih okvira.

**Slika 199.2 Razne vrijednosti MTUa u primjeni.**




Na slici 199.2 je vidljivo kako je računalo (**PC**) spojeno na preklopnik (**SWITCH-1**) koji je konfiguriran da koristi standardni ***MTU*** 1500, a koji je potom spojen na usmjerivač koji sa svojim ***LAN*** sučeljem (donjom mrežnom karticom) koristi isto standardni ***MTU*** 1500. Usmjerivač (**ROUTER**) s gornje strane ima WAN mrežno sučelje koje, ako je primjerice spojeno na *DSL* i koristi *PPPoE*, ima ***MTU*** veličine 1454 bajta.

Sva komunikacija računala prema lokalnoj mreži i svim računalima na njoj, kao i prema dva poslužitelja; **Server-1** i **Server-2,** ali samo preko njihove prve mrežne kartice **LAN 1** preko koje su spojeni na ovaj preklopnik, koristi ***MTU*** veličine 1500 bajta.

Oba ova poslužitelja (**Server-1** i **Server-2**) imaju i dodatnu mrežnu karticu (**LAN 2**) preko koje su spojeni na drugi preklopnik koji ima konfiguriran ***MTU*** 9000, a na koji se spajaju mreža i uređaji kojima je potrebna ovakva brža komunikacija koja zahtjeva ***MTU*** 9000, poput uređaja za pohranu i dijeljenje podataka preko mreže, kao što su ***NAS*** ili ***SAN*** sustavi. Na ovaj drugi (donji) preklopnik (**SWITCH-2**) je stoga i spojen ***NAS/SAN*** sustav za mrežnu pohranu i dijeljenje podataka (**NAS/SAN STORAGE**) preko kojega oba poslužitelja brže mogu pristupati dijeljenim podacima; obično datotekama (***NAS***) ili blokovima podataka odnosno diskovima (***SAN*** sustav).

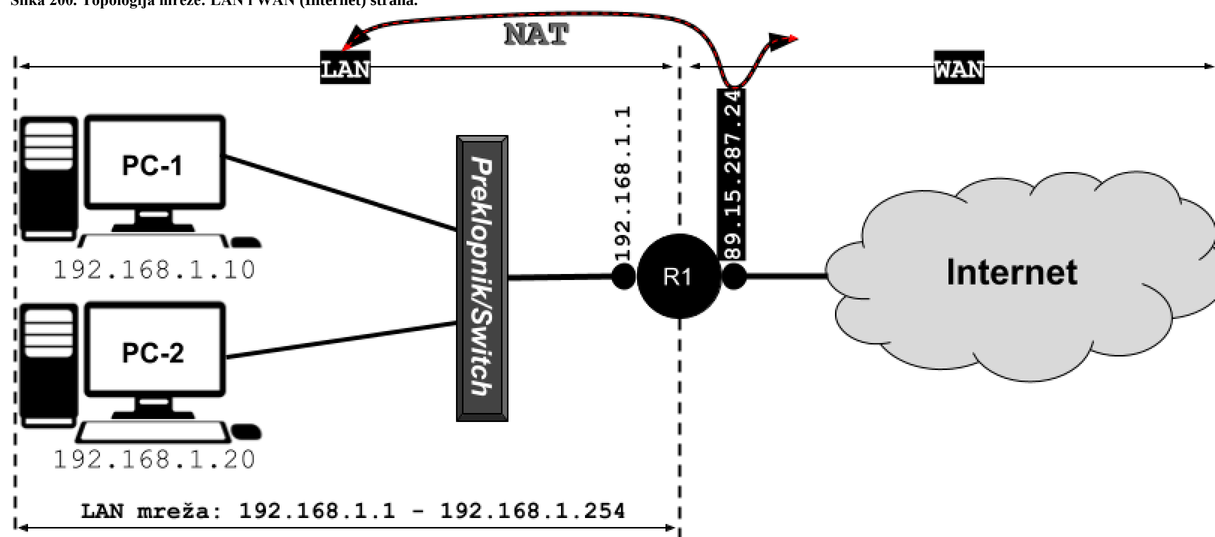
**Izvori informacija: (61),(62),(517),(518),(525),(526),(527),(528),(K-6),(K-10), `man ping`, RFC 791.**

## 23.4.4. Translacija adresa (*NAT*)

Translacija odnosno prevođenje adresa znano i kao *Network address translation* **(NAT)** se odnosi na metodu preslikavanja jednog IP adresnog prostora odnosno opsega IP adresa, u drugi. Dalje u tekstu, govorit ćemo o sâmo jednoj od ***NAT*** metode, koja je najčešće u upotrebi. Dakle radi se o preslikavanju IP adresa s jedne mreže u kojoj imamo jedan IP adresni prostor na drugu mrežu s drugim IP adresnim prostorom. Ovu operaciju odrađuje usmjerivač, tako što na razni IP protokola unutar njegovog zaglavlja mijenja IP adrese i naravno ponovno izračunava *provjerni* zbroj na razini *IP* protokola (OSI sloj tri), za svaki takav paket (*datagram*) koji mora izaći na vanjsku mrežu (pr. internet). Pojednostavljeno ova metoda se koristi za skrivanje i zamjenu cijelog unutarnjeg IP adresnog prostora u lokalnoj (**LAN**) mreži s javnom (vanjskom) IP adresom našeg usmjerivača. Možemo to reći i ovako: ovom metodom preslikavamo mnoge (ili sve) privatne unutarnje IP adrese u lokalnoj mreži (***LAN***), u jednu jedinu javnu vanjsku IP adresu dostupnu s interneta.

Dakle računala na lokalnoj mreži dijele jednu vanjsku, javnu IP adresu. Naime kada računalo u lokalnoj mreži želi komunicirati s drugim računalom ili poslužiteljem na internetu, komunikacija prolazi kroz usmjerivač. Obično u lokalnoj mreži imamo adresni opseg IP adresa poput: 192.168.1.1 do 192.168.1.254, dok je javna IP adresa ona koju nam dodjeljuje telekom (*ISP*), a ona se dodjeljuje vanjskom (**WAN**) mrežnom sučelju usmjerivača, poput primjera na slici **200**.
Sâmo ta javna IP adresa (pr. IP: **89.15.287.24**) je dostupna svima s interneta.

**Slika 200. Topologija mreže: LAN i WAN (Internet) strana.**




U ***NAT*** načinu rada, naš usmjerivač za svaki naš mrežni paket na lokalnoj mreži (***LAN***) koji izlazi van, mijenja izvorišnu IP adresu (*Source IP*) sa svojom vanjskom (javnom) IP adresom, skrivajući našu stvarnu privatnu (unutarnju) IP adresu.
U konkretnom slučaju bi se u komunikaciji računala (PC-1) prema internetu, IP adresa prvog računala (192.168.1.10) zamaskirala kao vanjska IP adresa usmjerivača (89.15.287.24). Isto tako bi se i IP adresa bilo kojeg drugog računala na lokalnoj mreži, koje izlazi na vanjsku mrežu (internet), zamaskirala kao vanjska IP adresa usmjerivača (u konkretnom primjeru 89.15.287.24).
Ova metoda je potrebna jer inače mrežni paketi nikada ne bi našli put nazad do našeg usmjerivača, pošto bi im izvorišna IP adresa bila naša privatna IP adresa, poput 192.168.1.10, koja nije javno registrirana, niti je dostupna ili poznato njeno odredište. U toku ovog postupka usmjerivač mora zapamtiti s koje je stvarne unutarnje IP adrese (s **LAN** mreže) izvorno došao paket, kako bi ga znao i vratiti nazad.

Kako se za svaki paket koji izlazi (pr. prema internetu) mijenja izvorišna IP adresa, usmjerivač mora pratiti svaku pojedinu vezu: izvor-odredište, zvanu *sesija*. Stoga kako bi kada se paket vrati nazad, usmjerivač znao kojem računalu (IP adresi) u lokalnoj mreži pripada. Usmjerivač stoga gradi takozvanu *NAT* tablicu za svaku novu konekciju (sesiju) i to tako da povezuje originalnu izvorišnu i odredišnu IP adresu te koristi *TCP* ili *UDP* portove (ovisno je li se koristio TCP ili UDP protokol za prijenos) kao jedinstveni identifikator svake zasebne (pojedine) konekcije odnosno sesije.

Ova metoda povezivanja se zove ***PAT*** (Engl. *Port Address Translation*). **Sesija** označava ostvarenu komunikacijsku vezu između dvije krajnje točke. Tako je jedna sesija veza između računala **A** i **B** ostvarena upotrebom HTTP protokola, dok je već druga sesija veza između računala **A** i **B** ili **C** i **D** ostvarena primjerice upotrebom protokola HTTPS (ili nekog drugog).

Dakle *sesijom* možemo nazvati svaki komunikacijski kanal između dva sugovornika (računala) koji razgovaraju istim komunikacijskim protokolom (pr. HTTP, SSH, HTTPS, … ili TCP, …). U povratku paketa s interneta nazad, usmjerivač za svaki mrežni paket mora zamijeniti svoju vanjsku IP adresu sa stvarnom unutarnjom IP adresom računala u lokalnoj mreži kojemu je paket i bio namijenjen. Za ovu operaciju se također koristi ***NAT*** tablica. Dodatno postoji još cijeli nîz mehanizama koji su zaduženi za svaku pojedinu *sesiju*: bilo otvorenu, u procesu zatvaranja ili u nekom drugom (među)stanju.



Za način kreiranja i upotrebe ***efemernih*** portova, pogledajte poglavlje: **19.8.1. TCP/UDP Portovi.**
Za primjere konfiguracije ***NAT*** pravila, pogledajte poglavlje: **26.7.2.1. Primjeri.**



Detalji ***NAT****iranja* su opisani u RFC 2663. Ovakva metoda rada naravno zahtjeva određene resurse usmjerivača.

Navedena *NAT* metoda se često naziva i:

- ***One-to-many NAT ili Port address translation (PAT).***
- ***IP masquerading ili NAT overload odnosno Many-to-one NAT.***

Druga česta metoda rada ***NAT***-a je ona u kojoj se radi translacija jedne unutarnje IP adrese na jednu vanjsku adresu (**1:1**) pa se naziva i ***1-to-1 NAT***. To bi u ovom primjeru značilo da bi se recimo jedna unutarnja IP adresa (pr. 192.168.1.10) trajno preslikavala kao vanjska IP adresa: 89.15.287.24, što se ponekad koristi za mrežne poslužitelje, koji bi izvana uvijek bili dostupni s fiksnom javnom (***WAN***) adresom, iako bi u stvarnosti bili u lokalnoj (***LAN***) mreži sa svojom lokalnom IP adresom.

**Izvori informacija: (794), (K-6), (K-10), RFC 2663, pogledajte primjer na posteru/plakatu s primjerima: P4**.

## 23.4.4.1. Source NAT (SNAT)

Kako smo vidjeli, klasična upotrebe ***NAT***-a obično se koristi za skrivanje i zamjenu cijelog unutarnjeg IP adresnog prostora u lokalnoj (**LAN**) mreži s javnom to jest vanjskom IP adresom našeg usmjerivača. Možemo to reći i ovako: ovom metodom preslikavamo mnoge (ili sve) privatne unutarnje IP adrese u lokalnoj mreži (***LAN***), u jednu jedinu javnu vanjsku IP adresu dostupnu s interneta. Dakle računala u lokalnoj mreži dijele jednu vanjsku, javnu IP adresu. Naime kada računalo u lokalnoj mreži želi komunicirati s drugim računalom ili poslužiteljem na internetu, komunikacija prolazi kroz usmjerivač.

Obično u lokalnoj mreži imamo adresni opseg IP adresa poput: 192.168.1.1 do 192.168.1.254, dok je javna IP adresa ona koju nam dodjeljuje telekom (*ISP*), a ona se dodjeljuje vanjskom (**WAN**) mrežnom sučelju usmjerivača, poput primjera na slici **200.A.** Sâmo ta javna IP adresa (pr. IP: **89.15.287.24**) je dostupna svima s interneta i upravo ona se koristi za komunikaciju s vanjskim svijetom odnosno servisima i uslugama na javnoj mreži (***WAN***) odnosno na internetu. Zamislimo stanje kao na slici dolje (**200.A.**), ali bez novog usmjerivača u donjem dijelu slike.
Ako naše računalo (**PC-1**) s IP adresom ***192.168.1.10*** želi doći do web stranice kojoj na internetu pristupamo preko njene IP adrese: 10.10.10.10, na TCP portu 80, događa se sljedeće. Naš usmjerivač (**R1**) će prihvatiti naš zahtjev i obično pomoću klasične ***NAT*** tehnologije preuzeti naš zahtjev tako da će sakriti našu IP adresu i umjesto nje staviti svoju vanjsku IP adresu (***89.15.287.24***) te će se nje poslati mrežni paket prema odredišnoj IP adresi.

U slučaju da je ta odredišna IP adresa stvarni Web poslužitelj on će nam odgovoriti nazad, na IP adresu vanjskog sučelja **R2** usmjerivača. On kada zaprimi taj paket, na osnovu svoje ***NAT*** tablice znat će da je taj odgovor za računalo ***PC-1*** s IP adresom ***192.168.1.10*** na koju će i vratiti ovaj paket. Slično je i da se umjesto web poslužitelja zahtjev poslao na takozvani ***Load Balancer*** uređaj, koji bi tada taj zahtjev proslijedio na jedan od stvarnih Web poslužitelja.

Tada bi taj web poslužitelj, primjerice Web poslužitelj 1, zahtjev vratio prema svom podrazumijevanom usmjerivaču (*Default Gateway*), koji bi u ovom slučaju bio upravo sam ***Load balancer***.
On bi tada ponovno napravio ***NAT*** sakrivajući stvarnu (unutarnju) IP adresu stvarnog web poslužitelja (***192.168.2.10***), i postavljajući svoju vanjsku javnu IP adresu (***10.10.10.1***) kao izvorišnu, s kojom bi paket poslao nazad prema našem usmjerivaču **R1** to jest prema njegovoj javnoj IP adresi (***89.15.287.24***) s koje je paket i došao.

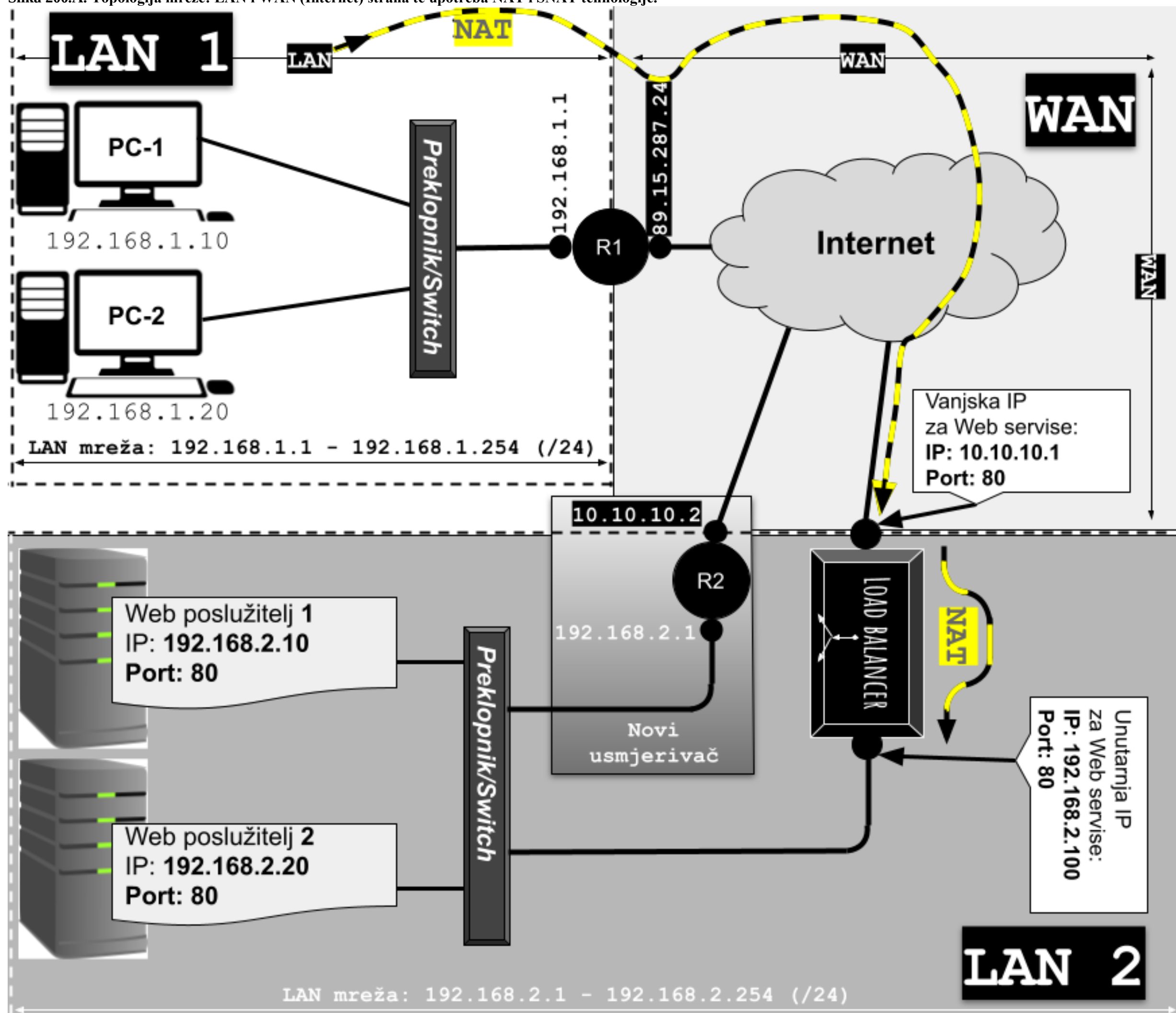
Međutim zamislio scenarij na slici **200.A.** u kojem smo u donju **LAN** mrežu postavili novi usmjerivač (***R2***) te na strani web poslužitelja postavili da im je podrazumijevani usmjerivač sada unutarnja IP adresa usmjerivača **R2** (***192.168.2.1***).
Tada bi pri odgovoru, web poslužitelj, pošto je upit došao s njemu vanjske i nepoznate IP adrese (***89.15.287.24***), on odgovor vratio na podrazumijevani usmjerivač s IP adresom (***192.168.2.1***), a ne na ***Load Balancer*** (***192.168.2.100***).



Ovakav problem se zove problem ***asimetrične*** putanje!

Paket dolazi kroz *Load Balancer*, ali pošto on nije podrazumijevani usmjerivač (*Default Gateway*) web poslužitelju, povratni paket će biti poslan na usmjerivač **R2**. Usmjerivač **R2** će ga uredno vratiti prema našem računalu koje je i iniciralo komunikaciju, ali pošto paket dolazi s druge (krive) adrese (ne više s *Load Balancera* na koji je bio poslan) on će biti odbačen jer naše računalo očekuje odgovor od strane s kojom se komunicira, a to je bila IP adresa **10.10.10.1**.

Slika 200.A. Topologija mreže: LAN i WAN (Internet) strana te upotreba NAT i SNAT tehnologije.




## *Source NAT (SNAT) kao rješenje*

Rješenje je ili vratiti podrazumijevani usmjerivač za web poslužitelje na unutarnju IP adresu ***Load balancera*** (***192.168.2.100***) ili uvođenje tehnologije translacije izvorišne IP adrese, koje se zove ***Source NAT*** odnosno ***SNAT***. Naime u složenijim mrežama postoji više razloga zbog kojih web poslužiteljima ***Load Balancer*** ne bi bio postavljen kao podrazumijevani usmjerivač: zbog nadogradnji programa i sustava, pristupa za održavanje i slično.

***Pogledajmo rješenje ovog problema pomoću iptables sustava, kroz primjere konfiguracije:***

**Upotreba SNAT-a**

Sa sljedećom konfiguracijom ćemo napraviti **SNAT** pravilo koje će za sav vanjski promet koji ulazi u lokalnu mrežu iz **WAN** sučelja `eth0` s IP: `10.10.10.1`, preko **LAN** sučelja `eth1` promijeniti izvorišnu (*source*) IP adresu u adresu `eth1` sučelja: `192.168.2.100`, tako da će računalima odnosno konkretno poslužiteljima (***Web poslužitelj 1*** i ***Web poslužitelj 2***) iza ovog Linux ***Load balancera*** izgledati kao da promet dolazi s njega to jest s IP adrese `192.168.2.100`.

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to 192.168.2.100
```

***Upotreba SNAT POOL-a***

Na sustavima koji moraju održavati veliki broj konekcija (60.000+), upotreba samo jedne jedine IP adrese kao ***SNAT IP*** adrese često nije dovoljna, Naime zbog ograničenja broja dostupnih portova; teoretski 65.535, za svaku IP adresu.
Stoga je jedino moguće koristiti više IP adresa i tako povećati sveukupan broj aktivnih konekcija.
Za to se koristi mehanizam koji se zove ***SNAT pool*** odnosno bazen/spremište s dostupnim IP adresama za ***SNAT***.
Dakle moguće je koristiti više IP adresa istovremeno kao ***SNAT pool***, za povećanje broja mogućih aktivnih konekcija, ali pri tome ne možemo koristiti primarnu IP adresu mrežnog sučelja**.**
Zamislimo ***SNAT pool*** u kojem je opseg IP adresa: 192.168.2.2→192.168.2.20; koji se konfigurira ovako:

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to 192.168.2.2-192.168.2.20
```

Tako će se umjesto klasičnog ***SNAT***-a i jedne jedine IP adrese moći koristiti konkretno 18 IP adresa iz **SNAT poola**.

***Također imajte na umu da će SNAT koristiti praćenje IP veze, pa provjerite je li vaša*** ip conntrack ***varijabla (dakle sysctl varijabla*** `net.netfilter.nf_conntrack_max`***) konfigurirana na odgovarajuću veličinu.***
***Na primjeru iz prethodne slike pogledajmo dio komunikacije: PC-1 → Web servisi (Load Balancer) → Web poslužitelj-1***

**Pogledajmo mrežni paket odnosno što koji uređaj radi s njim u slučaju upotrebe samo *NAT-a (i očito DNAT-a).***

→ *Odlazni paket*

| **Adresa i port** | **PC – 1**<br>*Korak 1* → | **R1**<br>**(radi NAT)**<br>*Korak 2* → | **Load Balancer**<br>**(radi NAT)**<br>*Korak 3* → | **Web poslužitelj -1**<br>*(zaprima paket)*<br>*Korak 4* |
|---|---|---|---|---|
| Source IP<br>Source Port | 192.168.1.10<br>4500 | **89.15.287.24**<br>4500 | **89.15.287.24**<br>4500 | **89.15.287.24**<br>4500 |
| Destination IP<br>Destination Port | 10.10.10.1<br>80 | 10.10.10.1<br>80 | **192.168.2.10**<br>80 | **192.168.2.10**<br>80 |

← *Povratni paket*

| **Adresa i port** | **Web poslužitelj -1**<br>*(vraća paket)*<br>*Korak 1* → | **Load Balancer**<br>**(provjerava NAT tablicu)**<br>*Korak 2* → | **R1**<br>**(provjerava NAT tablicu)**<br>*Korak 3* → | **PC – 1**<br>*Korak 4* →<br>***(zaprimanje odgovora)*** |
|---|---|---|---|---|
| Source IP<br>Source Port | 192.168.2.10<br>80 | **10.10.10.1**<br>80 | **10.10.10.1**<br>80 | **10.10.10.1**<br>80 |
| Destination IP<br>Destination Port | 89.15.287.24<br>4500 | **89.15.287.24**<br>4500 | 192.168.1.10<br>4500 | 192.168.1.10<br>4500 |

**Pogledajmo i mrežni paket odnosno što koji uređaj radi s njim, u slučaju upotrebe *NAT*-a (od strane R1 usmjerivača) te *SNAT-a* od strane *Load Balancera. Ovdje se odrađuje i DNAT (o njemu u jednom od sljedećih poglavlja).***

→ *Odlazni paket*

| **Adresa i port** | **PC – 1**<br>*Korak 1* → | **R1**<br>**(radi NAT)**<br>*Korak 2* → | **Load Balancer**<br>**(radi SNAT)**<br>*Korak 3* → | **Web poslužitelj -1**<br>*(zaprima paket)*<br>*Korak 4* |
|---|---|---|---|---|
| Source IP<br>Source Port | 192.168.1.10<br>4500 | **89.15.287.24**<br>4500 | **192.168.2.100**<br>4500 | **192.168.2.100**<br>4500 |
| Destination IP<br>Destination Port | 10.10.10.1<br>80 | 10.10.10.1<br>80 | **192.168.2.10**<br>80 | **192.168.2.10**<br>80 |

← *Povratni paket*

| **Adresa i port** | **Web poslužitelj -1**<br>*(vraća paket)*<br>*Korak 1* → | **Load Balancer**<br>**(provjerava NAT/SNAT tablicu)**<br>*Korak 2* → | **R1**<br>**(provjerava NAT tablicu)**<br>*Korak 3* → | **PC – 1**<br>*Korak 4* →<br>***(zaprimanje odgovora)*** |
|---|---|---|---|---|
| Source IP<br>Source Port | 192.168.2.10<br>80 | **10.10.10.1**<br>80 | **10.10.10.1**<br>80 | **10.10.10.1**<br>80 |
| Destination IP<br>Destination Port | **192.168.2.100**<br>4500 | **89.15.287.24**<br>4500 | 192.168.1.10<br>4500 | 192.168.1.10<br>4500 |

Prisjetimo se da uređaji koji rade ***NAT***, pohranjuju u svojoj ***NAT*** tablici aktivnih preslikavanja svaku pojedinu sesiju: Izvorišna (*Source*) IP + port → odredišna (*Destination*) IP + port, na osnovu koje uspijevaju raditi translacije adresa. Dakle i kod običnog ***NAT***-a i ***SNAT***-a, IP adrese i pripadajući portovi su identifikatori pojedine komunikacije (*sesije*).

Vidljivo je da je kod iniciranja konekcije računalo **PC-1** na IP: 192.168.1.10, to jest konkretno *sesija* web preglednika na njemu kreirala nasumični port ***4500***, koji se naziva ***efemerni*** port. Taj konkretna port je na tom računalu identifikator sesije unutar web preglednika; konkretno kada se zaprimi povratni paket na ovu IP adresu (192.168.1.10), upravo će port ***4500*** govoriti operativnom sustavu, da sadržaj paketa treba dostaviti web pregledniku. Dok je s druge strane od računala **PC-1** prema odredištu, odredišni (*Destination*) port 80 identifikator da se radi o odredišnom web servisu (port 80 = *HTTP* protokol).



Za način kreiranja i upotrebe ***efemernih*** portova, pogledajte poglavlje:
**19.8.1. TCP/UDP Portovi.**
Za primjere konfiguracije ***NAT*** i/ili ***SNAT*** pravila, pogledajte poglavlje:
**26.7.2.1. Primjeri.**
Pogledajte i poglavlje o **DNAT**u:
**23.4.4.2. Destination NAT (DNAT).**



Osim s `iptables`, ***NAT*** i ***SNAT*** se mogu konfigurirati i s novijom naredbom `nft` to jest sustavom ***nftables.***

**Izvori informacija: (773), (794),(1221),(1222),(1223),(1224),(1298),** `man iptables`**,**`man nft`.

### 23.4.4.2. Destination NAT (DNAT)

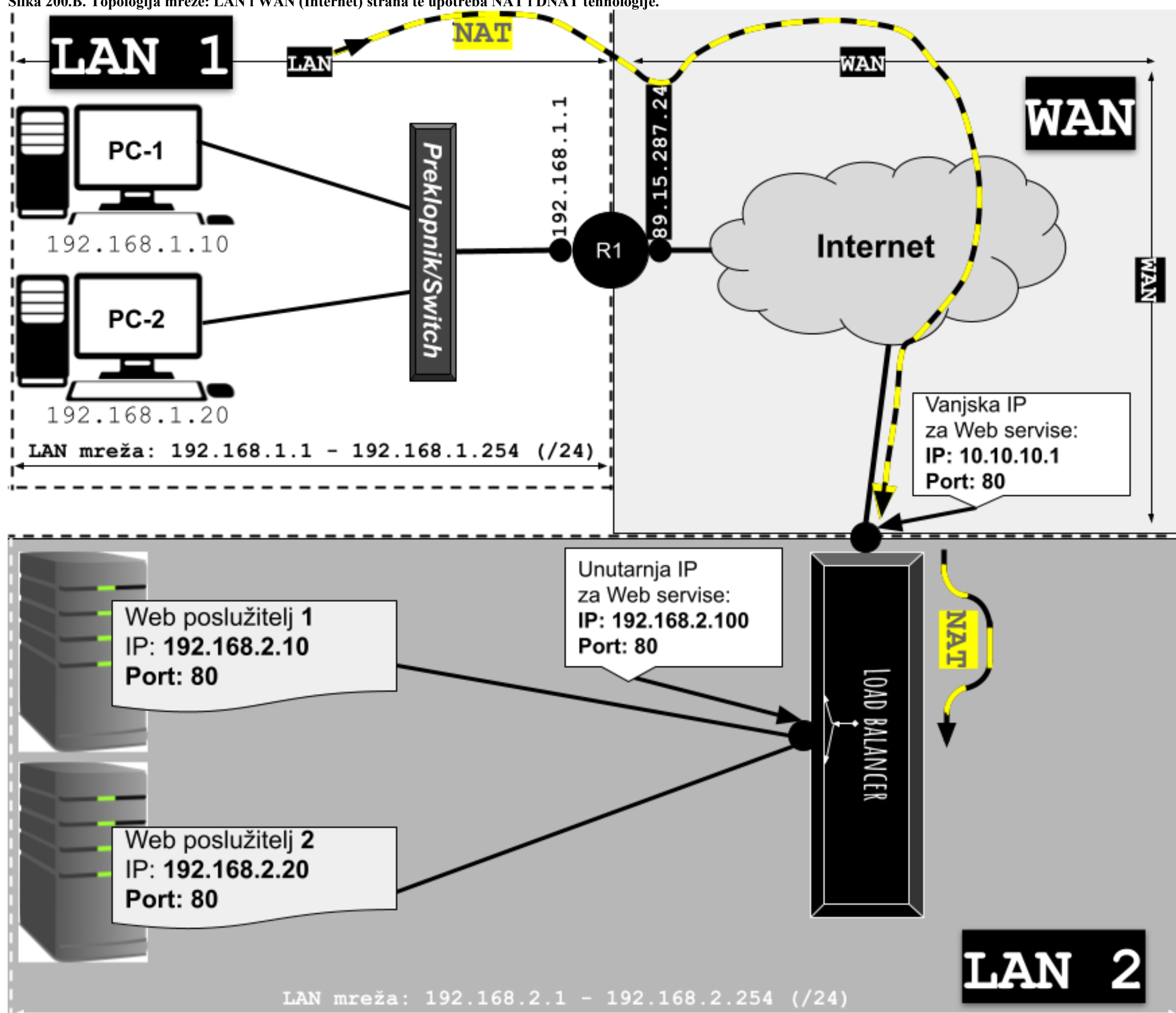
*Destination NAT* (**DNAT**) odnosno prevođenje odredišne mrežne adrese je tehnika za transparentnu promjenu odredišne IP adrese preusmjerenog paketa i izvođenje inverzne funkcije za sve odgovore. Svaki usmjerivač, ***load balancer*** ili ***Proxy*** poslužitelj smješten između dvije krajnje točke u komunikaciji može izvesti ovakvu transformaciju paketa.
Naime odredišni ***NAT*** (***DNAT***) mijenja odredišnu adresu paketa koji prolaze kroz usmjerivač, a nudi i mogućnost izvođenja prevođenja portova u TCP/UDP zaglavljima. Odredišni NAT (***DNAT***) se uglavnom koristi za preusmjeravanje dolaznih paketa s vanjskom adresom ili odredišnim portom na internu IP adresu ili port unutar lokalne (**LAN**) mreže.
Dakle ***DNAT*** prema logici rada radi suprotno od ***SNAT-a***. To znači da on mijenja polje odredišne adrese u IP zaglavlju, tako da se paketi prosljeđuju na drugačiju adresu od izvorne odredišne IP adrese.

Pogledajmo kratku usporedbu ***SNATa*** u odnosu na ***DNAT***:

| Upotreba SNATa | Upotreba DNATa |
|---|---|
| Obično se koristi za promjenu privatne (***LAN***) adrese ili porta u javnu (***WAN***) adresu ili port za pakete koji napuštaju lokalnu mrežu. | Obično se koristi za preusmjeravanje dolaznih paketa s odredištem javne adrese (***WAN***) ili porta na privatnu IP adresu (***LAN***) ili port unutar mreže. |
| Mijenja izvorišne IP adrese. | Mijenja odredišne IP adrese. |
| Mijenja izvorišni port. | Mijenja odredišni port. |
| Izvodi se nakon donošenja odluke o usmjeravanju. | Izvodi se prije donošenja odluke o usmjeravanju. |
| U procesu se odredišna IP adresa zadržava, a izvorna IP adresa se mijenja. | U procesu se izvorna IP adresa se zadržava, a odredišna IP adresa se mijenja. |

**Pogledajmo mrežni paket odnosno što koji uređaj radi s njim u slučaju upotrebe *NAT-a i DNAT-a, koji smo u prethodnoj cjelini samo naveli, a sada ćemo ga objasniti na malo pojednostavljenoj inačici prethodne slike 200.B.***

**Slika 200.B. Topologija mreže: LAN i WAN (Internet) strana te upotreba NAT i DNAT tehnologije.**



**→ *Odlazni paket***

| Adresa<br>i port | PC – 1<br>*Korak 1 →* | R1<br>(radi NAT)<br>*Korak 2 →* | Load Balancer<br>(radi NAT i DNAT)<br>*Korak 3 →* | Web poslužitelj -1<br>*(zaprima paket)*<br>*Korak 4* |
|---|---|---|---|---|
| Source IP | 192.168.1.10 | **89.15.287.24** | **89.15.287.24 (NAT)** | **89.15.287.24** |
| Source Port | 4500 | 4500 | 4500 | 4500 |
| Destination IP | 10.10.10.1 | 10.10.10.1 | **192.168.2.10 (DNAT)** | **192.168.2.10** |
| Destination Port | 80 | 80 | 80 | 80 |

Kada računalo **PC-1** s IP 192.168.1.10 šalje paket na 10.10.10.1, on dolazi do usmjerivača **R1** koji je kao zadani usmjerivač (*Default gateway*) računalu **PC-1** i on izvorišnu adresu mijenja u svoju vanjsku javnu IP adresu (89.15.287.24) dakle radi prvi NAT. Pri tome **R1** ne mijenja izvorišni port niti odredišnu IP adresu. Paket zaprima ***Load Balancer*** i on, pošto mora promet balansirati na stvarni Web poslužitelj, odabire ***Web poslužitelj 1*** (192.168.2.10), i zatim mijenja odredišnu IP adresu u IP adresu web poslužitelja (192.168.2.10) dakle ovdje on rad DNAT odnosno zamjenu odredišne IP adrese i potom šalje paket. U cijelom ovom procesu slanja, nitko ne dira niti odredišni port je on definira web (*HTTP*) servis. U povratku se događa suprotno. Sličnu upotrebu ***DNATa*** odrađuju i ***Proxy*** poslužitelji.



Za primjere konfiguracije ***Load Balancera*** i upotrebu ***Proxy*** poslužitelja, pogledajte poglavlja:
**7.2.2.3. Upotreba posredničkog (*Proxy*) poslužitelja.**
**23.4.5.1. Keepalived i balansiranje opterećenja (*Load Balancing*).**

***Pogledajmo i povratni paket***

| Adresa<br>i port | Web poslužitelj -1<br>*(vraća paket)*<br>*Korak 1 →* | Load Balancer<br>(provjerava NAT tablicu)<br>*Korak 2 →* | R1<br>(provjerava NAT tablicu)<br>*Korak 3 →* | PC – 1<br>*Korak 4 →*<br>***(zaprimanje odgovora)*** |
|---|---|---|---|---|
| Source IP | 192.168.2.10 | **10.10.10.1 (NAT)** | **10.10.10.1** | **10.10.10.1** |
| Source Port | 80 | 80 | 80 | 80 |
| Destination IP | 89.15.287.24 | **89.15.287.24** | 192.168.1.10 (**DNAT**) | 192.168.1.10 |
| Destination Port | 4500 | 4500 | 4500 | 4500 |

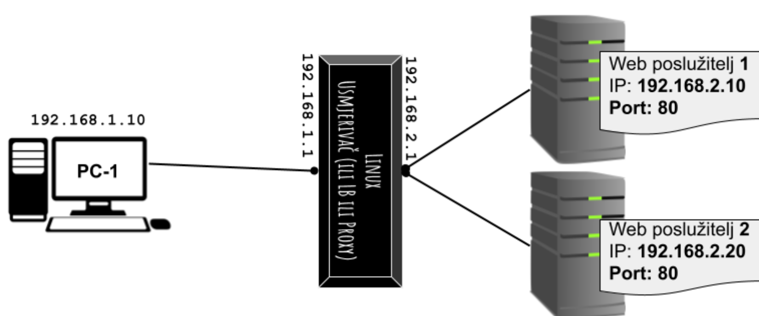
**Pogledajmo primjer ručne konfiguracije *DNATa*.**
Slijedi primjer upotreba ***DNATa*** za sve protokole (portove) na jednu IP adresu:

```
iptables -t nat -A PREROUTING -d 192.168.1.1 -j DNAT --to-destination 192.168.1.129
```

U ovom primjeru, svi paketi koji stignu na usmjerivač s odredištem 192.168.1.1 biti će prvo prepravljeni a potom proslijeđeni, kao da im je odredišna IP adresa 192.168.1.129. Dakle na svim paketima koji prolaze kroz naš Linux usmjerivač, a kojima je odredišna IP adresa 192.168.1.1, ona će biti zamijenjena s IP adresom 192.168.1.129 i takav paket će biti poslan na mrežu.

**Drugi jednostavniji primjer upotrebe DNAT-a bi bio kao na slici 200.C.**

**Slika 200.C. Upotreba DNAT-a**




U ovom primjeru, zbog potreba testiranja, sav promet s bilo kojeg računala, a tako i **PC-1** (192.168.1.1) koji bi bio namijenjen poslužitelju 192.168.2.10, a koji će morati proći kroz podrazumijevani usmjerivač (*Default gateway*) to jest **Linux** na slici, će biti uhvaćen te će mu se promijeniti odredišna IP adresa u 192.168.2.20. Kao takav, paket će biti poslan prema drugom poslužitelju: 192.168.2.20.
Pogledajmo ovakvo ***DNAT*** pravilo:

```
iptables -t nat -A PREROUTING
-d 192.168.2.10 -j DNAT --to-destination 192.168.2.20
```

**DNAT se, kako smo vidjeli, obično i odrađuje korištenjem NAT+PAT tehnologije ili unutar Load Balancer ili Proxy poslužitelja/uređaja !**



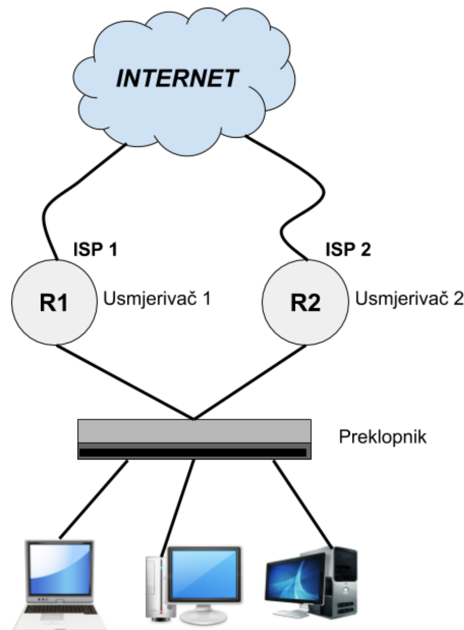
Za primjere konfiguracije ***NAT*** i/ili ***SNAT*** pravila, pogledajte prijašnje poglavlje, ali i poglavlje:
**26.7.2.1. Primjeri.**

**Izvori informacija: (794),(1298),(1376),(1377),** `man iptables`, `man nft`.

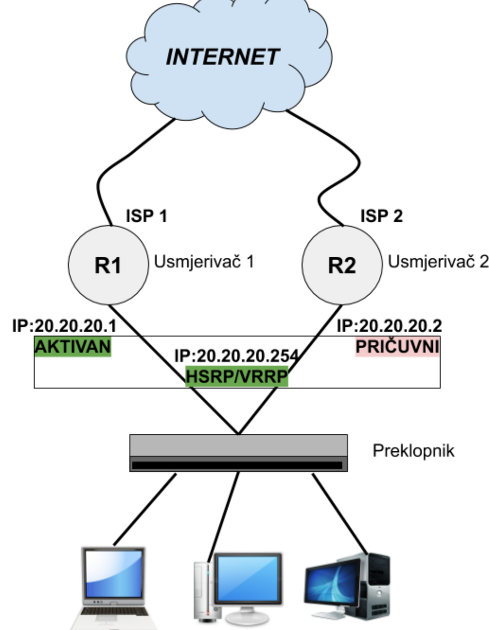
### 23.4.5. Redundancija na OSI sloju tri (OSI 3) i *VRRP* protokol

Prisjetimo se da smo govorili o redundanciji (zalihosti) na OSI sloju dva (OSI 2) u poglavljima: **20.6.5** i **20.6.6.** Međutim sada ćemo govoriti o redundanciji na OSI sloju tri (OSI 3), ali ne upotrebom protokola za usmjeravanje (engl. *Routing protocols*) već protokola koji nam nude mogućnost kreiranja zajedničke virtualne IP adrese između više mrežnih uređaja na ovom sloju mreže. Prvi ovakav protokol naziva **HSRP** (Engl. *Hot Standby Router Protocol*) razvila je tvrtka ***Cisco*** Systems 1998 godine, a definiran je u dokumentu: RFC 2281. Potreba za ovakvim protokolom se pojavila zbog potrebe uvođenja redundancije na razini usmjerivača ili uređaja koji rade na OSI sloju tri (OSI 3). Ovaj protokol radi tako da dva ili više uređaja koji ga podržavaju, kreiraju jednu virtualnu IP adresu koju dijele među sobom, tako da sâmo jedan od uređaja stvarno koristi tu adresu te s nje može slati i primati podatke, a (svi) drugi uređaji su u stanju mirovanja. Sve dok primarni uređaj koji opslužuje ovu IP adresu ne postane nedostupan zbog bilo kojeg od razloga: primjerice nedostupno/neispravno mu je određeno mrežno sučelje ili slično. Tada se pomoću mehanizma izbora (Engl. *Election*) odabire novi primarni (aktivni) uređaj koji će na sebe preuzeti ovu IP adresu. To se najviše koristi za pristupne uređaje (usmjerivače) u slučajevima kada imamo dva ili više uređaja odnosno pružatelja Internetskih usluga (*ISP*), a obično sa strane lokalne (***LAN***) mreže. Primjer je vidljiv na slici **200.1** odnosno na slici **200.2** uz upotrebu HSRP ili VRRP protokola s virtualnom IP adresom koju koriste svi na ***LAN*** mreži, za izlaz na internet.

**Slika 200.1. LAN mreža s dva ISP-a**




**Slika 200.2. LAN mreža s dva ISP i HSRP/VRRP protokolom u primjeni**




Kako je vidljivo na slici **200.2** sva računala u komunikaciji prema internetu koristit će IP adresu: `20.20.20.254` kao adresu svog primarnog usmjerivača (Engl. *Default Gateway*), a koja je zapravo virtualna IP adresa o kojoj se brine *HSRP* odnosno *VRRP* protokol. Naime nakon što je tvrtka *Cisco Systems* razvila *HSRP* protokol, od strane **IEEE** organizacije razvijen je otvoreni protokol naziva **VRRP** (Engl. *Virtual Router Redundancy Protocol*) koji praktično ima identičnu namjenu i upotrebu, a kojega je potom podržao i sam ***Cisco***. Tako da od 2001. godine svi veći proizvođači mrežne opreme kreću s njegovom podrškom, a danas ga svi i podržavaju. ***VRRP*** protokol je prvo definiran standardom RFC 3768, a kasnije s RFC 5798.
Zbog toga što je on (***VRRP***) otvoreni standard, nudi nam mogućnost povezivanja uređaja različitih proizvođača (*Juniper, Cisco, Arista, HP, Brocade,…Linux*). Trenutno su u upotrebi inačice 2 i 3 *VRRP* protokola. Razlika je u tome što inačica v.3. podržava i IPv6 te su brojači (Engl. *Timers*) u novijoj inačici u milisekundama, a ne u sekundama kao u staroj inačici.
Međutim na operativnim sustavima baziranim na ***BSD Unix-u*** se razvio drugi protokol naziva ***CARP*** (Engl. *Common Address Redundancy Protocol*) koji nije kompatibilan s *VRRP* protokolom, iako i on radi na istom principu.



***VRRP*** *protokol je u Linuxu podržan kroz projekt odnosno servis imena* ***keepalived*** *koji sadrži i dodatne funkcionalnosti, poput implementiranog* ***BFD*** *protokola za detekciju ispada komunikacijskih veza te bazičnog load balancinga.*

**Upoznajmo se s osnovama rada VRRP protokola.**
Upotrebom ovog protokola, prvo se kreira *VRRP* grupa kojoj se dodjeljuje *VRRP* virtualna IP adresa. Grupa ima identifikator koji je cjelobrojni broj. *VRRP* grupa može imati vrijednost od 1 do 255. Nadalje dva ili više uređaja (usmjerivača ili uređaja koji rade na OSI sloju tri), a koji podržavaju *VRRP* protokol u inicijalnom trenutku postavljaju, svaki svoj težinski koeficijent odnosno prioritet* (Engl. *Priority*). Ovaj prioritet je cjelobrojni broj od 0 do 255. Tada dolazi do izbora primarnog uređaja** odnosno onoga koji će biti aktivan u određenoj VRRP grupi, tako da će na izborima uvijek pobijediti onaj uređaj koji ima najveći, (ovaj) koeficijent. Standardno se podrazumijeva koeficijent vrijednosti 100 (ako nije postavljen ručno).
Važno je znati da se izbori rade za svaku *VRRP* grupu zasebno, jer svaka zasebna *VRRP* grupa predstavlja jednu instancu *VRRP* protokola. Identifikator grupe je takozvani ***VRID*** (Engl. *Virtual Router IDentifier*).

U našem primjeru imamo konfiguriranu sâmo jednu *VRRP* grupu čija virtualna IP adresa je: `20.20.20.254`. Dodatno *VRRP* uređaji svako malo (definirano u vrijednosti brojača koji se zove ***Advertisement Interval***) šalju poruke s kojima potvrđuju da su živi. Ove poruke se standardno šalju na *multicast* adresu koja je rezervirana za *VRRP* protokol: `224.0.0.18` na onom mrežnom sučelju na kojemu je definirana *VRRP* grupa.



Kao opcija, moguće je umjesto *multicasta* koristiti i *unicast* adrese za razmjenu ovih poruka, između ***VRRP*** uređaja.

***MAC*** adresa koju *VRRP* uređaji u komunikaciji moraju koristiti je iz opsega adresa: `00-00-5E-00-01-`**`XX`**. Pri tome je **`XX`** vrijednost od `00` do `FF`, a ona predstavlja **VRID** broj odnosno zadnji bajt (**`XX`**) zapravo sadrži ***VRID*** broj. Važno je razumjeti kako virtualna ***VRRP IP*** adresa ima naravno (ovu) svoju pripadajuću MAC adresu, a pri tome se ***VRRP IP*** s pripadajućom MAC adresom koristi samo i isključivo na jednom jedinom aktivnom uređaju (Engl. *Master*) jer preko njega i teče sav mrežni promet. U našem slučaju sa slike **200.2** bi to bio aktivni usmjerivač **R1** dok bi pričuvni (Engl. *Backup/Slave*) usmjerivač **R2** bio u stanju čekanja i bez dodijeljene (virtualne) ***VRRP IP*** adrese. ***VRRP IP*** adresa se naziva i ***VIP*** adresa.

Postoji i još jedna dodatna mogućnost, a to je autorizacija prilikom razmjene *VRRP* poruka, kao zaštitni mehanizam kako se ne bi netko ubacio u *VRRP* komunikaciju i preuzeo *VRRP* virtualnu IP adresu.

**Što se događa u trenutku kada je primarni/aktivi (*Master*) usmjerivač ispao iz igre odnosno, ako zbog nekih problema više ne šalje „*Advertisement*" poruke da je živ odnosno aktivan (funkcionalan).**
Tada prvi sljedeći *VRRP* uređaj u *VRRP* grupi s prvim sljedećim najvećim prioritetom* (brojem) preuzima njegovu funkciju i on postaje aktivan odnosno „*Master*" za tu *VRRP* grupu i njenu pripadajuću (virtualnu) IP adresu. Međutim pošto svi drugi mrežni uređaji koji rade na OSI sloju dva (OSI 2), a to se prvenstveno odnosi na preklopnike (*switcheve*) u svojoj ARP/MAC tablici imaju zapisan pâr: ***VRRP IP* + *VRRP MAC***, svima na mreži od tog trenutka VRRP IP adresa više ne bi bila dostupna jer joj se sada promijenila pripadajuća MAC adresa od našeg novo izabranog VRRP aktivnog usmjerivača. Stoga isti tren kada je novo izabrani aktivni usmjerivač (*VRRP Master*) dobio funkciju *Master*-a, on šalje posebnu ***ARP*** poruku na cijelu mrežu, na sve mrežne uređaje i računala na mreži. Ova posebna *ARP* poruka se zove ***Gratuitous ARP*** poruka i ona govori svima da se za staru (*VRRP*) virtualnu IP adresu promijenila njena pripadajuća MAC adresa. Od tog trenutka svi uređaji na mreži (preklopnici, usmjerivači, računala, …) brišu stari *ARP* unos te zapisuju i koriste ovaj novi, tako da će se svi mrežni paketi moći uredno dostaviti na novo odredište. Ne zaboravite da preklopnici (*switchevi*) za prebacivanje mrežnih paketa sa svog jednog mrežnog sučelja na drugo, odnosno za dostavljanje paketa na željeno odredište gledaju samo ***MAC*** adrese, a ne i ***IP*** adresu paketa.



Za prisilno osvježavanje ***ARP*** (*MAC*) tablica upotrebom *Gratuitous ARP* poruka pogledajte poglavlje:
**20.7.1.3. Scenarij za prisilno osvježavanje ARP tablice (*Gratuitous ARP*) detaljnije.**

**Konfiguracija VRRP servisa *keepalived* na Linuxu**
Za potrebe ovog testa potrebna su nam dva računala koja će biti korišteni kao zamjena za usmjerivače **R1** i **R2** na slikama: **200.1** i **200.2**. Na oba računala trebamo instalirati `keepalived` softverski paket na sljedeći način:

```
yum install -y keepalived
```

Pri tome oba računala prema lokalnoj mreži (LAN) imaju `eth0` mrežno sučelje na kojem ćemo i koristiti ***VRRP*** protokol.

***Dodatno moramo omogućiti povezivanje ovog servisa na IP adresu koja još nije lokalna IP adresa, što ćemo napraviti sa:***

```
sysctl -w net.ipv4.ip_nonlocal_bind=1
sysctl -p
```

Sada na oba računala kreirajmo i otvorimo datoteku imena: `/etc/keepalived/keepalived.conf`.
Pogledajmo ove datoteke usporedno; samo jednu sekciju koja je dovoljna: `vrrp_instance`, u konfiguraciji za oba računala:

| Prvo Linux računalo (**R1**) | Drugo Linux računalo (**R2**) | Opis konfiguracije |
|---|---|---|
| `vrrp_instance PRVA {`<br>`state MASTER`<br><br>`interface eth0`<br>`virtual_router_id 10`<br>`priority 150`<br>`advert_int 1`<br>`authentication {`<br>`auth_type PASS`<br>`auth_pass 1234`<br>`}`<br>`virtual_ipaddress {`<br>`20.20.20.254/24`<br>`}`<br>`}` | `vrrp_instance PRVA {`<br>`state BACKUP`<br><br>`interface eth0`<br>`virtual_router_id 10`<br>`priority 100`<br>`advert_int 1`<br>`authentication {`<br>`auth_type PASS`<br>`auth_pass 1234`<br>`}`<br>`virtual_ipaddress {`<br>`20.20.20.254/24`<br>`}`<br>`}` | •Logički naziv VRRP instance<br>•Inicijalno stanje servisa**:<br>→MASTER ili BACKUP<br>•Mrežno sučelje za rad<br>•Broj (identifikator) VRRP **grupe** (**VRID**)*<br>•Prioritet* (veći broj je veći *prio*)<br>•Frekvencija slanjaVRRP poruka<br>•Autentikacija:<br>Vrsta: **PASS** ili **AH** (*IPSEC*)<br>Lozinka: slijedi nekriptirana lozinka<br><br>•VRRP virtualna IP za ovu **grupu*** |

I nakon što smo napravili promjene u konfiguraciji (datoteci od gore), na oba računala pokrenimo ***keepalived*** servis:

```
systemctl start keepalived
```

*Keepalived* servis ima i cijeli nîz naprednih mogućnosti; poput praćenja mrežnih sučelja (`track_interface`) koje, ako se ugasi ili ne radi, uzrokuje ispad VRRP-a na lokalnom računalu odnosno tranziciju u stanje ***BACKUP*** ili ***FAULT***. Isto tako je moguće dodati i sekciju `vrrp_script` pomoću koje se može provjeravati/okidati željena skripta, svakih ***n*** sekundi. Skripta potom može provjeravati status nekog servisa, parametre mreže ili što vam je već potrebno kao okidač prelaska na drugi *VRRP* usmjerivač ili servis.
Opcionalno je moguće definirati i skripte koje se aktiviraju tijekom tranzicije u: ***MASTER***, ***BACKUP*** ili ***FAULT*** stanja sa: `notify_master`, `notify_backup` i `notify_fault`. Zatim je moguće konfigurirati ponašanje *VRRP* grupe: što kada se primarni (*Master*) uređaj oporavi; opcija: `nopreempt`. Moguće je i promijeniti slanje poruka sa standardne *multicast* IP adrese za *VRRP*: `224.0.0.18` na *unicast* adrese uređaja u komunikaciji sa opcijama: `unicast_src_ip` (označava pošiljatelja) i `unicast_peer` (označava udaljeni/drugi VRRP uređaj). Dodatno moguće je da *keepalived* pošalje *e-mail*, kada dođe do neke promijene u radu; što je potrebno definirati u novoj sekciji: `global_defs`.



*Novije inačice programa nude i mogućnost balansiranja mrežnog prometa (Engl.* ***Load Balancing****). Stoga on mora imati i mogućnost praćenja dostupnosti odredišnih poslužiteljskih servisa, što postiže slanjem ili* ***TCP probe*** *ili poruka na aplikacijskoj razini (pr. HTTP) prema udaljenim servisima.*



*U slučaju slanja „****TCP probe****" poruka pogledajte i poglavlje:*
**24.2.12. TCP reset.**

Pošto *VRRP* na aktivnom (*Master*) uređaju dodjeljuje (*VRRP*) virtualnu IP adresu na postojeće mrežno sučelje, ovu virtualnu IP adresu je moguće vidjeti samo sa sljedećom naredbom:

```
ip addr show
```



*Keepalived* se oslanja na kernel modul imena: `ip_vs`.

Provjerimo imamo li ga učitanog sa sljedećom naredbom:

```
lsmod | grep ip_vs
```

***IPVS*** (engl. *IP Virtual Server*) je razvijen tako da se oslanja na ***Netfilter*** komponentu Linuxa te implementira balansiranje mrežnog prometa na transportnom sloju mreže. U slučaju kada *keepalived* koristimo i za balansiranje mrežnog prometa (engl. *Load Balancing*) za tu potrebu se u pozadini koriste kernel moduli za tu funkcionalnost. Nabrojat ćemo ih sâmo nekoliko:

- `ip_vs_rr` - daje nam mehanizam za tzv. *Round Robin* (kružno dodjeljivanje) metodu preraspodjele prometa.
- `ip_vs_lc` - daje nam mehanizam za tzv. *Least connection* metodu preraspodjele prometa.
- `ip_vs_dh` - daje nam mehanizam za tzv. *Destination hashing* metodu preraspodjele prometa.
- `ip_vs_mh` - koristi tzv. *Maglev* algoritam(tek od kernela 4.18+) koji uvijek osigurava odabir istog odredišnog servisa*.
- `ip_vs_sh` - daje nam mehanizam za tzv. *Source hashing* metodu preraspodjele prometa.
- `ip_vs_sed` - daje nam mehanizam za tzv. *Shortest expected delay* metodu preraspodjele prometa.
- `ip_vs_nq` - daje nam mehanizam za tzv. *Never queue* metodu preraspodjele prometa.



Za detaljniju listu algoritama za balansiranje opterećenja, pogledajte poglavlje:
**23.4.5.4. Algoritmi za balansiranje opterećenja (Load Balancing).**

Za listu odnosno ispis svih kernel modula za ovu namjenu pokrenite sljedeću naredbu, odnosno nîz naredbi:

```
ls -al /lib/modules/`uname -r`/kernel/net/netfilter/ipvs/
```



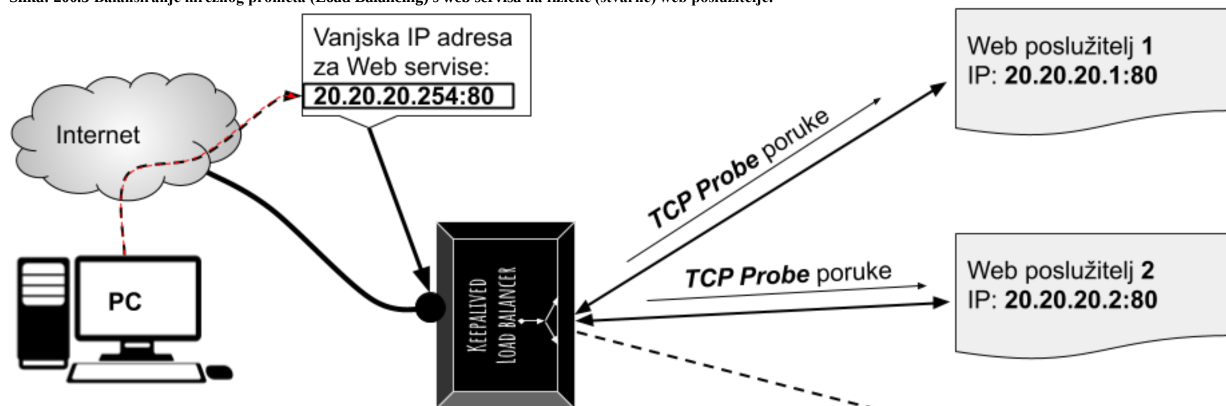
Važna postavka za povezivanje virtualne IP adrese koja još nije lokalna IP adresa, definira se u sysctl varijabli: `net.ipv4.ip_nonlocal_bind` koja u slučaju upotrebe ***VRRP*** protokola treba biti uključena (u stanju 1).

**Izvori informacija:** `man keepalived`,`man keepalived.conf`,`man 7 ip`.

### 23.4.5.1. Keepalived i balansiranje opterećenja (*Load Balancing*)

Servis *Keepalived* nam nudi i osnovnu mogućnost balansiranja opterećenja odnosno raspodjele mrežnog prometa (Engl. *Load Balancing*). Iako za tu namjenu postoje znatno snažniji i konfigurabilniji specijalizirani servisi poput ***HAproxy*** (za HTTP/HTTPS i TCP promet), mi ćemo se ipak nakratko osvrnuti na osnovnu mogućnost servisa *keepalived* za raspodjelu odnosno balansiranje mrežnog prometa. U njegovoj konfiguraciji ćemo se nadovezati na već navedenu konfiguraciju virtualne IP adrese: `20.20.20.254` koju ćemo sada koristiti kao IP adresu, koja će se koristiti kao vanjska IP adresa na koju će se spajati vanjski klijenti (za primjer upotrebe web servisa) odnosno web preglednici.
Iza te vanjske IP adrese će se nalaziti dva web poslužitelja prema kojima će naš *keepalived* servis balansirati mrežni promet, kako je vidljivo na slici **200.3**.

**Slika: 200.3 Balansiranje mrežnog prometa (Load Balancing) s web servisa na fizičke (stvarne) web poslužitelje.**




***Iako ovakav scenarij nije dobar zbog sigurnosnih razloga jer bi trebale biti odvojene unutarnje i vanjske IP adrese na različitim mrežnim sučeljima Load Balancera, mi ćemo ga koristiti zbog lakšeg razumijevanja njegovog načina rada***.

Dakle u navedenom scenariju, klijenti se sa svojim web preglednicima spajaju na vanjsku IP adresu `20.20.20.254`, s koje naš ***keepalived*** servis radi balansiranje prometa prema stvarnim web poslužiteljima na IP adresama: **20.20.20.1** i **20.20.20.2**.
U ovoj komunikaciji se koristi TCP port 80 dakle HTTP. Dodatno naš ***keepalived*** mora svako malo provjeravati jesu li naši web poslužitelji uopće živi, konstantnim slanjem TCP *keepalive* poruka na njih.



Za ***TCP keepalive*** poruke pogledajte poglavlje:
**24.2.15. TCP keepalives**.
Pogledajte i primjer upotrebe ***keepalived*** servisa, u kombinaciji sa servisom ***conntrackd***:
**26.8.4.1. Servis conntrack i sinkronizacija stanja mrežnih veza.**
Pogledajte i primjere rada *Reverznih Proxy* poslužitelja:
**26.4.5. Proxy poslužitelji.**

Pogledajmo **okvirnu** konfiguraciju ovakvog ***keepalived*** servisa uz opise (**#**) u istom retku:

```
virtual_server 20.20.20.254 80 {    # Definira se vanjska IP adresa i port
  delay_loop 5                      # Definicija vremena odgađanja prije prve provjere
  lvs_sched rr                      # Definicija algoritma balansiranja (rr=Round Robin)*
  lvs_method NAT                    # Način rada NAT=translacija adrese:
                                    # -> Vanjska IP: virtual_server
                                    # -> Unutarnje IP: real_server (za sve njih)
  protocol TCP                      # Definira se upotreba TCP protokola
  ha_suspend                        # Ako VirtServ IP nije postavljena, zaustavi aktivnost
                                    # provjere dostupnosti (healtcheck)
real_server 20.20.20.1 80 {       # Definicija prvog web poslužitelja (IP i PORT)
        TCP_CHECK {              # Definira se upotreba TCP keepalive/healthchecker
            connect_timeout 5     # Maks. vrijeme dozvoljeno za odgovor na provjeru
        }
  }
real_server 20.20.20.2 80 {       # Definicija drugog web poslužitelja (IP i PORT)
        TCP_CHECK {              # TCP keepalive/healthchecker(provjera dostupnosti)*
            connect_timeout 5     # Maks. vrijeme dozvoljeno za odgovor na provjeru*
        }
  }
}
```

**Izvor informacija: (913), (K-6), (K-10), RFC 2281, RFC 3768 i RFC 5798 te pogledajte poster: P4.**

### 23.4.5.2. Oblik VRRP poruke (paketa)

***VRRP*** poruka odnosno mrežni paket se sastoji od sljedećih dijelova, vidljivih u tablici:

| **IP (IPv4 ili IPv6) zaglavlje** | | | | |
|---|---|---|---|---|
| Version | Type | Virtual Router ID | Priority | Count IP address |
| Authentication type | Advertisement Interval | Checksum | | |
| IP address(es) | | | | |
| Authentication Data | | | | |

Slijedi opis polja:

- **IP zaglavlje** nećemo detaljnije objašnjavati, a ono sadrži: *izvorišnu* i *odredišnu* IP adresu (standardno *multicast*: *224.0.0.18* ili unicast) te protokol broj ***112*** u slučaju VRRP-a, *TTL* i druga polja.
- `Version` – polje definira inačicu VRRP protokola (v.1, v.2 ili v.3).
- `Type` – polje definira vrstu VRRP paketa: definiran je samo broj `1`: kao `Advertisement` poruka.
- `Virtual Router ID` – polje definira ***VRID*** broj odnosno pripadnost VRRP grupi (1-255).
- `Priority` – polje definira prioritet (težinski koeficijent) uređaja koji šalje ovu poruku.
- `Count IP address` – polje definira VRRP (virtualnu/e) IP adresu.
- `Authentication type` – polje definira metodu autentikacije za **v.1/2.** protokola; za sada su podržane: **0** (bez autentikacije) te **1** ili **2**.
- `Advertisement Interval` – polje definira svakih koliko sekundi će se slati „`Advertisement`“ poruke o dostupnosti VRRP uređaja.
- `Checksum` – u ovo polje se upisuje provjerni zbroj (*Checksum*) svih VRRP polja zbog detekcije grešaka unutar cijelog VRRP zaglavlja.
- `IP addressess` – polje sadrži jednu (ili moguće i više) virtualnih VRRP IP adresa unutar VRRP grupe (***VRID***).
- `Authentication data` – polje sadrži lozinku ako se koristi (vezano za polje: `Authentication type`).

Sada pogledajmo jedan ***VRRP*** mrežni paket poslan s IP adrese: `20.20.20.1` i to s aktivnog *VRRP* uređaja, kako je vidljivo prema OSI slojevima: **1**, **2**, **3** i **4** kako slijedi u ispisu:

```
(1)Frame 21: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
(2)Ethernet II, Src: 00:00:5e:00:01:0a, Dst: 01:00:5e:00:00:12
(3)Internet Protocol Version 4, Src: 20.20.20.1, Dst: 224.0.0.18
(4)Virtual Router Redundancy Protocol
    Version 2, Packet type 1 (Advertisement)
        0010 .... = VRRP protocol version: 2
        .... 0001 = VRRP packet type: Advertisement (1)
    Virtual Rtr ID: 10
    Priority: 150
    Addr Count: 1
    Auth Type: Simple Text Authentication [RFC 2338] / Reserved [RFC 3768] (1)
    Adver Int: 1
    Checksum: 0x5375 [correct]
    [Checksum Status: Good]
    IP Address: 20.20.20.254
    Authentication String: 1234
```

Ovdje vidimo da se na OSI sloju dva `(2)` s MAC adrese `00:00:5e:00:01:0a` šalje *VRRP* poruka (ovaj paket) na *multicast* MAC adresu `01:00:5e:00:00:12` što znači da ju svi preklopnici moraju proslijediti na sva svoja mrežna sučelja (portove). Naime ovdje koristimo standardnu *multicast* komunikaciju, iako je moguće forsirati i *unicast* prema potrebi.
Na OSI sloju tri `(3)` je zatim vidljivo da se s naše IP adrese `20.20.20.1` šalje paket na *multicast* IP adresu rezerviranu za VRRP protokol `224.0.0.18`.
I sada na OSI sloju četiri `(4)` vidimo *VRRP* dio poruke. Dakle vidimo da se radi o VRRP inačici `2` te poruci tipa: `Advertisement`.
Nadalje vidimo da je ***VRID*** (`Virtual Rtr ID`) broj odnosno grupa broj `10`, a što je vidljivo i u (`Src`) MAC adresi: `00:00:5e:00:01:0a` (`0a` = heksadecimalno `10`).
Vidimo da je prioritet našeg Linux računala (koje šalje ovu poruku) `150` te kako je virtualna VRRP IP adresa ove grupe broj `10` IP adresa: `20.20.20.254`.



Vezano za *multicast* adrese i njihov način upotrebe i rada, pogledajte poglavlje:
**22.3. Multicast.**
Pošto se s **VRRP** protokolom postiže visoka dostupnost sustava, za detalje o tome što ona znači, proučite poglavlje:
**26.8.4. Visoko dostupni sustavi (*High Availability*).**

**Izvori informacija: (795),(796),(797), RFC 3768, RFC 5798.**

### 23.4.5.3. Balansiranje opterećenja (*Load Balancing*) – ručni rad

Kako smo vidjeli na primjeru upotrebe keepalived servisa, na Linuxu, na najnižoj razini postoji implementacija raznih algoritama za balansiranje opterećenja (Engl. *Load Balancing*) koje u pozadini ovaj servis i koristi. Međutim za upotrebu ovih algoritama odnosno osnovno balansiranje opterećenja pod Linuxom, nije nam potreban keepalived servis. Jedino što nam je potrebno je podrška za ove algoritme koja se bazira na ***IPVS*** (Engl. *IP Virtual Server*) komponenti koja je razvijena tako da se oslanja na ***Netfilter*** komponentu Linuxa te implementira balansiranje mrežnog prometa na transportnom sloju mreže Linuxa. Dakle prvo moramo učitati ***IPVS*** kernel modul imena: `ip_vs`. Provjerimo imamo li ga učitanog sa sljedećom naredbom:

```
lsmod | grep ip_vs
```

Ako on slučajno nije učitan, možemo ga i ručno učitati, na način kako se i učitavaju kernel moduli:

```
modprobe ip_vs
```

Algoritme za balansiranje opterećenja smo već spomenuli, a detaljnije ih možete proučiti u sljedećem poglavlju.
Potom moramo instalirati naredbu s kojom možemo konfigurirati mehanizme balansiranja opterećenja.

Stoga ju instalirajmo na sljedeći način:

```
yum -y ipvsadm
```

Sada smo dobili novu naredbu imena: ipvsadm. Osim toga instaliran je i servis ***ipvsadm (*)*** koji se može koristiti, imamo li potrebu da se pravila odnosno određena konfiguracija balansiranja opterećenja učitava tijekom svakog pokretanja računala. Inicijalna konfiguracija servisa ***ipvsadm*** se nalazi pohranjena u datoteci: `/etc/sysconfig/ipvsadm-config`.

Mi ćemo napraviti istu konfiguraciju koju smo napravili sa ***keepalived*** servisom, oslanjajući se na shemu na slici **200.3**, koja se nalazi dvije stranice prije. Dakle kreirat ćemo jednu (*sekundarnu*) ***IP*** adresu na našem mrežnom sučelju `eth0` koju ćemo potom koristiti kao ***IP*** adresu našeg novog *Load Balancera*.
Dodijelimo sekundarnu ***IP*** adresu: ***20.20.20.254*** našem mrežnom sučelju `eth0`:

```
ip addr add 20.20.20.254/24 dev eth0
```

Naš budući *Load Balancer* će sve zahtjeve koji dođu na njega tj. na IP: ***20.20.20.254***, i to na njegov ***TCP*** port ***80*** (*HTTP*), balansirati odnosno raspodjeljivati na stvarne Web poslužitelje, na IP adresama: ***20.20.20.1*** (prvi web poslužitelj) i ***20.20.20.2*** (drugi web poslužitelj). Algoritam koji ćemo koristiti za balansiranje prometa je *Round Robin* (`rr`). Postavimo ova pravila:

```
ipvsadm --add-service --tcp-service 20.20.20.254:80 --scheduler rr

ipvsadm --add-server --tcp-service 20.20.20.254:80 --real-server 20.20.20.1:80 -m
ipvsadm --add-server --tcp-service 20.20.20.254:80 --real-server 20.20.20.2:80 -m
```

- U prvom koraku smo kreirali *Load Balancer* kao ***TCP*** servis (`--tcp-service`) koji sluša na ***IP*** adresi: ***20.20.20.254***, na ***TCP*** portu ***80*** te će koristiti *Round Robin* metodu (`--scheduler rr`) balansiranja opterećenja.
- U drugom redu (naredbi) za ***TCP*** servis *Load Balancera* kreiranog u prvom koraku, dakle na navedenoj ***IP*** adresi: `20.20.20.254` i pripadajućem ***TCP*** portu ***80***, dodajemo prvi stvarni web poslužitelj (`--real-server`) na ***IP*** adresi `20.20.20.1` i ***TCP*** portu ***80***. Potom navodimo da će se raditi translacija **IP** adresa (***NAT***), s prekidačem (`-m`).
  ***NAT*** način rada navodimo jer želimo da je stvarnom Web poslužitelju naš *Load Balancer* izlazni uređaj, kako bi on sav promet slao na njega te kako je Web klijentima on odredišni uređaj na koji se spajaju. To znači da će sav dolazni Web promet (TCP port 80 - HTTP) dolaziti na IP adresu ***LB***-a: `20.20.20.254` na ***TCP*** port ***80***, a on će pomoću ***NAT*** metode prosljeđivati promet balansirajući slanje između dva stvarna Web poslužitelja: `20.20.20.1` te `20.20.20.2`.
  U povratku će Web poslužitelji vraćati promet na naš *Load Balancer* na IP adresu: `20.20.20.254` s koje će ga *Load Balancer* proslijediti klijentima koji su i inicirali promet (pr. Web preglednicima). Dakle u ***NAT*** načinu rada *Load Balancer* je posrednik između Web klijenata (Web preglednika koji se spajaju na Web poslužitelje) i Web poslužitelja.
- U trećem redu navodimo sljedeći stvarni Web poslužitelj na ***IP*** adresi: `20.20.20.2`, slično kao i u drugom redu.

Sada pogledajmo što smo sve napravili, s naredbom ipvsadm na sljedeći način:

```
ipvsadm -Ln

IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port     Scheduler Flags
  -> RemoteAddress:Port                  Forward Weight ActiveConn InActConn
TCP  20.20.20.254:80          rr
  -> 20.20.20.1:80                         Masq    1      0          0
  -> 20.20.20.2:80                         Masq    1      0          0
```

Vidimo *Load Balancer* kako koristi ***TCP*** protokol (`TCP`) te da radi na sljedećoj ***IP*** adresu i portu: (`20.20.20.254:80`) te kako se koristi *Round Robin* (`rr`) metoda balansiranja opterećenja, koja za tu namjenu koristi kernel modul: `ip_vs_rr`.
U sljedeća dva reda vidimo dva stvarna Web poslužitelja s pripadajućim IP adresama i portovima (`20.20.20.1:80` i `20.20.20.2:80`) te kako se od njih i prema njima njih koristi ***NAT*** metoda koja se ovdje vidi kao: `Masq` (*masquerading*).

Ova pravila možemo ručno snimiti u datoteku koju će servis ***ipvsadm**** **(ako smo ga aktivirali)** čitati tijekom restarta računala; sa:

```
ipvsadm-save -n > /etc/sysconfig/ipvsadm
```

Ne zaboravimo uključiti i prosljeđivanje paketa između mrežnih sučelja, jer bez toga nam sve napravljeno neće raditi:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

**Izvori informacija: (942),(943),(1427),(1428), `man ipvsadm`,`man ipvsadm-save`,`man 5 procfs`.**

### 23.4.5.4. Algoritmi za balansiranje opterećenja (Load Balancing)

U nastavku ćemo se upoznati s nekim od dostupnih algoritama za balansiranje opterećenja odnosno takozvani *Load balancing* na Linuxu. Važno je razumjeti da mnogi čimbenici utječu na pravilan izbor algoritma za balansiranje nekad nazivano i uravnoteženje opterećenja. Konačni izbor algoritama najviše ovisi o aplikacijama kojima se upravlja. Jedna od preporuka je započeti s jednostavnim statičkim algoritmima te prijeći na dinamične i složenije algoritme prema potrebi.

Kod upotrebe programa ili sustava za balansiranje opterećenja pretpostavili smo da je svaka mrežna veza neovisna o svakoj drugoj vezi, tako da se svaka veza može dodijeliti drugom poslužitelju neovisno o svim prošlim, sadašnjim ili budućim dodjelama. Međutim, ponekad se veze s istog klijenta moraju dodijeliti uvijek istom poslužitelju zbog funkcionalnosti ili zbog izvedbe. **FTP** protokol je primjer zbog funkcionalnosti; jer klijent otvara dva komunikacijska kanala odnosno porta; jedan kontrolni (port 21) i jedan za podatke (port 20) i to ako **FTP** radi u aktivnom načinu rada. Međutim kod pasivnog načina rada **FTPa**, nakon spajanja klijenta na kontrolni port (port 21), poslužitelj mu šalje podatak koji (nasumični) port mu je dostupan za podatke, na koji se potom klijent i spaja. Stoga i potreba za spajanje klijenta uvijek na isti FTP poslužitelj.
Drugi protokol za koji je također važno da se klijent uvijek spaja na isti poslužitelj je **SSL** (engl. *Secure Socket Layer*), koji se primjerice koristi za **HTTPS**, sigurnu elektroničku poštu (*secure mail*) i druge aplikacijske protokole.
Naime prilikom uspostavljanja *SSL* (*TLS*) veze klijenta prema poslužitelju dolazi do razmjene sigurnosnih ključeva za kriptiranje (šifriranje) podataka, pa je stoga važno da klijent uvijek komunicira s istim poslužiteljem.
Način rada u kojem klijent uvijek mora komunicirati s istim poslužiteljem, naziva se ***Session persistence*** ili ***Sticky session***.

Sortirana lista svih trenutno dostupnih algoritama (*ovisi o inačici kernela*) može se dobiti sa sljedećim nizom naredbi:

```
ls -al /lib/modules/`uname -r`/kernel/net/netfilter/ipvs/ |awk '{print $9}' | grep
ip_vs | sort
```

Učitavanje potrebnih kernel modula smo opisali u prethodnim cjelinama, pa se na to sada nećemo fokusirati. Važno je znati da u Linuxu podršku za balansiranje (raspodjeljivanje) opterećenja i pripadajuće algoritme dolazi u takozvanoj ***IPVS*** (Engl. *IP Virtual Server*) komponenti sustava.

Važno je znati da postoji cijeli niz ***sysctl*** varijabli i ***/proc*** datoteka za ovu namjenu.
Listu ovih ***sysctl*** varijabli možemo vidjeti sa sljedećom naredbom, dok za detalje o varijablama pogledajte izvor (**1429**):

```
sysctl -a | grep net.ipv4.vs.
```

Odnosno listu pripadajućih **/proc** datoteka možemo vidjeti sa:

```
ls -s /proc/sys/net/ipv4/vs/
```

Vezano za algoritme, razlikujemo dvije kategorije algoritama za balansiranje (raspodjeljivanje) mrežnog prometa.
**Statički algoritmi:**

- ***Round robin*** (`ip_vs_rr`) - prema ovom algoritmu sustav prosljeđuje svaki novi zahtjev za povezivanjem sljedećem poslužitelju u nizu (iz popisa dostupnih poslužitelja). Ravnomjerno raspoređujući veze preko niza poslužitelja na koje se balansira mrežni promet odnosno opterećenje. Dakle *Round-robin* algoritam šalje svaki dolazni zahtjev sljedećem poslužitelju na popisu poslužitelja (tzv. ***pool*** ili ***real servers***). Ova metoda dobro funkcionira u većini primjena, ravnomjerno distribuirajući mrežni promet, ako su poslužitelji prema kojima se balansira promet podjednake hardverske i programske (softverske) konfiguracije. Međutim u slučajevima veza čiji životni vijek nije podjednak to jest ako se neke veze brže zatvaraju od drugih, ova metoda nije najbolja. Pošto ovaj algoritam ne prati broj aktivnih veza prema svakom pojedinom poslužitelju, može doći do toga da određene veze koje duže traju ostaju otvorene na određenim poslužiteljima, na koje će se i dalje raspodjeljivati promet, jednako kao i na one poslužitelje kojima su se veze već zatvorile odnosno koji imaju efektivno manji broj aktivnih veza. To konkretno dovodi do nejednolikog opterećenja poslužitelja.

- ***Weighted round robin*** (`ip_vs_wrr`) - on je sličan klasičnom *round robin* algoritmu, ali dodaje mogućnost raspodjeljivanja dolaznih zahtjeva klijenata iz skupa poslužitelja (***pool***) prema relativnom kapacitetu svakog poslužitelja. Najprikladniji je za raspodjeljivanje dolaznih zahtjeva klijenata preko poslužitelja s različitim mogućnostima ili dostupnim resursima. Pri tome se dodjeljuje težinski koeficijent svakom (aplikacijskom) poslužitelju na temelju kriterija po vlastitom izboru, koji označava relativnu sposobnost obrade prometa svakog dostupnog poslužitelja. Poslužitelji s većim težinskim koeficijentima prvi primaju nove veze od onih s manjim težinskim koeficijentima. Pri tome poslužitelji s višim težinskim koeficijentima dobivaju više novih veza od onih s manjim težinskim koeficijentima, a poslužitelji s jednakim težinskim koeficijentima dobivaju jednaki broj veza.

- ***Source IP Hash*** (`ip_vs_sh`) - algoritam za raspodjelu (uravnoteženje) opterećenja koristi izvorišnu IP adresu klijenta za generiranje jedinstvenog (*hash*) ključa za povezivanje klijenta s određenim poslužiteljem. Budući da se ključ može ponovno generirati ako se sesija prekine, to omogućuje preusmjeravanje zahtjeva za ponovno povezivanje na isti poslužitelj koji je prethodno korišten. To se zove afinitet poslužitelja. Ova metoda uravnoteženja opterećenja najprikladnija je kada se klijent uvijek mora proslijediti na isti poslužitelj pri svakoj uzastopnoj vezi. Primjer ovakvog rada je u scenarijima kada korisnik web trgovine u svoju košaricu za kupnju stavlja određene artikle, na jednom poslužitelju, a oni bi trebali biti tamo i kada se korisnik kasnije poveže.

- ***SIP persistence engine*** (`ip_vs_pe_sip`) - ovaj algoritam održava postojanost veze prema poslužiteljima na temelju **SIP** (ID) poziva.

- **Destination IP Hash** (`ip_vs_dh`) - algoritam za raspodjelu (uravnoteženje) opterećenja koristi odredišnu IP adresu klijenta za generiranje jedinstvenog (*hash*) ključa za povezivanje klijenta s određenim poslužiteljem. Ovaj algoritam je dizajniran za korištenje u klasteru poslužitelja kao tzv. *Proxy-cache*. Postoje i njegove dvije pod izvedenice:
    - ***Locality-based least-connection*** (`ip_vs_lblc`) - ovo je algoritam za raspodjelu (uravnoteženje) opterećenja prema odredišnoj IP adresi. Obično se koristi u klasterima *Proxy-cache* sustava. Ovaj algoritam obično usmjerava paket namijenjen određenoj odredišnoj IP adresi na svoj poslužitelj, ako je poslužitelj živ i pod opterećenjem. Ako je poslužitelj preopterećen (broj njegovih aktivnih veza veći je od njegovog težinskog koeficijenta) i postoji poslužitelj koji je polovično opterećen, tada vezu dodjeljuje poslužitelju s najmanje veza na ovoj IP adresi.
    - ***Locality-based least-connection with replication scheduling*** (`ip_vs_lblcr`) - također je algoritam za raspodjelu (uravnoteženje) opterećenja prema odredišnoj IP adresi Obično se koristi u *Proxy-cache* sustavima. Razlikuje se od *LBLC* prema slijedećem: *balanser* opterećenja održava mapiranja od cilja do skupa poslužiteljskih čvorova koji mogu opsluživati ciljanu IP adresu. Zahtjevi za cilj dodjeljuju se čvoru s najmanjim brojem veza iz skupa poslužitelja. Ako su svi čvorovi u skupu poslužitelja preopterećeni, odabire se čvor s najmanje veza i dodaje ga se u privremeni skup poslužitelja. Ako ovaj skup poslužitelja nije modificiran određeno vrijeme, najopterećeniji čvor se uklanja iz skupa poslužitelja.

**Dinamički algoritmi:**

- **Least connection** (`ip_vs_lc`) - ovaj algoritam usmjerava nove mrežne veze na onaj poslužitelj s najmanjim brojem uspostavljenih veza. Ovo je jedan od algoritama dinamičkog raspoređivanja veza odnosno prometa, koji dinamički prebrojava broj aktivnih veza za svaki poslužitelj kako bi procijenio njegovo opterećenje. Njegov unutarnji mehanizam uravnoteženja opterećenja bilježi broj aktivnih veza prema svakom poslužitelju, povećavajući broj veza prema poslužiteljima kod otvaranja novih veza, te smanjujući broj veza poslužitelja kada se veza zatvori ili joj istekne vrijeme trajanja. Ovaj algoritam je stoga učinkovit u raspodjeli prometa uz brigu o opterećenju poslužitelja što se tiče broja aktivnih veza na svaki pojedini poslužitelj (iz popisa dostupnih poslužitelja). On je dobar odabir u slučaju kada veze imaju različita vremena trajanja, prema poslužiteljima podjednake hardverske i programske (softverske) konfiguracije.
- **Weighted Least connection** (`ip_vs_wlc`) - proširuje *Least connection* algoritam kako bi se uzele u obzir različite karakteristike aplikacijskih poslužitelja. I ovdje se dodjeljuje težinski koeficijent svakom (aplikacijskom) poslužitelju, ali na temelju relativne procesorske snage i dostupnih hardverskh resursa. Odluke o raspodjeljivanju (uravnoteženju) opterećenja temelje se na aktivnim vezama i dodijeljenim težinskim koeficijentima poslužitelja. Primjerice ako postoje dva poslužitelja s najmanjim brojem veza, preferira se poslužitelj s najvećim težinskim koeficijentom.
- **Maglev hashing** (`ip_vs_mh`) - dodjeljuje mrežne veze poslužiteljima traženjem statički dodijeljene posebne *hash* tablice koja se naziva tablica pretraživanja. *Maglev hashiranje* radi dodjeljivanjem popisa preferencija svih pozicija tablice pretraživanja prema svakom odredištu. Kroz ovu operaciju, mehanizam raspršivanja daje gotovo jednako iskorištenje odredišta (poslužitelja) i pruža minimalne smetnje. Kada se skup odredišta odnosno poslužitelja promijeni, nova veza će vjerojatno biti poslana na isto odredište (poslužitelj) kao i prije.
- **Shortest expected delay scheduling** (`ip_vs_sed`) - ovaj algoritam dodjeljuje mrežne veze poslužitelju s najkraćim očekivanim kašnjenjem (engl. *delay*). Baziran je na izračunu koji koristi mjerenje broja aktivnih veza i težinski koeficijent svakog poslužitelja.
- **Weighted overflow** (`ip_vs_ovf`) - ovaj algoritam usmjerava mrežne veze na poslužitelj s najvećim težinskim koeficijentom koji je trenutno dostupan, a prelazi na sljedeći kada aktivne veze premaše poseban težinski koeficijent (engl. *fixed service rate*) poslužitelja.
- **Never queue** (`ip_vs_nq`) - ovaj algoritam koristi red čekanja koji koristi model s dvije brzine. Kada je dostupan neaktivan poslužitelj, nove veze će biti poslane na neaktivan poslužitelj, umjesto da se čeka na onaj brži. Kada nema dostupnog poslužitelja u stanju mirovanja, nove veze će biti poslane poslužitelju koji minimizira očekivano kašnjenje upotrebom algoritma za planiranje najkraćeg očekivanog kašnjenja (engl. *Shortest Expected Delay scheduling*).
- **Weighted failover** (`ip_vs_fo`) - ovaj algoritam radi tako da usmjerava nove veze na poslužitelj s najvećim težinskim koeficijentom, koji je trenutno dostupan.
- **Weighted random twos choice least-connection** (`ip_vs_twos`) - ovaj algoritam svaki puta odabire dva nasumična (odredišna) poslužitelja te u konačnici odabire onaj s najmanjim brojem aktivnih veza uz normalizaciju na osnovu težinskog koeficijenta.



Za detalje o **Proxy** poslužiteljima (koji mogu obavljati i zadaću ***Load Balancing-a***), pogledajte:
**26.4.5. Proxy poslužitelji.** ← pogledajte *Reverzne Proxy* poslužitelje.

**Izvori informacija: (1426),(1427),(1428),(1429),(1430),(1431),(1432),(1433),** **`man ipvsadm`**, **`man ipvsadm-save`**, **`man 5 procfs`**.

# 24. Transportni protokoli (OSI sloj 4)

Prije nego se upoznamo s transportnom protokolima, moramo razumjeti što su komunikacijski protokoli.
Komunikacijski protokol čini skup pravila koja omogućuje dvama ili više entiteta komunikacijskog sustava da prenose informacije odnosno da međusobno komuniciraju. Protokol definira pravila, sintaksu, semantiku i sinkronizaciju komunikacije te moguće metode ispravljanja pogrešaka. Protokoli se mogu implementirati hardverom, softverom ili njihovom kombinacijom. Komunikacijski sustavi koriste dobro definirane formate za razmjenu različitih poruka. Svaka poruka ima točno značenje koje ima za cilj dobiti odgovor iz niza mogućih odgovora unaprijed određenih za tu određenu situaciju. Sudionici u komunikaciji moraju usuglasiti komunikacijske protokole. Da bi se postigao sporazum, protokol se može razviti u tehnički standard, što je vrlo čest slučaj u praksi.

Vidjeli smo kako se **TCP/IP** skup protokola oslanja na OSI model kao teoretski model u kojem se aplikacijski podaci spuštaju prema nižim slojevima, od kojih svaki sloj dodaje svoje zaglavlje i ima svoju funkciju. U konačnici se nakon što je i najniži sloj dodao svoje zaglavlje, formira mrežni paket koji se šalje na mrežu. U ovoj knjizi krenuli smo s primanjem paketa te najnižim OSI slojem OSI 1 te smo vidjeli kako on dodaje svoje zaglavlje te sve prosljeđuje OSI 2 sloju koji u svom zaglavlju osim drugih polja zapisuje izvorišnu i odredišnu **MAC** adresu (takozvane *Ethernet* adrese). Potom sve dolazi do OSI 3 sloja odnosno IP sloja prema TCP/IP modelu. **IP** sloj to jest sâm *Internet Protokol* je zadužen za komunikaciju s krajnjim točkama u komunikaciji (pr. računalima, mrežnim uređajima,…) pri slanju podataka. S druge strane odnosno kod primanja podataka od aplikacije odnosno s viših slojeva, on je zadužen za ugniježđenje (*enkapsulaciju*) podataka s viših slojeva poput *TCP* ili *UDP* sloja te svega onoga što oni nose. Unutar IP zaglavlja nalaze se izvorišna i odredišna IP adresu, ali i druga polja važna za dostavljanje ***datagrama.***
Ideja dizajna ***IP*** protokola radi prema "*end-to-end*" principu, odnosno osiguravanju komunikacije između dvije krajnje točke u komunikaciji. Imajući na umu da se cijela mrežna infrastruktura može smatrati nepouzdanom, počevši od svakog mrežnog elementa, preko njihovih međuveza odnosno sâmih medija preko kojih su povezani. Dodatni faktor koji se uzeo u obzir pri dizajnu je i činjenica da je mreža dinamična i stalno se mijenja. Kao posljedica ovakvog dizajna vezanih za navedene pretpostavke, Internet protokol (**IP**) jedino nudi dostavu paketa, prema principu "*best-effort*" odnosno najbolje što se može, a što čini komunikaciju potencijalno nepouzdanom. To znači kako se sâm IP protokol smatra protokolom koji se ne brine o sâmoj konekciji i urednom dostavljanju paketa na odredište.
Stoga dolazimo do transportnog odnosno OSI 4 sloja odnosno TCP ili UDP sloja prema TCP/IP modelu. Transportni protokoli zaduženi su za prijenos podataka (paketa) preko mreže, prema definiciji TCP/IP protokola. Oni u svom zaglavlju osim sâmih podataka (ako gledamo u smjeru slanja prema mreži) nose informaciju i o takozvanim izvorišnim i odredišnim portovima, koji identificiraju aplikaciju (sa strane klijenta) te vršni aplikacijski protokol prema poslužitelju (pr. ***HTTP***, ***HTTPS***, ***SSH***, …).

Prema TCP/IP modelu, transportne protokole dijelimo u dvije kategorije:

- Pouzdane odnosno one koji se brinu o tome je li svaki poslani paket i primljen na odredištu, za što se koristi **TCP** protokol **(**engl**.** *Transmission Control Protocol***)**.
- Manje pouzdane koji se brinu samo o tome da je svaki paket uredno poslan, bez informacije o tome je li na odredištu paket i uredno zaprimljen, za što je zadužen **UDP** protokol (engl. *User Datagram Protocol*).

Sumirajmo: upotrebom TCP/IP skupa protokola u mrežnoj komunikaciji, podaci koji se šalju ili primaju s mreže razlamaju se u manje cjeline koje nazivamo mrežni paketi. Pri tome svaki mrežni paket osim samih podataka koje prenosi, mora sadržavati identifikatore svake pojedine veze (navodimo ih od nižih do viših OSI slojeva):

- **OSI 2** (*Ethernet*) - izvorišna i odredišna **MAC** adresa odnosno *Ethernet* [**MAC**] adresa pošiljatelja i primatelja.
- **OSI 3** (*IP*) - izvorišna i odredišna **IP** adresa: **IP** adresa pošiljatelja i **IP** adresa primatelja paketa.
    - **OSI 4** (*TCP/UDP*) - na razini transportnih protokola su to TCP/UDP portovi koji su identifikatori klijentske aplikacije i poslužiteljske strane (aplikacijskog poslužitelja) i to izvorišni i odredišni **TCP** ili **UDP** port.

Pogledajmo i okvirni izgled mrežnog paketa s naglaskom na dijelove koje formira svaki OSI sloj tj. TCP/IP sloj:

| ***OSI 1*** | ***OSI 2*** | ***OSI 3 (IP)*** | ***OSI 4 (TCP/UDP)*** | | ***OSI 5 - 7*** | ***OSI 2*** | ***OSI 1*** |
|---|---|---|---|---|---|---|---|
| *Preambula*<br>*+*<br>*SFD* | *Odredišna* ***MAC***<br>*Izvorišna* ***MAC***<br><br>***EtherType***<br>(*za OSI 3*)<br>. . . | *Izvorišna IP*<br>*Odredišna IP*<br><br>***Protokol***<br>(*za OSI 4*),<br>*TTL,* | *Izvorišni* ***port***<br>*Odredišni* ***port*** | | **Aplikacijski podaci [DATA]** | *FCS* | *Interframe gap* |
| | | | **TCP** | **UDP** | | | |
| | | | *SEQ Nr.*<br>*ACK Nr.* | *Length*<br>*Checksum* | | | |

Nadalje, za svaku pojedinu vezu, navedene identifikatore veze (IP adrese, MAC adrese i portovi) povezuje jedinstveni identifikator koji se naziva mrežna utičnica odnosno ***network socket***. Tako se pristigli TCP paketi prepoznaju kao pripadajući određenoj (pojedinoj) TCP/UDP vezi i aplikaciji koja ih opslužuje, upravo prema navedenoj mrežnoj utičnici (tzv. *network socketu*). Na transportom sloju se u TCP/UDP zaglavlju, osim portova, dodaju i druga polja o kojima ćemo govoriti ubrzo.

**Priča o TCP/UDP portovima**
Svaka strana TCP (ili UDP) konekcije u komunikaciji ima dodijeljenu 16-bitnu oznaku kojom se identificira aplikacijski protokol, za obje strane komunikacije; kako za slanje tako i za primanje, koja se zove ***port***. TCP i UDP protokoli dopuštaju maksimalno 65.536 portova [$2^{16}$]. Nadalje, za svaki aplikacijski protokol je definiran ***port*** preko kojega se on koristi, primjerice: **HTTP**, **HTTPS**, **SSH**, **DNS**, **FTP**, **LDAP**, … .

To znači da pomoću ***portova*** svako računalo može u svakom trenutku paralelno komunicirati s više aplikacija (protokola) jer svaka od njih koristi drugačiji ***port.*** To također znači da poslužiteljsko računalo može pružiti uslugu za više klijenata istovremeno, sve dok se klijent brine o pokretanju istovremenih veza na jedan odredišni port s različitih izvorišnih portova.

*Internet Assigned Numbers Authority* (**IANA**) organizacija podijelila je **portove** u nekoliko raspona:

- **Rezervirane** i točno definirane portove iz opsega portova: **0-1023** u kojem su definirani poznati aplikacijski protokoli poput: **HTTP**, **HTTPS**, **SSH**, **DNS**, **FTP**, **SMTP**, **LDAP**, **RADIUS**, **SNMP**, … .
- **Registrirane** portove, registrirane od strane organizacija i tvrtki (iz opsega: **1024-49151**).
- **Dinamičke** portove odnosno sve ostale koji nisu prethodno definirani (obično iz opsega: **49152-65535**).
  - **Efemerni portovi** su unutar gornjeg opsega (ili posebnih opsega), a koriste se kao privremeni portovi:
    - Prema **RFC 6056**, oni su iz opsega: **1024–65535.**
    - Prema **IANA** i **RFC 6335** oni su iz opsega: **49152–65535**.
    - Za **Linux** je to često opseg: **32768–60999**, dok za duge operativne sustave ovisi kojeg se pravila pridržavaju.

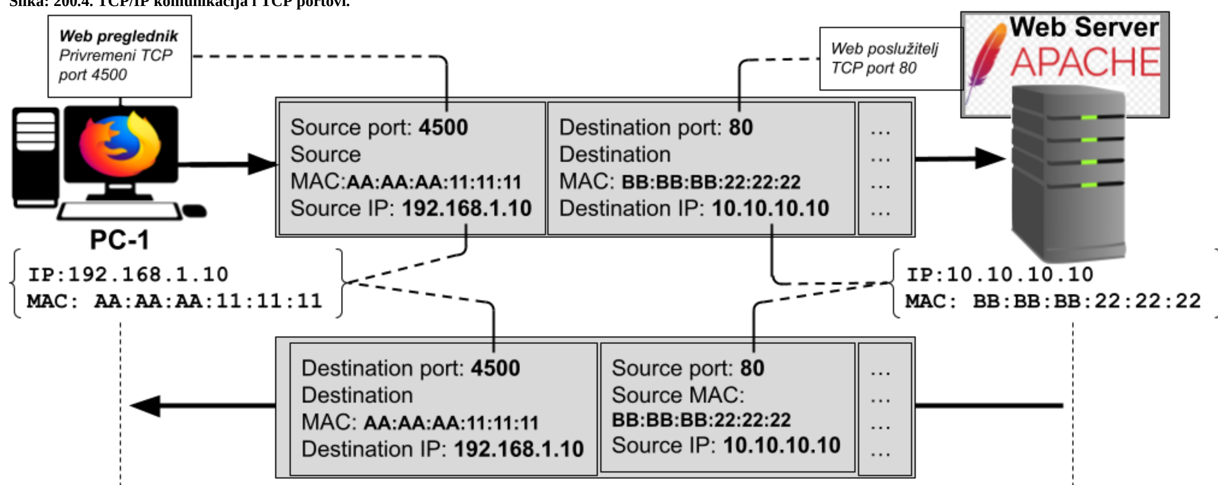


Vezano za TCP/UDP portove i dodatne detalje o njima, pogledajte i poglavlje:
**19.8.1. TCP/UDP Portovi**.

Ako pogledamo komunikaciju računala (**PC-1**) koje se s web preglednikom spaja na web poslužitelj (**web server**), događa se sljedeće (kako je vidljivo na slici 200.4.):

- U ovom slučaju Web preglednik otvara privremeni (**efemerni**) TCP port **4500** koji se veže za sam web preglednik te koristi taj port kao izvorišni. On koristi svoje MAC i IP adrese kao izvorišne. Dok kao odredišne MAC i IP adrese postavlja adrese web poslužitelja, na TCP port **80** jer je to prema standardu port za HTTP aplikacijski protokol.
- Web poslužitelj u svom odgovoru koristi svoje MAC i IP adrese kao izvorišne, te svoj TCP port **80** kao izvorišni jer se komunikacija odvija s njegovim **HTTP** (*Apache*) servisom koji radi na TCP portu **80**. On kao odredišne MAC i IP adrese postavlja adrese **PC-1** računala, kao i odredišni TCP port **4500** jer će on na strani računala (**PC-1**) biti prepoznat kao port na koji je spojen web preglednik i na koji će dostaviti tražene podatke.

**Slika: 200.4. TCP/IP komunikacija i TCP portovi.**




Pogledajmo sve i na najnižoj razini, na razini mrežnih utičnica (tzv. ***network socketa***).

Pogledajmo što se događa na strani klijenta to jest web preglednika (**PC-1**):

```
netstat -tunap | grep Firefox

Proto Recv-Q Send-Q Local Address          Foreign Address     State         PID/Program name
tcp        0      0 192.168.1.10:4500      10.10.10.10:80      ESTABLISHED   5666/Firefox
```

← Vidimo da je pokrenut web preglednik ***Firefox***, koji ima *PID* broj: `5666`. te da je on sa svoje IP adrese `192.168.1.10` i porta `4500` otvorio TCP konekciju prema web poslužitelju na udaljenoj IP adresi: `10.10.10.10` na portu 80 (`:80`).

Za taj *PID* broj web preglednika, provjerimo njegove *file deskriptore* koji su u ovom slučaju veza prema ***network socketu***:

```
ls -la /proc/5666/fd

dr-x------ 2 root root  3 Mar 13 11:36 .
dr-xr-xr-x 9 root root  0 Mar 13 11:36 ..
lrwx------ 1 root root 64 Mar 13 11:36 0 -> /dev/pts/1
lrwx------ 1 root root 64 Mar 13 11:36 2 -> /dev/pts/1
lrwx------ 1 root root 64 Mar 13 11:36 3 -> 'socket:[35499]'
```

← Vidimo vezu aplikacije (**Firefox**) sa *file deskriptorom* broj 3 prema ***network socketu*** broj: `35499`.



Vezano za ***file deskriptore*** pogledajte poglavlje:
**4.5.4. Opisnici datoteke (*File deskriptori*).**

A sada prema *PID* broju `5666`, pogledajmo vezu tog ***network socketa*** (`35499`) prema udaljenom **Web** poslužitelju na IP adresi: 10.10.10.10:

```
lsof -nP -p5666 | egrep -i "sock|IP"

COMMAND  PID  USER FD TYPE  DEVICE SIZE/OFF NODE NAME
Firefox  5666 root 3u IPv4  35499  0t0      TCP  192.1681.1.10:4500 -> 10.10.10.10:80
(ESTABLISHED)
```

← Jasno je vidljivo da *Firefox* proces s *PID* brojem `5666` s lokalnom IPv4 adresom `192.1681.1.10`, ima otvorenu TCP vezu prema Web poslužitelju na IP adresi: `10.10.10.10` na portu 80.

**Network socketi**
Otvaranjem svake nove (pojedine) veze prema udaljenom poslužitelju, stvara se jedinstveni ***network socket*** sa svojim identifikatorom (***INODE*** brojem), kako na strani klijenta, tako i na strani poslužitelja. Taj ***network socket*** se potom koristi u mrežnoj komunikaciji s drugom (udaljenom) stranom komunikacije. Dakle ***Network sockete*** koriste aplikacije kao jedinstvene poveznice na otvorene mrežne veze prema udaljenoj strani u komunikaciji.



Vezano za ***network sockete*** i dodatne detalje o njima, pogledajte i poglavlja:
**9.4.3.2. Network *socketi*.**
**25.1. Linux mrežni model.**

*Za ograničenja socket međumemorija pogledajte poglavlje:*
**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**



Svaka pojedina mrežna utičnica (**network socket**) nosi jedinstveni identifikator (***INODE*** broj), a veže se za pet parametara veze:
- Protokol; primjerice TCP ili UDP.
- Pâr: izvorišna IP adresa i njen pripadajući port te odredišna IP adresa i njen pripadajući port.

Mrežna utičnica (**network socket**) jedinstveni je identifikator svake pojedine otvorene komunikacijske veze.

**Izvori informacija: (771),(773),(1134),(1451),(K-6),(K-9)**, `man 7 udp`, `man 7 tcp`, **RFC 768**, **RFC 793**.

# 24.1. UDP

**UDP** (*User Datagram Protocol*) transportni protokol koristi jednostavan, ali vrlo brzi sustav za prijenos podataka bez provjere o uspješnom primanju ili integritetu podataka poslanih unutar mrežnih paketa. Zanimljivo je što unutar svakog paketa u *UDP* zaglavlju postoji polje unutar kojeg se izračunava i upisuje provjerni zbroj, koji se može koristiti za provjeru integriteta sâmo *UDP* zaglavlja, ali ne i sâmih podataka. Problem je u tome što je to opcija koja se obično ne koristi (ovisno o aplikacijskom protokolu). Dakle *UDP* uglavnom prepušta sve provjere na stranu sâme aplikacije koja šalje ili prima podatke, pa ako aplikacija ne radi nikakve provjere ispravnosti paketa (što je čest slučaj) ovakva komunikacija je potencijalno nepouzdana.

Dakle UDP protokol je nepouzdan, ali vrlo brz. Želite li pouzdanost, a morate ili želite koristiti *UDP*, sve zaštitne mehanizme morate ugraditi u vašu aplikaciju. UDP kao transportni protokol koristi se za: ***Broadcast***, ***Multicast*** i ***Anycast*** komunikaciju!
Ako ipak želite sâmo brzinu onda je *UDP* pravi izbor. UDP se često koristi kada imamo komunikaciju koja je vremenski osjetljiva, a nije nam previše važno hoće li se koji paket usput izgubiti u prijenosu.
Primjer ovakve upotrebe je audio ili video komunikacija, gdje nam nije važno, ako se na trenutak izgubi koji dio slike ili djelić glasa, ali nam je važno da nema vremenskog kašnjenja. Osim ove primjene UDP se koristi i kod drugih vrsta komunikacije kao što su razni protokoli za logiranje poruka (spremanje sistemskih poruka s raznih uređaja na centralnu lokaciju) kada nam nije najbitnije, ako se jedna mala poruka izgubi i slično.
Još jedan primjer su **DNS** (*Domain Name Service*) protokol ili **NTP** (*Network Time Protocol*) protokoli koji, ako ne dobiju odgovor u nekom vremenu, poruku će poslati ponovno.
Koriste ih i **DHCP** (*Dynamic Host Configuration Protocol*), **TFTP** (*Trivial FTP*) kao i mnogi drugi mrežni protokoli.

**Izvori informacija: (1134),(1451),(K-6),(K-9)**, `man 7 udp`, **RFC 768**.

### 24.1.1. Oblik UDP poruke (paketa)

Sada ćemo analizirati **UDP** poruke odnosno pakete. **UDP** poruka (paket) sastoji se od sljedećih polja:

| **UDP zaglavlje** | |
|---|---|
| Izvorišni port (*Source port*) | Odredišni port (*Destination port*) |
| Length | Checksum |

Slijedi opis polja **UDP** zaglavlja:

- `Source port` je port pošiljaoca (izvorišni *port*) paketa. Ako se radi o paketu od klijenta, onda je ovaj port lokalni port na kojem klijent otvara ovu komunikaciju. Ako se radi o paketu koji dolazi od poslužitelja, onda je ovaj port lokalni poslužiteljev port odnosno onaj koji označava vrstu protokola odnosno sâm servis tj. protokol (pr. port 80 je zadužen za HTTP protokol, dok je port 53 zadužen za DNS protokol, port 22 za SSH protokol i tako dalje).

- `Destination port` je port primatelja (odredišni *port*) paketa. Ako se radi o paketu koji šalje klijent prema poslužitelju, ovaj port označava servis odnosno protokol. Ako se primjerice radi o portu 53, onda se radi o DNS protokolu. U suprotnom smjeru: poslužitelj→ klijent, port 53 postaje izvorišni (*source*) *port*, a odredišni (*destination*) *port* je port koji je klijent lokalno otvorio i koji je praktično njegov identifikator aplikacije koja ga koristi.

- `Length` – u ovom polju se upisuje veličina u *bajtima* od UDP zaglavlja i podataka (zbirno). Maksimalna veličina je standardno 65535 *bajta* što se odnosi na cijeli paket: UDP zaglavlje + DATA dio koji čine IP zaglavlje i podaci.

- `Checksum` – ovdje se zapisuje provjerni zbroj (*checksum*) sâmo od UDP zaglavlja. Ovo polje je opcionalno. Ako nije u upotrebi (što je obično slučaj) popunjeno je nulama.



*Veličina cijelog* **UDP** *zaglavlja je 8* ***bajta.***

Pogledajmo i jedan *UDP* paket na mreži, s označenim OSI slojevima [1-4] (malo smo skratili ispis):

```
(1)  Frame 54: 170 bytes on wire (1360 bits),170 bytes captured (1360 bits) on interface 0
(2)  Ethernet II, Src: (74:2b:62:89:1a:f6), Dst: (08:00:27:93:e8:86)
(3)  Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.100
(4)  User Datagram Protocol, Src Port: 63927, Dst Port: 53
        Source Port: 63927
        Destination Port: 50007
        Length: 136
        Checksum: 0x482c [unverified]
        [Checksum Status: Unverified]
        [Stream index: 4]
     Data (128 bytes)
       Data: 4ab57fe52143bdf32d9edafff9b6b33782f6c4bdf2a02172...
       [Length: 128]
```

Raščlanimo ispis prema OSI slojevima:

- Vidimo OSI sloj **(1)** te OSI sloj **(2)** u kojem se vide **MAC** adrese: `Src MAC: 74:2b:62:89:1a:f6` i `Dst MAC: 08:00:27:93:e8:86`. Pri tome je odredišna **MAC** adresa (`Dst MAC`) računala, **MAC** adresa mrežnog sučelja koje prima ovaj paket, a izvorišna (`Src MAC`) **MAC** adresa je **MAC** (*Ethernet*) adresa naše mrežne kartice.

- Na OSI sloju **(3)** odnosno IP je izvorišna (naša) IP adresa (adresa pošiljaoca) `Src: 192.168.1.10` te odredišna IP adresa `Dst: 192.168.1.100` na koju se paket šalje.

- Na OSI sloju **(4)** je vidljivo kako se radi o *UDP* protokolu gdje vidimo izvorišni port `Src Port: 63927` i odredišni port `Dst Port: 53`. Odredišni port govori da se radi o DNS servisu na koji se spajamo na drugu (DNS poslužiteljsku) stranu. Potom su vidljivi i drugi dijelovi *UDP* zaglavlja unutar paketa. Pri tome je razumljivo kako je: `Data (128 bytes)` dio, dakle *DATA* dio (u kojemu su stvarni aplikacijski podaci), veličine 128 *bajta*, a polje `Length: 136`. Polje *Length* sadrži ukupnu veličinu paketa: *UDP* zaglavlje i *DATA* dio, pri čemu je *DATA* dio u ovom slučaju 128 *bajta* a UDP zaglavlje je standardno 8 *bajta*; pa dobivamo: 128+8=136 što odgovara *Length* polju.

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na UDP sloju. Sintaksa ove naredbe koju ćemo korisiti jest:
***`nstat -opcije ImeBrojača`***

Primjerice, ako nas zanima brojač imena `UdpInDatagrams`:

```
nstat -az UdpInDatagrams
#kernel
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `UdpInDatagrams` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za UDP sloj:

| Naziv brojača | Opis |
|---|---|
| **`UdpInDatagrams`** | Ukupan broj primljenih UDP *datagrama (paketa)* isporučenih UDP sloju. |
| **`UdpNoPorts`** | Ukupan broj primljenih UDP *datagrama* za koje nije bilo odredišnog mrežnog servisa (aplikacije) na odredišnom portu. |
| **`UdpInErrors`** | Broj primljenih UDP *datagrama* koji se nisu mogli isporučiti iz drugih razloga osim nepostojanja aplikacije na odredišnom portu. |
| **`UdpOutDatagrams`** | Ukupan broj UDP *datagrama* poslanih s ovog računala. |
| **`UdpRcvbufErrors`** | Povećava se kada se ne može dodijeliti memorija za obradu dolaznog UDP *paketa* odnosno kada nema dovoljno memorije za prijem. To se može događati zbog *socket* međumemorije, koju možemo vidjeti kao "**Recv-Q**". Primjerice s naredbom:<br>**`ss -nt`**<br>Ili zbog međumemorija koje možemo povećati sa sljedećim *sysctl* varijablama:<br>`net.core.`**`r`**`mem_max`<br>`net.core.`**`r`**`mem_default`<br>`net.ipv4.udp_mem`<br>`net.ipv4.udp_`**`r`**`mem_min` |
| **`UdpSndbufErrors`** | Povećava se kada se memorija ne može dodijeliti za slanje UDP paketa. To se može događati zbog *socket* međumemorije, koju možemo vidjeti kao "**Send-Q**". Primjerice s naredbom:<br>**`ss -nt`**<br>Ili zbog međumemorija koje možemo povećati sa *sysctl* varijablama:<br>`net.core.`**`w`**`mem_max`<br>`net.core.`**`w`**`mem_default`<br>`net.ipv4.udp_mem`<br>`net.ipv4.udp_`**`w`**`mem_min` |
| **`UdpInCsumErrors`** | Povećava se kada primljeni UDP paket ima nevažeći kontrolni (provjerni) zbroj. |
| **`UdpIgnoredMulti`** | Povećava se za UDP *multicast* pakete. |
| **UDP Lite** | |
| **`UdpLiteInDatagrams`** | Ukupan broj ***UDP-Lite*** *datagrama* koji su isporučeni UDP-Lite korisnicima. Prekidi u vrijednosti ovog brojača mogu se pojaviti prilikom ponovnog pokretanja sustava upravljanja te u drugim slučajevima kao što je naznačeno vrijednošću `udpliteStatsDiscontinuityTime`. |
| **`UdpLiteNoPorts`** | Ukupan broj primljenih UDP-Lite *datagrama* za koje nije bilo nije bilo odredišnog mrežnog servisa (aplikacije) na odredišnom portu. |
| **`UdpLiteInErrors`** | Broj primljenih UDP-Lite *datagrama* koji se nisu mogli isporučiti iz drugih razloga osim nepostojanja aplikacije na odredišnom portu. |
| **`UdpLiteOutDatagrams`** | Ukupan broj UDP-Lite *datagrama* poslanih s ovog računala. |
| **`UdpLiteRcvbufErrors`** | Povećava se kada se ne može dodijeliti memorija za obradu dolaznog UDP-Lite *paketa* odnosno kada nema dovoljno memorije za prijem. Pogledajte: **`UdpRcvbufErrors.`** |
| **`UdpLiteSndbufErrors`** | Povećava se kada se ne može dodijeliti memorija za obradu odlaznog UDP-Lite *paketa* odnosno kada nema dovoljno memorije za slanje. Pogledajte: **`UdpSndbufErrors.`** |
| **`UdpLiteInCsumErrors`** | Povećava se kada primljeni UDP-Lite paket ima nevažeći kontrolni (provjerni) zbroj. |
| **`UdpLiteIgnoredMulti`** | Povećava se za UDP-Lite *multicast* pakete. |

**UDP Lite** je varijanta *User Datagram Protocola* (*UDP*) koja se koristi za prijenos podataka kroz mrežu, ali s određenim dodatkom to jest s mogućnošću postavljanja selektivne detekcije grešaka.

Osnovna razlika između *UDP Lite* i običnog UDP-a je u načinu obrade grešaka. UDP Lite omogućuje postavljanje prilagodljivog polja duljine unutar zaglavlja UDP paketa, koje se naziva "**checksum coverage field**".
Ovo polje omogućuje primatelju da selektivno provjeri samo određeni dio podataka u paketu, dok ostatak paketa može biti oštećen bez utjecaja na ispravnost ostalih dijelova.
UDP Lite je definiran u **RFC 3828**.

**Izvori informacija: (1134),(1451),(1651),(K-6),(K-9)**,**`man 7 udp`**,**`man nstat`**,**RFC 768, RFC 1213, RFC 3828**.

# 24.2. TCP

**TCP** je transportni protokol, protokol koji se brine za stanje veze (tzv. *connection-oriented*) kao i za integritet podataka.
To znači kako se u komunikaciji dva računala otvara konekcija pomoću tzv. metode trostrukog rukovanja (engl. *Three-way handshake*) i TCP protokol se nadalje brine da je svaki segment podataka koji je poslan s jednog računala na drugo računalo, primljen u ispravnom (nepromijenjenom) obliku te pravilnim redoslijedom. Za svaki paket koji je s druge strane zaprimljen neispravan, zatraži se *retransmisija* i paket se šalje ponovno, sve dok se ne zaprimi potpuno ispravan. Kako bi se uopće mogla provjeriti ispravnost (integritet) podataka (prema **RFC 793**) unutar svakog paketa u TCP zaglavlju postoji polje za koje se izračunava i upisuje *provjerni* zbroj (Engl. *checksum*). Dakle ako unutar TCP zaglavlja dođe do grešaka u prijenosu, to će biti detektirano. Tijekom zatvaranja veze koristi se metoda dogovaranja oko zatvaranja veze, slična kao i kod otvaranja veze (konekcije). Naravno u prijenosu podataka kroz medij (kabel, optičko vlakno, zrak) može doći, i dolazi, do gubitka djela podataka na što je TCP otporan odnosno što on uspijeva popraviti.


***TCP kao transportni protokol ne može se koristiti za: Broadcast, Multicast i Anycast komunikaciju!***

Pogledajmo **TCP zaglavlje** sa svim poljima, s čijim detaljnim funkcionalnostima i mehanizmima ćemo se kasnije upoznati:

<table>
<tr><td colspan="4">Source Port (izvorišni port)</td><td colspan="5">Destination Port (odredišni port)</td></tr>
<tr><td colspan="9">Sequence Number</td></tr>
<tr><td colspan="9">Acknowledgment (ACK) Number</td></tr>
<tr><td rowspan="2">Data Offset</td><td rowspan="2">Reserved</td><td colspan="6">FLAGS</td><td rowspan="2">Window</td></tr>
<tr><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td></tr>
<tr><td colspan="4">Checksum</td><td colspan="5">Urgent Pointer</td></tr>
<tr><td colspan="4">Options (opcije)</td><td colspan="5">Padding</td></tr>
<tr><td colspan="9">DATA (podaci)</td></tr>
</table>


*Veličina cijelog* **TCP** *zaglavlja je* ***20 bajta****, ali može se povećati sve do* **60 bajta** *(pomoću polja* ***Offset****).*

Slijedi opis polja ***TCP*** zaglavlja:

- `Source Port` - 16 bitni broj, označava izvorišni (*source*) *port*. Ovo je port pošiljaoca (izvorišni *port*) paketa. Ako se radi o paketu od klijenta, onda je ovaj port lokalni (***efemerni***) port na kojem klijent otvara ovu komunikaciju. Ako se radi o paketu koji dolazi od poslužitelja, onda je ovaj port lokalni poslužiteljev port odnosno onaj koji označava vrstu protokola odnosno sâm servis tj. protokol (pr. port 80 je zadužen za HTTP protokol).


Pogledajte poglavlje: **19.8.1. TCP/UDP Portovi**.

- `Destination Port` - 16 bitni broj, označava odredišni (*destination*) *port*. Ovo je port primatelja (odredišni *port*) paketa. Ako se radi o paketu koji šalje klijent prema poslužitelju, ovaj port označava servis odnosno protokol. Ako se primjerice radi o portu 53, onda se radi o DNS protokolu. U suprotnom smjeru: poslužitelj→ klijent, port 53 postaje izvorišni (*source*) *port*, a odredišni (*destination*) *port* je port koji je klijent lokalno otvorio i koji je praktično njegov identifikator aplikacije koja ga koristi (takozvani **efemerni** port).


Pogledajte poglavlje: **19.8.1. TCP/UDP Portovi**.

- `Sequence Number` - 32 bitni broj, označava nîz slijeda podataka koji se šalju.
- `Acknowledgment Number` - 32 bitni broj, kojim se potvrđuje primitak podataka, sve do *bajta* koji se potvrđuje +1 (+1 je prvi slijedeći *bajt*).
- `Data offset` - 4 bitni broj unutar kojeg se definira veličinu cijelog TCP zaglavlja. Standardno je TCP zaglavlje veličine 20 bajta, a može se proširiti za 40 bajta (u ***opcijama***), na ukupno 60 bajta.
- `Reserved` - 6 bitova, rezervirano za buduću primjenu.
- `FLAGS` odnosno kontrolni bitovi (proširena inačica, prema RFC 3168), svaki bit pri tome uključuje ili isključuje određenu zastavicu, koje mogu biti:
  - `Nonce` - ***Nonce sum*** odnosno ***NS,*** koristi se u kombinaciji sa ***CWR*** i ***ECE*** (*ECN-Echo*), za *ECN* (*Explicit Congestion Notification*).


Pogledajte poglavlje: **24.2.9.3. Explicit Congestion Notification (*ECN*).**

- `CWR` - ***Congestion Window Reduced*** (**CWR**) indikacija da se radi o *TCP* paketu koji ima postavljenu *ECE* zastavicu (RFC 3168).


Pogledajte poglavlje: **24.2.9.3. Explicit Congestion Notification (*ECN*).**

- `ECE` (**`ECN-Echo`**) - indicira kako TCP sugovornik (druga strana komunikacije) ima sposobnost **ECN** (***Explicit Congestion Notification***) odnosno detekciju zagušenja mreže. Dogovara se u tijeku inicijalizacije TCP komunikacijskog kanala (dodano od RFC 3168).


Pogledajte poglavlje: **24.2.9.3. Explicit Congestion Notification (*ECN*).**

- `URG` - indicira kako je **URG**ent polje u opcijama aktivno.


Pogledajte poglavlje **24.2.10. TCP Push (*PSH*) i Urgent (*URG*).**

- `ACK` - indicira koliko zadnjih podataka (indirektno koji paket) je uredno zaprimljen.


Pogledajte poglavlje: **24.2.5. Standardne potvrde (*ACK*).**

- `PSH` - indicira se ***PUSH*** metoda slanja paketa.


Pogledajte poglavlje: **24.2.10. TCP Push (*PSH*) i Urgent (*URG*).**

- `RST` - indicira se resetiranje TCP komunikacijskog kanala.


Pogledajte poglavlje: **24.2.12. TCP reset**.

- `SYN` - indicira se sinkronizacijski paket.


Pogledajte poglavlje: **24.2.2. Kako se uspostavlja TCP veza**.

- `FIN` - indicira se zatvaranje TCP komunikacijskog kanala

- `Window` (`Window size`) - 16 bitova - veličina podataka koju klijent može *trenutno* prihvatiti.
- `Checksum` - 16 bitova - ovo je provjerni zbroj (*checksum*) **SAMO** TCP zaglavlja, ali ne i podataka (DATA).
- `Urgent Pointer` 16 bitova - ako je zastavica (*Flag*) `URG` postavljena na "1" - označava koliko *bajta* podataka, od ukupno poslanih, je važno (*Urgent*).

- `Options` minimalno 8 bitova do dva puta po 8 bitova. Prvih osam (8) bitova se zove `Kind`, a drugih osam `Length`. Često korištene TCP opcije su (navodimo sâmo njih nekoliko):
  - Kind:`0` = *End of option list (kraj liste s opcijama).*
  - Kind:`1` = *No-Operation.*
  - Kind:`2` Length:4 = *Maximum Segment Size* **MSS**.


Pogledajte poglavlje: **24.2.1. Maximum segment size (*MSS*).**

  - Kind:`3` Length:3 = *Window Scale*.


Pogledajte poglavlje: **24.2.8.2. Skaliranje TCP prozora (*Window Scaling*)**.

  - Kind:`4` Length:2 = *SACK Permitted*.


Pogledajte poglavlje: **24.2.6. Selektivne potvrde (*SACK*)**.

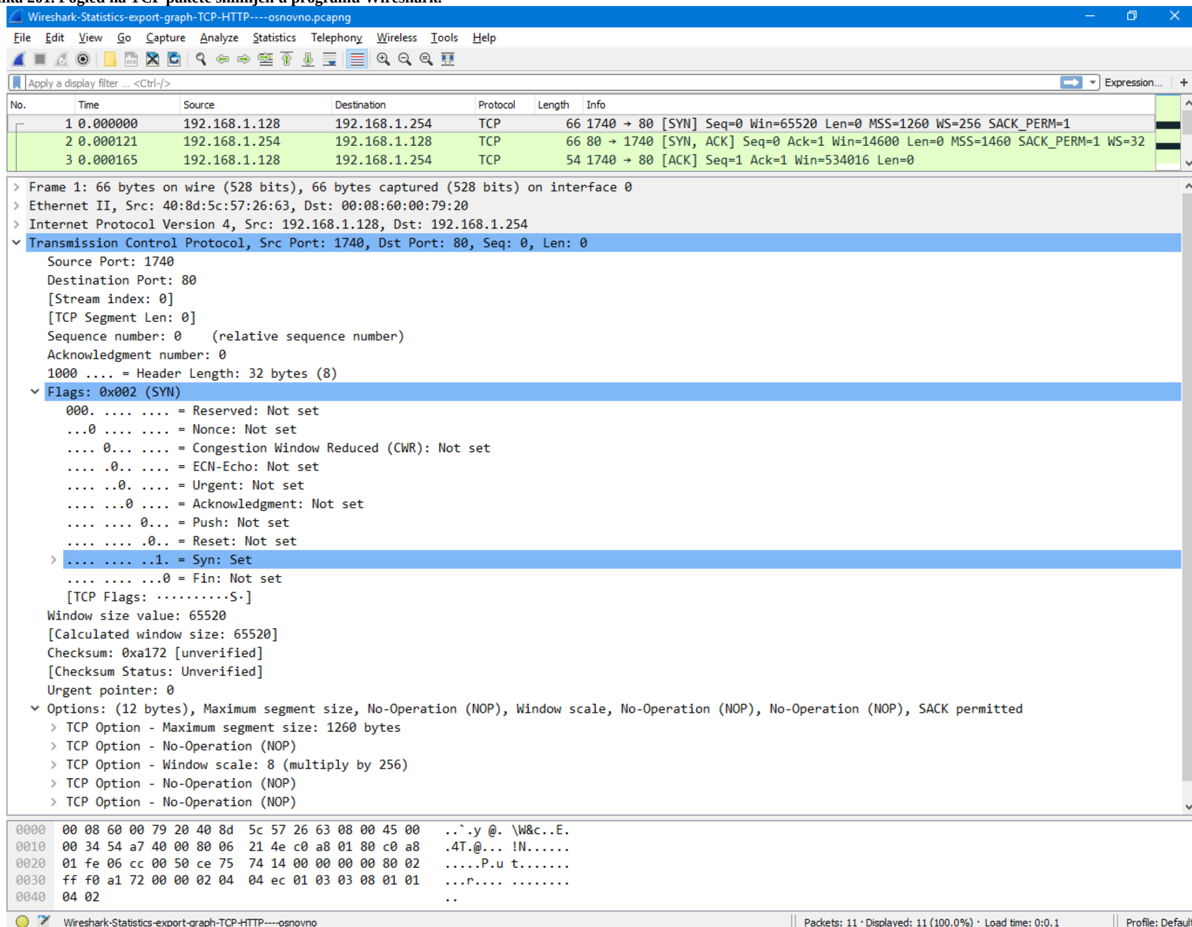
  - Kind:`5` = ***SACK***.

**Lista svih opcija nalazi se definirana na poveznici:**
**https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1**

- `Padding` - označava kraj TCP dijela paketa/segmenta. Popunjen je nulama sve do 32 bita zbrojeno s opcijama (`Options`).

Pogledajmo i jedan TCP paket u kojemu se naše računalo (*IP: 192.168.1.128*) s Web preglednikom spaja na Web poslužitelj na IP adresi: *192.168.1.254*. Promet je snimljen s programom **Wireshark.** Na slici 201 je fokus na zastavice (`Flags:`) i opcije.

**Slika 201. Pogled na TCP pakete snimljen u programu Wireshark.**




## Sada još malo bacimo pogled na *portove.*



Vezano za portove, podsjetimo se detalja u poglavlju:
**19.8.1. TCP/UDP Portovi**.

Naime, ako gledamo komunikaciju klijenta prema poslužitelju, kao u ovom primjeru na slici; naše računalo s IP adrese ***192.168.1.128*** se spaja na web poslužitelj koji ima IP adresu ***192.168.1.254***. Pri tome je naše računalo na IP adresi: ***192.168.1.128,*** otvorilo nasumični lokalni, takozvani **efemerni**, ***TCP*** port (***SRC***) broj ***1740*** koji je identifikator ove veze za naše računalo. Odnosno za naš Web preglednik s kojim smo otvorili vezu prema udaljenom poslužitelju, koje je u ovom slučaju web poslužitelj. S druge strane Web poslužitelj ima otvoren TCP port ***80*** na IP adresi: ***192.168.1.254,*** na koju se i spaja naše računalo odnosno naš Web preglednik. Naime nama je odredišni TCP port ***80*** na web poslužitelju s IP adresom: ***192.168.1.254***, upravo onaj port na koji se spajamo.



***TCP port 80 (na poslužitelju) je definiran kao rezervirani port za HTTP protokol (Hyper Text Transfer Protocol).***

U slučaju suprotne komunikacije: web poslužitelj → naše računalo, događa se sljedeće:
Kada nam Web poslužitelj odgovori, on odgovara sa svoje IP adrese; koja je sada izvorišna (***SRC***) IP adresa: ***192.168.1.254***, s koje se paket šalje na našu odredišnu IP adresu: ***192.168.1.128.***
Nadalje, sada je njegov izvorišni TCP port ***80*** jer je to port njegovog Web (HTTP) servisa, ali je njemu odredišni (***DST***) port onaj koji smo mi kao klijent postavili na: ***1740*** *(to je **efemerni** port klijenta)*. Taj isti odredišni TCP port: ***1740*** je Web poslužitelju također identifikator veze prema nama, odnosno prema IP adresi: ***192.168.1.128***.

**Izvori informacija: (1134),(1135),(1451),(1477),(K-6),(K-9), `man 7 tcp`, RFC 793, RFC 3168.**

*U narednim cjelinama upoznat ćemo se sa svim mehanizmima rada TCP protokola.*

## 24.2.1. Maximum segment size (*MSS*)

U prethodnim poglavljima, naučili smo kako **MTU** vrijednost definira maksimalnu veličinu mrežnih okvira (paketa) na razini mreže (*ethernet*) odnosno mrežnih okvira koji će biti poslani na mrežu, za određeno mrežno sučelje. Međutim na TCP razini imamo još jednu vrijednost koja se naziva *Maximum segment size* (***MSS***) odnosno maksimalnu veličinu (*TCP*) segmenta (mrežnog paketa).

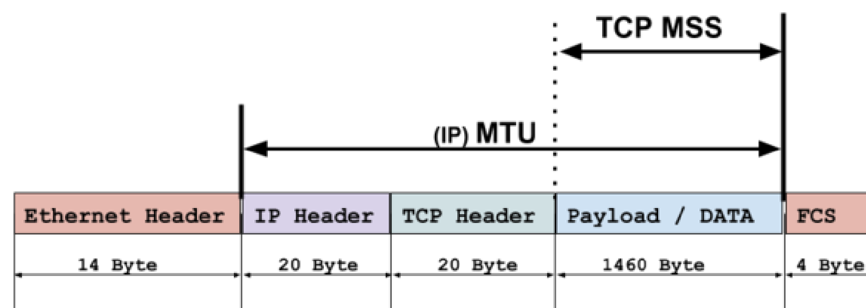
***MSS praktično označava maksimalnu veličinu podataka unutar TCP sloja, koja naravno mora biti manja od MTU veličine.*** Naime od trenutka uspostavljanja TCP veze (od prve poruke: ***TCP SYN***), klijent prvo prema poslužitelju u opcijama šalje svoju ***MSS*** veličinu, koja može biti primjerice 1460 bajta, što znači kako on neće slati pakete veće od **MTU** u svakom slučaju, odnosno da unutar TCP paketa i to sâmo *DATA* dijela (s podacima) neće slati više od 1460 bajta. Međutim poslužitelj (druga strana komunikacije) mu može odgovoriti u *TCP SYN ACK* poruci, kako on neće slati više od 1380 bajta unutar TCP *DATA* dijela zbog bilo kojeg razloga: primjerice jer između sebe i klijenta koristi neko *tuneliranje* pa stoga ima manji ***MTU***.

Klijentska strana to treba prihvatiti i od tog trenutka neće niti ona prema tom konkretnom poslužitelju slati više od 1380 bajta unutar TCP *DATA* dijela (s podacima) kako ne bi uzrokovao probleme u prijenosu paketa preko mreže. Naime tako će ukupna veličina mrežnog okvira ostati unutar MTU granice od **1500** bajta jer tada imamo:
**1380** (*TCP DATA=MSS*) + **20** (uobičajeno) TCP + **20** IP = **1420** bajta pa ostaje još 80 bajta za druga zaglavlja, ako će se trebati koristiti; kao što su primjerice protokoli za *tuneliranje (GRE)* koji dodaju i svoje zaglavlje, jer je tako tražila poslužiteljska strana odnosno druga strana u komunikaciji.

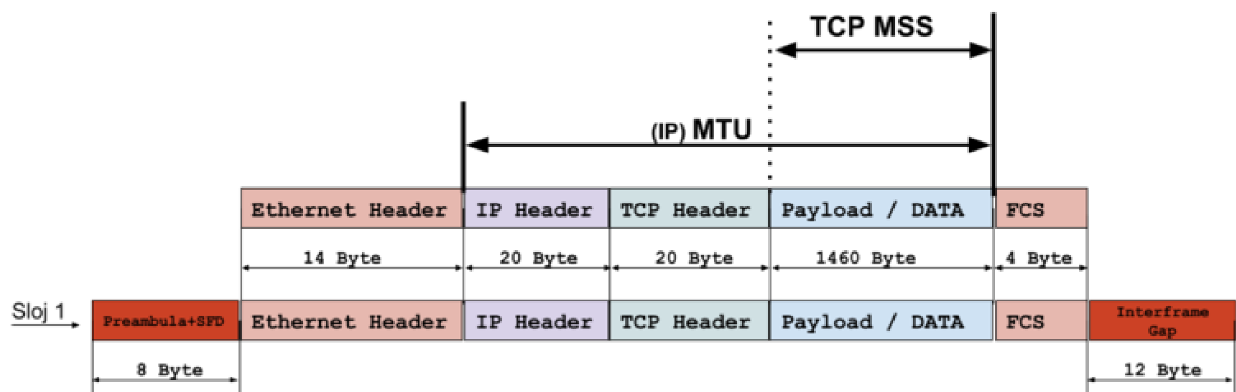
Dakle ***MSS*** je parametar polja opcija *TCP* zaglavlja koji određuje najveću količinu podataka navedenih u bajtima koje računalo ili komunikacijski uređaj može primati u jednom TCP segmentu. Pri tome se ne broji *TCP* zaglavlje niti *IP* zaglavlje, za razliku od primjerice *MTU*a za *IP datagrame*. *IP datagram* koji sadrži *TCP* segment može biti samostalan unutar jednog paketa ili se može sastojati iz nekoliko fragmentiranih dijelova. Bez obzira na to radi li se o prvom ili drugom slučaju ***MSS*** ograničenje odnosi se na ukupnu količinu podataka sadržanih u konačno rekonstruiranom *TCP segmentu (paketu)*.
***MSS*** je definiran u RFC 879.

Naime *TCP* segmenti su poruke koje nose podatke u komunikaciji između uređaja u *TCP* mrežnoj komunikaciji.
Podatkovno polje (*DATA* dio) je mjesto gdje se prenose stvarni podaci koji se transportiraju, a budući da je duljina podatkovnog polja u *TCP* promjenjiva, pitanje je koliko se maksimalno podataka može spremiti unutar jednog *TCP* segmenta koji pojednostavljeno možemo promatrati kao jedan logički paket, ako ga gledamo sa *TCP* sloja. Možemo reći i sljedeće: ***MSS*** se odnosi na maksimalnu količinu podataka koje jedan TCP segment može nositi, što ne uključuje *TCP* i IP zaglavlja, već sâmo podatkovni (*DATA)* dio. Odnos između ***MTU*** i ***MSS*** možemo promatrati na sljedeći način, kako je vidljivo na slici **202**.

**Slika 202. Odnos: MTU i MSS**




Odnosno ako gledamo i **OSI** sloj jedan (OSI **1**), tada to sve izgleda kako je vidljivo na slici **202.1**.
**Slika 202.1. Pogled na cijeli mrežni paket te odnos MTU i MSS veličine.**





Pogledajte i poglavlje o **MTU** vrijednostima:
**23.4.2. Maximum Transmission Unit (*MTU*).**

Sada u tablici pogledajmo *Ethernet* mrežni okvir (paket) na malo detaljnijoj razini:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | `OSI 5 - 7`<br>`Aplikacijski podaci`<br>`← 46 - 1460 bajta →`<br>↓↓↓↓↓↓↓↓↓↓↓↓<br>↓↓↓↓ | | | |
| ***OSI 1*** | ***OSI 1*** | ***OSI 2*** | ***OSI 2*** | ***OSI 2*** | ***OSI 3*** | ***OSI 4*** | | ***OSI 2*** | ***OSI 1*** |
| ***Preambula*** | *SFD (start frame delimiter)* | *Destination MAC* | *Source MAC* | *Ether Type (za OSI 3)* | ***IP***<br>*Source IP, Destination IP, Protokol (za OSI 4), TTL, Flags, ToS, …* | ***TCP ili UDP***<br>***Za TCP:***<br>**Source port**<br>**Destination port**<br>***SEQ Nr.***<br>***ACK Nr.***<br>***Window size***<br>***Flags***<br>***URG***<br>***Checksum***<br>**…** | ***Za UDP:***<br>**Source port**<br>**Destination port**<br>***Checksum***<br>***Length*** | *Frame check sequence FCS (CRC)* | ***Interframe gap (IFG)*** |
| | | | | | | *20 – 60 bajta* | *8 bajta* | | |
| | | | | | ***20 bajta*** | `46 - 1460 bajta podaci`<br>`(← TCP MSS →)`<br>`+`<br>`20 - 60 bajta TCP`<br>`ili`<br>`8 bajta UDP` | | | |
| | | ***6 bajta*** | ***6 bajta*** | ***2 bajta*** | `← MTU veličina do 1500 bajta →` | | | ***4 Bajta*** | |
| ***7 bajta*** | ***1 bajt*** | *Ethernet sloj* ← ***veličina do 1518 bajta*** → | | | | | | | ***12 bajta*** |
| ← ***Ukupno svih podataka „na mreži“ do prosječno 1538 bajta*** → | | | | | | | | | |

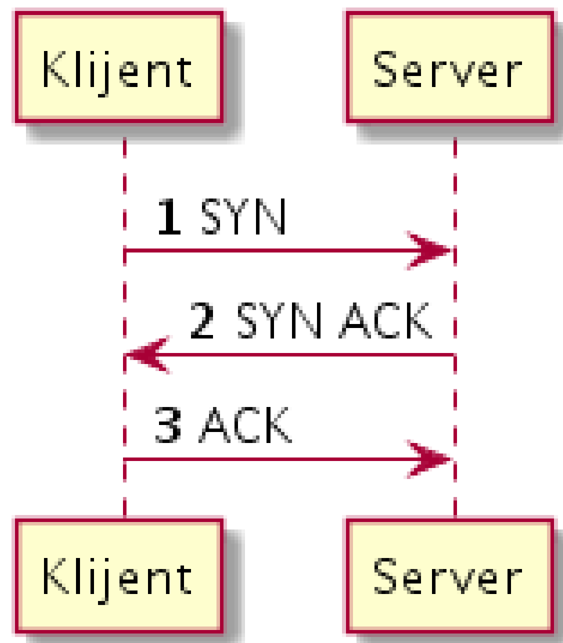
**Izvori informacija: (707),(1477),(K-6),(K-10), `man 7 tcp`, RFC 879.**

## 24.2.2. Kako se uspostavlja TCP veza

TCP veza se uspostavlja putem tkzv. *Three-way handshake* metode odnosno uspostave veze u tri koraka. Pogledajmo kako to izgleda (na slici 203. dolje), objašnjeno kroz sljedeće korake.

**Pogledajmo što se događa prilikom otvaranja odnosno uspostavljanja TCP veze**

**Slika 203. Uspostavljanje TCP veze u tri koraka.**




**Dakle, svaka TCP veza uspostavlja se u tri koraka:**
**1.** Računalo (***Klijent***) koje uspostavlja vezu šalje `TCP SYN` poruku na drugo računalo (***Server***). Također klijent postavlja `sequence number` (praktično redni broj paketa) na slučajnu vrijednost koja se koristi dalje. Primjer ovog broja (sekvence)je: heksadecimalno (dekadski):
`SEQ= F5 72 B7 12 (4117935890)`


Programi za praćenje mrežnog prometa, poput programa Wireshark zbog lakšeg razumijevanja, **SEQ** brojeve označavaju relativnim brojevima (kreću od 0) kako bi ih (ljudi) lakše pratili.

U našem primjeru, koristit ćemo relativne brojeve kako ćemo ih i vidjeti u programu ***Wireshark***. Dakle prvi paket `TCP SYN` ima `SEQ` broj **0** i `ACK` broj **0** jer se tek kreće s komunikacijom.
`Sequence number` je poput rednog broja svakog paketa koji se šalje, kako bi se kasnije na drugoj strani koja ih prima, mogli poredati prema redoslijedu kojim su poslani.

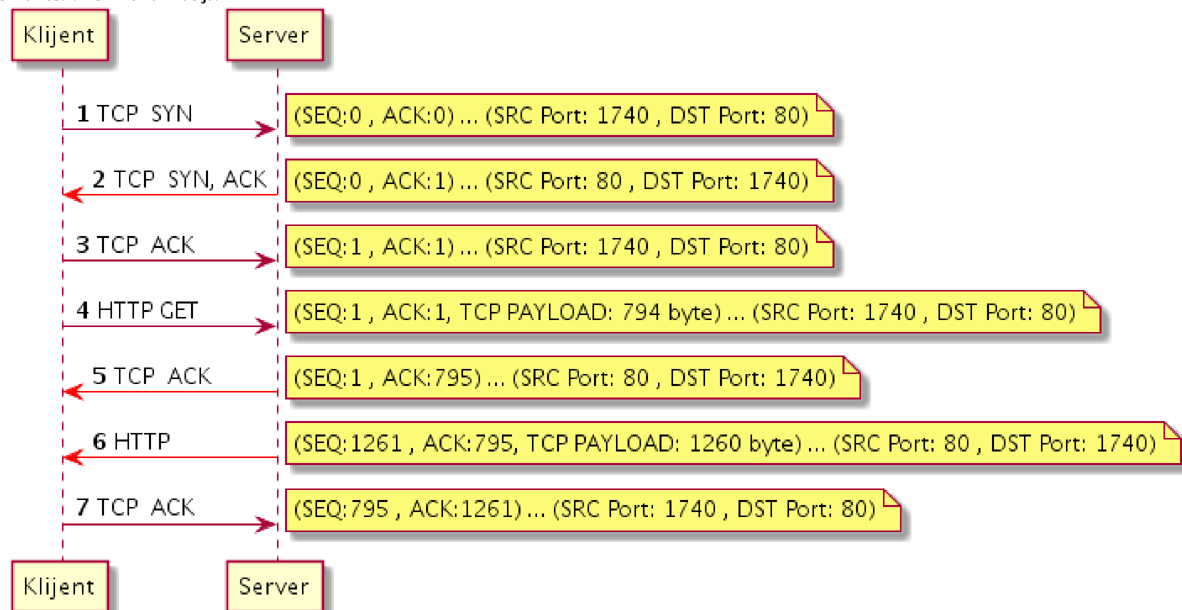
**2.** ***Server*** šalje `TCP SYN-ACK` poruku te postavlja `acknowledgment number` za jedan broj veći od `sequence number` broja iz prvog koraka. Dakle odgovara sa `SEQ 0` i `ACK 1`.

**3.** ***Klijent*** šalje `TCP ACK` poruku nazad na server, sa `SEQ 1` jer je ovo njegova sljedeća poruka u nîzu, te potvrđuje primljenu poruku broj **2.** s `ACK 1`. Ovo je zadnji korak u uspostavi *TCP* komunikacijskog kanala između dva računala.


Stvarni **ACK** i **SEQ** brojevi su 32-bitni brojevi, vidljivi u heksadecimalnom dijelu svakog paketa!

Nakon ovog procesa *TCP* veza između klijentskog i poslužiteljskog (*serverskog*) računala ili bilo koja druga dva računala ili mrežna uređaja je uspostavljena, te sada kreće komunikacija na aplikacijskoj razini. Sada pogledajmo kako detaljnije izgleda ova komunikacija, kao i dio komunikacije koje potom slijede. U primjeru ćemo nastaviti s komunikacijom klijenta i poslužitelja (*servera*), koji je u ovom primjeru web poslužitelj kako je vidljivo na slici **203.1**.

**Slika 203.1. TCP Komunikacija.**




Nakon uspostavljene TCP veze kreće promet preko nje.

Prvi paket (**1**) u komunikaciji nastavlja s označavanjem `SEQ` i `ACK` brojeva s time da se oni više ne povećavaju za **1** već za veličinu tzv. *Payload*-a odnosno veličinu koliko se podataka prenosi. Dakle `ACK` na drugoj strani (paket broj **2**) standardno potvrđuje primljeni `SEQ` broj te kreira novi koji je uvećan za taj *Payload* odnosno veličinu prenesenih podataka, kako vidimo u paketu broj **3**.

**4.** Paket šalje klijent, a ovo je prvi aplikacijski paket (**4**). Ovdje se radi o *HTTP* komunikaciji. Dakle ovdje bi to bio `HTTP GET` upit prema *web* (http) poslužitelju (***server*** na slici). `SEQ` broj je ostavljen na **1** pošto se nisu počeli prenositi podaci unutar *DATA/Payload* dijela paketa u tom smjeru, a zbog toga je i `ACK` broj ostao na **1**. Ovaj paket sadrži podatke u *payload* dijelu (konkretan HTTP upit), veličine 794 *bajta*.

**5.** Paket (**5**) je odgovor od web (HTTP) poslužitelja. U odgovoru se potvrđuje primitak paketa broj **4** (s porukom `ACK`) koji je bio veličine 794 *bajta*. Stoga odgovaramo s `ACK 795` jer smo primili sve do 794 *bajta* i očekujemo sljedeći niz (N+1). Dalje u paketu vidimo kako je `SEQ 1` jer se u tom smjeru nije ništa od podataka niti slalo prema klijentskoj strani. Kada govorimo ništa od podataka, mislimo na *payload* odnosno ***data*** dio paketa, koji je odgovoran za više protokole, iznad razine **TCP** ili **UDP** transportnih protokola, a u ovom slučaju za aplikacijski protokol HTTP tj. njegov sadržaj.

**6.** Paket (**6**) je paket u kojem *server* šalje klijentu nešto preko HTTP protokola, pa *payload* sadrži i neke podatke (*HTTP* podatke) i to veličine 1260 *bajta*. Ovo je prvi odgovor s podacima višeg protokola (*http*) s podacima veličine 1260 *bajta*.
Stoga *server* povećava `SEQ` na **1261**, a ostavljamo `ACK` na **795** jer je ova vrijednost nepromijenjena u odnosu na paket broj **5**.

**7.** Paket (**7**) je paket u kojem klijent odgovara *server*u da je primio paket broj **6**. Dakle odgovara samo s `TCP ACK` paketom u kojemu je `ACK` vrijednost **1261** jer potvrđujemo kako smo primili podatke sve do veličine **1260** (+ 1 znači kako očekujemo sljedeći niz podataka). `SEQ` broj je ostao na **795** jer u tom smjeru nije bilo promjena u prijenosu podataka.

Ovakvu analizu možete napraviti i sami upotrebom programa **Wireshark,** a možemo koristiti i program `tcpdump`
Prvo je potrebno pronaći prvu `TCP SYN` poruku, označiti ju s mišem, desni klik te odabrati ***Follow*** te ***TCP Stream***.
Sad će biti filtriran samo ovaj *TCP* komunikacijski kanal.
Potom možemo otići u meni ***Statistics*** i odabrati ***Flow Graph*** kako bi dobili graf s komunikacijom unutar ovog komunikacijskog kanala.

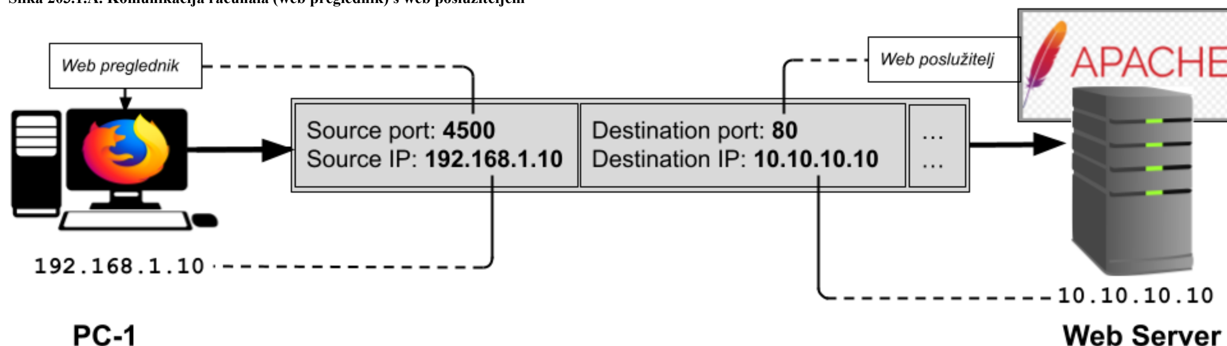


Vezano za naredbu `tcpdump` pogledajte poglavlje:
**25.7.8. Naredba *tcpdump*.**

## Kako aplikacije znaju da se određena komunikacija odnosi na njih?

Za svaki pojedini *TCP* komunikacijski kanal u svakom paketu u razmjeni podataka, unutar tog komunikacijskog kanala, vidljivi su izvorišni port (engl. *source*) i odredišni port (engl. *destination*). Ako klijent traži *TCP* vezu na odredišni port 80 (DST Port:80) poslužitelja, to znači kako se radi o HTTP protokolu. Klijentskom računalu je njegov izvorišni port vezan s aplikacijom koja je otvorila komunikaciju prema odredištu (web poslužitelju). To znači da je operativni sustav prilikom pokretanja aplikacije (pr. web preglednika) povezao aplikaciju s novo kreiranim izvorišnim portom koji će se koristiti u mrežnoj komunikaciji aplikacije s vanjskim svijetom; u konkretnom slučaju s udaljenim web poslužiteljem.
Ovaj izvorišni port se naziva ***efemernim*** (privremenim) portom.
U povratnom paketu, kada računalo (**PC-1**) zaprimi mrežni paket i kada vidi taj port (u dolaznom paketu je to sada odredišni port) i iz njega izvadi aplikacijske podatke, ono će odmah znati kojoj lokalnoj aplikaciji su ti podaci namijenjeni.

**Slika 203.1.A. Komunikacija računala (web preglednik) s web poslužiteljem**




Dok je odredišnom računalu (poslužitelju) njegov izvorišni port (80 u ovom slučaju) vezan za poslužiteljsku aplikaciju (sâm web poslužitelj konkretno) koji ga je otvorio. Kako bi ova veza uopće mogla biti ostvarena, poslužitelj (*server*) prvo mora imati otvoren port 80 (TCP u ovom slučaju) te na njemu pokrenut funkcionalan web poslužitelj, koji je u stanju odgovarati na zahtjeve prema HTTP protokolu koji radi na portu 80. Dakle prvo se šalje `TCP SYN` poruka za uspostavljanje nove veze. Međutim, ako nakon slanja `TCP SYN` poruke za uspostavljanje nove veze nije primljen odgovor (`SYN ACK`), poruka `TCP SYN` se šalje još nekoliko puta.

Ovaj broj ponavljanja ovisi o operativnom sustavu, a može se i mijenjati. Na Linuxu je ona definirana u varijabli `tcp_syn_retries` koju možemo i mijenjati. Standardno je definirano da se ova poruka ponavlja odnosno šalje šest (6) puta prije nego se krene u proces zatvaranja veze jer se ona, ako na nju nije odgovoreno niti nakon šestog pokušaja, nakon toga proglašava kandidatom veze koju će se zatvoriti.

| TCP opcija | Lokacija varijable | Sysctl naziv varijable | Standardna vrijednost |
|---|---|---|---|
| `tcp_syn_retries` | `/proc/sys/net/ipv4/tcp_syn_retries` | `net.ipv4.tcp_syn_retries` | `6` |

**Pogledajmo kako provjeriti otvorene TCP portove i sve ostvarene TCP veze.**
Prvo pogledajmo mrežne statistike s naredbom `netstat`:

```
netstat -tunap
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address      Foreign Address    State       PID/Program name
tcp        0      0 0.0.0.0:22         0.0.0.0:*          LISTEN      841/sshd: /usr/sbin
tcp        0    288 192.168.1.222:22   192.168.1.112:4567 ESTABLISHED 1183/sshd: root@pts
```

← U ispisu vidimo otvorene poslužiteljsku mrežnu aplikaciju u stanju `LISTEN`. Konkretno poslužiteljski servis SSH je otvorio TCP port 22 (`22`), pri tome je ime SSH procesa `sshd`. Vidimo i ostvarane konekcije kao `ESTABLISHED`.

A potom sve pogledajmo s novijom naredbom `ss`:

```
ss -atn
State     Recv-Q       Send-Q     Local Address:Port    Peer Address:Port
LISTEN     0            128       0.0.0.0:22            0.0.0.0:*
ESTAB      0             64       192.168.1.222:22      192.168.1.112:4567
```

Ako promatramo portove i ostvarene veze, vidimo da je SSH servis na lokalnom portu 22 odnosno vidljivo kao: `192.168.1.222:22`. To znači da je on na IP adresi `192.168.1.222` te da sluša na portu `:22`. Također vidimo da se na ovaj SSH servis spojilo udaljeno računalo s IP adresom: `192.168.1.112` i svojim lokalnim portom: `4567`. Lokalni port `4567` udaljenog računala jest izvorišni port, ako gledamo sa strane klijenta i to je tzv. **efemerni** port dok je njemu odredišni port `22` koji je prema standardu port za SSH servis.
SSH klijent komunicira s poslužiteljem: ***Source: [IP: 192.168.1.112, port 4567] → Destination: [IP: 192.168.1.222, port 22]***
Poslužitelj s klijentom koomunicira: ***Source: [IP: 192.168.1.222, port 22] → Destination: [IP: 192.168.1.112, port 4567]***

**Izvori informacija:** `man nstat`, `man netstat`, `man ss`, **RFC 793**

## 24.2.3. Sigurnosni aspekt uspostavljanja TCP veze (*SYN*)

Vrlo popularna vrsta napada uskraćivanja pristupa odnosno ***DOS*** (*Denial of Service*) uključuje napadača koji šalje mnoge (eventualno krivotvorene) *TCP SYN* pakete na vaš poslužitelj, ali nikada ne dovršava TCP trostruko uspostavljanje veze.
Ova vrsta napada zove se *TCP SYN flood*, odnosno preplavljivanje poslužitelja s *TCP SYN* porukama za uspostavljanje TCP veze. Ovakav napad vrlo brzo popunjava memoriju rezerviranu za veze u stanju otvaranja, sprječavajući ostvarivanje legitimnih veza.

Budući da veza ne mora biti dovršena, na strani napadača se ne zauzimaju resursi kao na strani poslužitelja, stoga je jednostavno izvršiti i održavati ovakve napade na poslužiteljsku stranu. Napadač šalje veliki broj `SYN` poruka, s namjerom da nikada ne odgovori na *SYN ACK* s *ACK* porukom s kojom bi se uspostavila TCP veze. Dakle poslužitelj se ostavlja u stanju u kojem on za svaki *SYN* šalje *SYN ACK*, ali pošto mu napadač nikada ne odgovori s *ACK* poslužitelj mora čekati na istek ove polu otvorene veze, što obično traje oko 60 sekundi. Pri velikom broju ovakvih poruka (napada), poslužitelju se prvo prepuni ***SYN queue*** memorija, ali mu se troše i CPU resursi, kao i slobodni portovi te u konačnici i *file deskriptori*. Ne zaboravimo kako je za jednu konekciju potreban barem jedan *file deskriptor*.
Kernel memorija zadužena za TCP konekcije u stanju otvaranja se zove ***SYN queue* ili *incomplete connection queue***.
Kao zaštitu, moguće je ograničiti (ili povećati) broj dolaznih konekcija na sustav odnosno definirati koliki broj istovremenih konekcija u stanju otvaranja; nakon poslanog prvog *TCP SYN* te našeg slanja *TCP SYN ACK*, a prije zadnjeg *TCP ACK* od *klijenta,* dozvoljavamo na sustav. Ova opcija se definira u `net.core.somaxconn`. Njena standardna vrijednost je obično 128.
Dakle ovdje se praktično radi o konekcijama u poluotvorenom stanju, prije nego konekcija pređe u stanje *ESTABLISHED*.
Drugi zaštitni mehanizam je tzv. *Syncookies*. Ako je postavljena varijabla `tcp_syncookies` (dostupna sâmo, ako je vaš kernel *kompiliran* s `CONFIG_SYNCOOKIES`) kernel obrađuje *TCP SYN* pakete normalno sve dok mu se navedena memorija rezervirana za konekcije u stanju otvaranja (***SYN queue***) ne popuni, pri čemu se funkcija `tcp_syncookies` aktivira. Omogućavanje *SYN cookies* metode je vrlo jednostavan način za obranu od napada pomoću *SYN* poplava/napada, pri čemu se koristi sâmo malo više CPU vremena. *SYN cookies* metoda ne radi prilagođavajući *SYN* red čekanja (***SYN queue***).
Umjesto toga kernel će odgovoriti na bilo koji *SYN* paket s legitimnom *SYN ACK* porukom kao i normalno, ali će predstaviti posebno oblikovan TCP sekvencijski broj (*SEQ*) koji kodira izvorišnu, odredišnu IP adresu te broj porta i vrijeme kada je paket poslan. Napadač koji obavlja ovaj napad nikada ne bi ni dobio ovaj paket (*SYN ACK*), ako koristi neku od metoda *spoofing*a odnosno kreiranja lažnih paketa, pa ne bi niti pokušao odgovoriti.
Pri tome se poslužitelj opterećuje i raste mu broj TCP veza koje su u stanju *SYN RECEIVED* sve dok im ne istekne vremenski okvir (*timer)* nakon kojeg bi one bile zatvorene.
S druge strane legitimni pokušaj spajanja klijenta, poslao bi treći paket za uspostavu *TCP* veze koji uključuje *SEQ* broj, a poslužitelj može potvrditi kako je ovo odgovor na valjani *SYN* paket te omogućiti vezu, iako nema odgovarajućeg unosa u *SYN redu* čekanja. Na Linuxu je metoda *SYN cookie* definirana u varijabli `tcp_syncookies` i standardno je uključena:

| Lokacija varijable | Sysctl naziv varijable | Standardna vrijednost |
|---|---|---|
| `/proc/sys/net/ipv4/tcp_syncookies` | `net.ipv4.tcp_syncookies` | `1` |

Vratimo se na ***SYN queue*** memoriju, koja je podijeljena u nekoliko cjelina i dijelova. Jedna cjelina je zadužena za jedan TCP port, primjerice port 80 (HTTP), a unutar te cjeline ima mjesta za određeni broj konekcija u stanju otvaranja o kojima smo govorili. U Linuxu maksimalan broj ovih cjelina je definiran u varijabli: `tcp_max_syn_backlog` čija vrijednost je standardno 128, što znači kako je za svaki pojedini otvoreni port moguće spremiti istovremeno 128 konekcija u stanju otvaranja.
Konekcije koje su poslije otvorene i u statusu su *ESTABLISHED* (ostvarene) više se ne čuvaju u ovoj ***TCP backlog*** memoriji. Dakle ovdje se radi o međumemoriji u koju se spremaju polu otvorene TCP konekcije, odnosno one koje još nisu dobile povratnu *TCP ACK* poruku. Ova vrijednost se može poprilično povećati za poslužitelje na koje se stalno ostvaruje veliki broj novih konekcija. Pogledajmo ovu varijablu u Linuxu (koju naravno možemo mijenjati):

| Lokacija varijable | Sysctl naziv varijable | Standardna vrijednost |
|---|---|---|
| `/proc/sys/net/ipv4/tcp_max_syn_backlogs` | `net.ipv4.tcp_max_syn_backlog` | `128` |



Pogledajte i poglavlje:
**25.1. Linux mrežni model.**

U određenim (vrlo rijetkim) slučajevima mrežni servis (*daemon*) može previše sporo prihvaćati nove konekcije.
U normalnom radu nikakav mehanizam neće spriječiti ovakvo ponašanje servisa jer je moguće da je servis trenutno preopterećen te trenutno ne može prihvaćati nove konekcije ili ih ne može ostvariti unutar vremenskih okvira unutar kojih bi se nova konekcija morala ostvariti. Drugi razlog može biti da se ***backlog queue*** prepunio, opet zbog više razloga, te je rezultat ponovno nemogućnost ostvarivanja novih konekcija, dok se *backlog* memorija ne isprazni. Ipak i ovdje postoji mogućnost optimizacije, definirana u sistemskoj varijabli. Ova varijabla je standardno u vrijednosti nula (`0`) dakle isključena je.
Naime varijabla: `net.ipv4.tcp_abort_on_overflow` govori kernelu da poništi nove veze, ako je sustav trenutačno prepunjen novim pokušajima povezivanja koje mrežni servis (*demon*) ne može podnijeti. Ono što to znači jest da, ako se na sustav pokušava spojiti 1.000 klijenata odjednom, nove veze ili one „viška“ (koje sustav nije u stanju prihvatiti) se mogu resetirati jer ih naš servis (*daemon*) ne može obraditi, u slučaju kada je ova varijabla uključena. Ako nije uključena, sustav odnosno servis će se pokušati osvježiti i pokušati obrađivati sve pristigle nove zahtjeve za spajanjem.

Sljedeća *sysctl* varijabla odnosno značajka koja je korisna je `net.ipv4.tcp_max_orphans`. Naime, ovdje se definira maksimalan broj mrežnih utičnica (*socketa*), koje trenutno nisu spojene s nijednim *file deskriptorom*. To se odnosi na mrežne *sockete* koji su u stanju zatvaranja, ali još nisu zatvoreni (u TIME_WAIT stanju su). Kada je ovaj broj premašen, prekinuta veza se resetira i ispisuje se upozorenje. Ovo ograničenje postoji za sprječavanje napada uskraćivanjem usluge (***DoS*** napad). U slučaju kada se dođe do ovog ograničenja, sustav će izbaciti poruku: `TCP: too many of orphaned sockets`.

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju. Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom. Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj:

| Naziv brojača | Opis |
|---|---|
| **TcpInSegs** | Broj paketa koje je primio TCP sloj. Uključuje pakete primljene greškom, kao što je pogreška kontrolne provjernog zbroja (*checksum* error), nevažeće TCP zaglavlje i slično. Samo jedna pogreška neće biti uključena: ako odredišna adresa sloja dva nije adresa mrežne kartice na sloju dva (MAC adresa). To se može dogoditi ako je paket *multicast* ili *broadcast* paket ili je mrežna kartica u promiskuitetnom načinu rada. U tim situacijama, paketi bi bili isporučeni TCP sloju, ali TCP sloj će odbaciti te pakete prije povećanja `TcpInSegs` brojača. `TcpInSegs` brojač nije svjestan **GRO**-a. Dakle, ako **GRO** spoji dva paketa, brojač `TcpInSegs` povećao bi se samo za 1. (**RFC 1213**) |
| **TcpOutSegs** | Broj paketa koje je poslao TCP sloj. Isključuje brojanje ponovno poslanih paketa. Ali uključuje pakete **SYN**, **ACK** i **RST**. `TcpOutSegs` je svjestan **GSO**-a, pa ako bi **GSO** podijelio paket na 2, `TcpOutSegs` će se povećati za 2. (**RFC 1213**) |
| **TcpActiveOpens** | Ovo stanje znači da TCP sloj šalje **SYN** i dolazi u stanje **SYN-SENT**. Svaki put kada se `TcpActiveOpens` poveća za 1, `TcpOutSegs` uvijek se treba povećati za 1. (**RFC 1213**) |
| **TcpPassiveOpens** | Ovo stanje znači da TCP sloj prima **SYN**, te odgovara sa **SYN+ACK**, te dolazi u stanje **SYN-RCVD**. (**RFC 1213**) |
| **TcpExtTCPRcvCoalesce** | Kada TCP sloj primi pakete, a aplikacija ih ne pročita, TCP sloj će ih pokušati spojiti. Ovaj brojač pokazuje koliko je paketa spojeno u takvoj situaciji. Ako je **GRO** omogućen, **GRO** bi spojio mnogo paketa, a ti se paketi ne bi računali u `TcpExtTCPRcvCoalesce`. |
| **TcpExtTCPAutoCorking** | Prilikom slanja paketa, TCP sloj će pokušati spojiti male pakete u jedan veći. Ovaj se brojač povećava za 1 za svaki spojeni paket u takvoj situaciji. |
| **TcpExtTCPOrigDataSent** | Broj odlaznih paketa s izvornim podacima (isključujući ponovni prijenos (*retransmisiju*) ali uključujući podatke u SYN). Ovaj se brojač razlikuje od `TcpOutSegs` jer `TcpOutSegs` također prati čiste ACK-ove. `TCPOrigDataSent` je korisniji za praćenje stope ponovnog slanja (*retransmisije*) TCP-a. |
| **TcpExtTCPBacklogDrop** | Ovaj brojač prati broj TCP veza koje su odbačene zbog prekoračenja ograničenja (*backlog međumemorije*) reda čekanja na uspostavu veze. Kada je broj zahtjeva za uspostavu veze veći od dostupnih resursa za obradu, sustav može odbaciti nove zahtjeve, a ovaj brojač bilježi te slučajeve. Povećanje ***sysctl*** varijabli: `net.core.somaxconn`, `net.core.netdev_max_backlog`, `net.ipv4.tcp_max_syn_backlog` i `net.ipv4.tcp_syncookies` može pomoći. |
| **TcpExtPFMemallocDrop** | Ovaj brojač prati broj TCP paketa koji su odbačeni zbog problema s alociranjem memorije na sustavu. To može značiti da sustav nije mogao dodijeliti dovoljno memorije za obradu dolaznih TCP paketa, što može biti znak preopterećenja ili problema s resursima. Povećanje ***sysctl*** varijabli: `net.core.rmem_max`, `net.core.wmem_max`, `net.ipv4.tcp_mem`, `net.ipv4.tcp_rmem` i `net.ipv4.tcp_wmem` ovdje može pomoći. |
| **TcpExtTCPMinTTLDrop** | Ovaj brojač prati broj TCP paketa koji su odbačeni zbog dolaska s minimalnim vremenom života (TTL) manjim od minimalnog dopuštenog TTL-a. TTL se koristi u IP paketima kako bi se spriječilo beskonačno rutiranje paketa u mreži. Kada paket doseže svoj **IP TTL**, on se odbacuje, a ovaj brojač bilježi takve slučajeve. |
| **TCPSynRetrans** | Broj **SYN** i **SYN/ACK** ponovnih slanja (*retransmisija*) za razlaganje ponovnog prijenosa u: **SYN, Fast retransmit**, **Timeout retransmit**, itd. |

Postavke međumemorija pomoću slijedećih *sysctl* varijabli mogu utjecati na probleme s odbacivanjem paketa vidljivih u gornjim statistikama: `net.core.rmem_max`, `net.core.rmem_default`, `net.ipv4.tcp_mem`, `net.ipv4.tcp_rmem`, `net.ipv4.tcp_wmem`, `net.core.somaxconn`, `net.core.netdev_max_backlog`, `net.ipv4.tcp_max_syn_backlog` i `net.ipv4.tcp_syncookies`.



Po tom pitanju pogledajte poglavlja:

**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

| Naziv brojača | Opis |
|---|---|
| **TcpRetransSegs** | Ukupan broj ponovno poslanih segmenata - to jest, broj poslanih TCP segmenata koji sadrže jedan ili više prethodno poslanih okteta. |
| **TCPFastOpenActiveFail** | Pokušaji brzog otvaranja TCP veze (**Fast open**) (SYN+podaci) nisu uspjeli jer ga druga strana ne prihvaća ili je isteklo vrijeme. |
| **TcpExtListenOverflows** **TcpExtListenDrop** | Kada kernel primi **SYN** od klijenta i ako je red čekanja za TCP pun, kernel će ispustiti SYN paket i dodati 1 u `TcpExtListenOverflows`. U isto vrijeme kernel će također dodati 1 u `TcpExtListenDrops`. Kada je TCP socket u stanju **LISTEN**, a kernel treba ispustiti paket, kernel bi uvijek dodao 1 u `TcpExtListenDrops`. Dakle, povećanje `TcpExtListenOverflows` omogućilo bi povećanje `TcpExtListenDrops` u isto vrijeme, ali bi se `TcpExtListenDrops` također povećalo bez povećanja `TcpExtListenOverflows`, npr. neuspješna dodjela memorije također bi omogućila povećanje `TcpExtListenDrops`. Dakle na `TcpExtListenDrops` utječe nedostatak resursa (*međumemorija*), koji se može poboljšati sa *sysctl* varijablama: `net.core.somaxconn`, `net.core.netdev_max_backlog`, `net.ipv4.tcp_max_syn_backlog` |
| **TcpInCsumErrors** | Povećava se kada primljeni TCP paket ima netočan kontrolni zbroj (*checksum*). |
| **TcpInErrs** | Ukupan broj segmenata primljenih pogreškom (npr. loš TCP checksum). |
| **TcpExtPruneCalled** | Povećava se pri pokušaju smanjenja memorije dodijeljene socketu [tcp_prune_queue()1] |
| **TcpExtRcvPruned** | Povećava se kada probno smanjenje dodijeljene memorije socketu nije uspjelo, podaci se ispuštaju [tcp_prune_queue()2] |
| **TcpExtOfoPruned** | Povećano pri čišćenju reda čekanja izvan reda strukture (**struct tcp_sock**) [tcp_prune_ofo_queue()] |
| **TcpExtTCPAbortOnData** | Ova statistika prati broj TCP veza koje su prekinute zbog dolaska neočekivanih podataka na vezi. To može značiti da su primljeni podaci koji ne odgovaraju očekivanom obrascu ili protokolu, što može biti znak problema s komunikacijom ili neispravno konfiguriranim aplikacijama. |
| **TcpExtTCPAbortOnClose** | Ova statistika prati broj TCP veza koje su prekinute kada je veza zatvorena (*closed*). To se može dogoditi kada jedna strana zatvori vezu prije nego što je druga strana završila s komunikacijom. |
| **TcpExtTCPAbortOnMemory** | Ova statistika prati broj TCP veza koje su prekinute zbog problema s memorijom. To se može dogoditi kada sustav nema dovoljno memorije za održavanje aktivne TCP veze ili kada je memorija postala neispravna. |
| **TcpExtTCPAbortOnTimeout** | Ova statistika prati broj TCP veza koje su prekinute zbog isteka vremenskog ograničenja. |
| **TcpExtTCPAbortOnLinger** | Ova statistika prati broj TCP veza koje su prekinute kada opcija "*linger*" nije bila uspješna. Linger opcija omogućuje procesu da čeka određeno vrijeme nakon što zatvori vezu prije nego što se veza zapravo zatvori. Ako ta operacija ne uspije, veza se može prekinuti. |
| **TcpExtTCPAbortFailed** | Ova statistika prati broj neuspješnih pokušaja prekida (*aborta*) TCP veza. To može biti zbog različitih razloga, uključujući poteškoće u komunikaciji s drugom stranom ili problema u sustavu. |
| **TcpExtTCPMemoryPressures** | Ova statistika prati pritisak (opterećenje) na memoriju uzrokovan TCP operacijama. To može uključivati potrošnju memorije za održavanje TCP veza, prilagodbu parametara veze radi uštede memorije ili upozorenja kada memorija postane kritično ograničena. |
| **TcpExtTCPMemoryPressuresChrono** | Ova statistika prati kronološki pritisak na memoriju uzrokovan TCP operacijama. To pruža dodatni kontekst o tome kako se pritisak na memoriju mijenjao tijekom vremena, što može pomoći u dijagnosticiranju problema ili optimizaciji sustava. |

**SYN cookie**

| Naziv brojača | Opis |
|---|---|
| **TcpExtSyncookiesSent** | Označava koliko je *SYN cookies* poslano. |
| **TcpExtSyncookiesRecv** | Koliko paketa odgovora na SYN cookies prima TCP stôg. |
| **TcpExtSyncookiesFailed** | MSS vrijednost dekodirana iz SYN cookies je nevažeća. Kada je ovaj brojač Ažuriran (povećan), primljeni paket neće se tretirati kao SYN cookie i `TcpExtSyncookiesRecv` brojač neće se ažurirati. |

**Izvori informacija: (340),(346),(347),(642),(1406),(1477),(1650),(1651), `man 7 tcp`,`man 7 ip`,`man nstat`, RFC 1213 .**

## 24.2.4. Trajanje TCP veze

Svaki uspostavljeni TCP komunikacijski kanal između dva sugovornika; pošiljatelj ←→ primatelj, stabilan je i otvoren sve dok:

- Kroz njega teče neka komunikacija ili se kroz njega šalju *TCP keepalive* paketi, koji održavaju vezu živom, čak i kada kroz nju ne teče promet. Ako su *keepalive* intervali dovoljno mali i ako se ovaj mehanizam koristi.



Pogledajte poglavlje:
**24.2.15 TCP keepalives.**

- Druga strana (primatelj) uredno potvrđuje primitak poslanih podataka (paketa) unutar postavljenih vremenskih ***RTO*** okvira (za normalne pakete i za *keepalive* pakete).



Pogledajte poglavlje:
**24.2.7.1 Vrijeme prolaska paketa (*Round-Trip Time*).**

Dodatno u operativnom sustavu Linux moguće je optimizirati postavke koje utječu na slučajeve u kojima dolazi do gubitka podataka, a koji mogu utjecati na automatsko zatvaranje TCP veze.
Za to su zadužene dvije varijable:

- `tcp_retries1` - postavljena je na vrijednost tri (`3`), dakle ako potvrda (*ACK*) ne dođe do pošiljatelja u okviru ***RTO*** vremenskog okvira, ponavlja se njeno slanje, do maksimalno (`3`) tri puta, kako je ovdje definirano.

- `tcp_retries2` - standardno postavljena vrijednost `15` označava maksimalan broj puta koliko TCP paket može biti ponovno poslan (***retransmisija***). Ovo je važno i stoga jer se ova vrijednost koristi i u izračunu za maksimalno vrijeme trajanja neaktivne konekcije. U maksimalno vrijeme trajanja neaktivne konekcije (kroz koju ne teče promet) koje se dobiva iz ove vrijednosti se nadodaje vrijeme prolaska paketa ***RTT*** **(***Round Trip Time*) pa se izračunava ***RTO*** ***(Retransmission Time Out)*** *vrijeme***.** **U** konačnici se ne preporučuje manja vrijednost od 100 sekundi (prema **RFC 1122**) što odgovara vrijednosti `9`. Vrijednost `15` odgovara trajanju između 13 i 30 minuta. Točno vrijeme ovisi o ***RTO***, koje se dinamički i automatski izračunava prema **RFC 6298**. Dakle ova vrijednost je balans između brzog detektiranja mrtvih konekcija i dužeg držanja onih koje su žive, ali kroz njih trenutno ne teče promet. Ako se ipak šalju paketi za održavanje veze (*TCP keepalive*) koji ju održavaju na životu, ona može beskonačno biti otvorena, osim ako ju bilo koja od strana u međuvremenu ne zatvori ili ako se ne dobije odgovor na određeni broj TCP *keepalive* poruka definiran u varijabli `net.ipv4.tcp_keepalive_probes`. *TCP keepalive* vrijednosti moraju biti ispod ovih granica. Za detalje pogledajte sljedeću stranicu.

Pogledajmo lokacije i standardne vrijednosti ovih varijabli, koje naravno možemo i mijenjati:

| Varijabla | Lokacija varijable | sysctl naziv varijable | Standardna vrijednost varijable |
|---|---|---|---|
| `tcp_retries1` | `/proc/sys/net/ipv4/tcp_retries1` | `net.ipv4.tcp_retries1` | **3** (puta) |
| `tcp_retries2` | `/proc/sys/net/ipv4/tcp_retries2` | `net.ipv4.tcp_retries2` | **15** (vrijednost) |

Pogledajmo ove vrijednosti pomoću naredbe `sysctl`:

```
sysctl -a | grep tcp_retries
net.ipv4.tcp_retries1 = 3
net.ipv4.tcp_retries2 = 15
```



Važno je razumjeti da promjene ovih vrijednosti utječu na sve buduće **TCP** konekcije na sustavu!



Za detalje kako mijenjati ***sysctl*** varijable, pogledajte poglavlje:
**4.5.8. Naredba *sysctl*.**

Vratimo se na maksimalan broj TCP retransmisija definiranih u ***sysctl*** varijabli `net.ipv4.tcp_retries2`. Kako smo već rekli ova vrijednost je u korelaciji s **RTO** vremenom, te `TCP_RTO_MAX` i `TCP_RTO_MIN`, vidljivo prema tablici:

| **Retransmisije** ***(vrij. varijable)*** | **RTO [ms]** | **Vrijeme isteka (*Timeout*) [sekunde]** | | **Retransmisije** ***(vrij. varijable)*** | **RTO [ms]** | **Vrijeme isteka (*Timeout*) [sekunde]** |
|---|---|---|---|---|---|---|
| 1 | 200 | 0,2 | | 9 | 51200 | 102,2 |
| 2 | 400 | 0,6 | | 10 | 102400 | 204,6 |
| 3 | 800 | 1,4 | | 11 | 120000 | 324,6 |
| 4 | 1600 | 3 | | 12 | 120000 | 444,6 |
| 5 | 3200 | 6,2 | | 13 | 120000 | 564,6 |
| 6 | 6400 | 12,6 | | 14 | 120000 | 684,6 |
| 7 | 12800 | 25,4 | | 15 | 120000 | 804,6 |
| 8 | 25600 | 51 | | 16 | 120000 | 924,6 |

Za praćenje stanja ***TCP*** veza, možete koristiti naredbu `netstat` u kojoj je posebno važan prekidač `-o` (za stanja veze):

```
netstat -tunapo
```



Pogledajte i poglavlja:
**24.2.13 Mehanizam TCP retransmisije.**
**24.2.15. TCP keepalives** → obratite pažnju na primjer ispisa naredbe **`netstat -anto`**.
**25.7.4. *Naredba netstat.***

**Slijedi napredni dio!**
Implementacija TCP protokola pod Linuxom dopušta aplikacijama postavljanje svojih vremenskih ograničenja na TCP sesiju (tzv. *TCP timeout*). Ova opcija mora biti uključena unutar programskog kôda konkretnog programa, a naziv ove opcije je `TCP_USER_TIMEOUT`. Navedena opcija (ako je postavljena u aplikaciji) nadilazi sysctl postavku `tcp_retries2` [**RFC1122**]. Ako se ova opcija ne koristi, sustav za aplikacije koristi standardne postavke. Ako se ipak koristi, u njoj se specificira maksimalna količina vremena u milisekundama tijekom koje poslani podaci mogu ostati nepotvrđeni (*TCP ACK*) ili podaci u međuspremniku mogu ostati nepreneseni (primjerice zbog nulte veličine TCP prozora), prije nego što će TCP prisilno zatvoriti vezu i poslati ***ETIMEDOUT*** poruku aplikaciji.
U različitim aplikacijama koje koriste ovu opciju, moguće ju je i definirati; kao što je to slučaj primjerice za sljedeće servise:

***SSH servis (SSHD) i klijent***
Ovdje imamo nekoliko opcija s kojima možemo povećati (ili smanjiti) trajanje neaktivnih konekcija

- Poslužiteljska strana - konfiguracijska datoteka (`/etc/ssh/sshd_config`):
  - `ClientAliveInterval` **`XY`** – ovdje se definira vrijeme neaktivne veze, nakon koje će se prekinuti TCP veza (pr. `25m` – 25 minuta) prema klijentu.
  - `ClientAliveCountMax` **`XY`** – ovdje se definira broj poruka (*keepalive*) koje se mogu poslati drugoj strani, bez povratnih poruka od strane klijenta (pr. `10` – 10 puta). Ako je postavljeno na `0`, neće se slati *keepalive* poruke.
- Klijentska strana - konfiguracijska datoteka (`/etc/ssh/ssh_config`):
  - `ServerAliveInterval` **`XY`** – ovdje se definira vrijeme neaktivne veze, nakon koje će se prekinuti TCP veza (pr. `25m` – 25 minuta) prema poslužitelju.
  - `ServerAliveCountMax` **`XY`** – ovdje se definira broj poruka (*keepalive*) koje se mogu poslati drugoj strani, bez povratnih poruka od strane poslužitelja (pr. `10` – 10 puta). Ako je postavljeno na `0`, neće se slati *keepalive* poruke.

***VSFTPD (FTP poslužitelj)***
I ovaj **FTP** poslužitelj koristi mogućnost postavljanja vremenskog ograničenja na TCP sesije (datoteka: `/etc/vsftpd/vsftpd.conf`). Varijabla koju je potrebno postaviti je:

- `idle_session_timeout` **`XY`** – ovdje se definira vrijeme neaktivne veze, nakon koje će se prekinuti TCP veza (standardno je postavljeno `300` – 5 minuta) prema klijentu.

Nadalje, postoji i opcija `SO_KEEPALIVE` koja se odnosi na tzv. ***keepalive*** značajku, a koju aplikacija također može koristiti, samo ako je tako napisana/programirana.

**Kako provjeriti koristi li određena mrežna aplikacija *TCP keepalive* mogućnost sustava?**
Pogledajmo nekoliko mrežnih servisa i njihove *brojače* (`-o`) u kojima ćemo vidjeti i status upotrebe *TCP keepalive* mogućnosti (filtrirano je samo nekoliko servisa):

```
netstat -anto | grep ESTABLISHED

Proto Recv-Q Send-Q Local Address   Foreign Address    State       Timer
tcp        0    112  10.10.10.1:22   10.10.10.100:2551 ESTABLISHED on (0.10/0/0)
tcp        0     96  10.10.10.1:80   10.10.10.100:7551 ESTABLISHED keepalive (84.28/0/0)
tcp        0      1  10.10.10.1:8888 10.10.10.100:7551 ESTABLISHED off (0.00/0/0)
```

Ovdje vidimo tri mrežna servisa: ***SSH***-port `22`; ***HTTP***-port `80` te posebni mrežni servis na portu `8888`, na koje imamo otvorene TCP veze. Za *SSH* servis (port `:22`) u stupcu **`Timer`** (koji ćemo sada promatrati) vidimo stanje `on` što znači kako ovaj servis u ovom trenutku čeka poruku o potvrdi paketa (`TCP ACK`), što nužno (o ovom djeliću sekunde) ne znači kako je *TCP keepalive* onemogućen, već da se trenutno čeka na TCP potvrdni paket, nakon kojega se može i preći u *keepalive* stanje.

Za ***HTTP*** servis (port `:80`) vidimo kako je on (servis) definitivno pokrenut tako da je otvorio *TCP socket*, tako da podržava *TCP keepalive* mogućnost jer se vidi ključna riječ: `keepalive` što znači i kako je *keepalive timer* aktivan.
Pošto je trenutno *keepalive timer* aktivan, uz njega su vidljivi brojevi koji znače sljedeće: prvi broj (`84.28`) je ***keepalive*** brojač, koji se stalno smanjuje (od inicijalnih ***75*** sekundi [`tcp_keepalive_intvl`]) te kada dođe na nulu (***0***), ponovno se provjerava je li bilo ikakvog prometa na toj konekciji, a ako nije, šalje se TCP *keepalive (samo ako je aplikacija otvorila socket s opcijom **SO_KEEPALIVE**).*

Drugi broj označava broj retransmisija (`0` u ovom slučaju), a treći broj *keepalive* poruka (*probes*) koje su poslane.
Zadnji servis (naša posebna mrežna aplikacija `:8888`) nije u mogućnosti koristiti *TCP keepalive*, pa ima stanje: `off`.

Globalne postavke sustava vezano za ***keepalive*** značajku definirane su u tri ***sysctl*** sistemske varijable:

| **sysctl naziv** varijable | **Standardna vrijednost varijable** | **Opis** |
|---|---|---|
| `net.ipv4.tcp_keepalive_time` | **7200 sekundi** | Maksimalno vrijeme koliko TCP veza može biti otvorena prije slanja keepalive poruka. |
| `net.ipv4.tcp_keepalive_intvl` | **75 sekundi** | Svakih koliko sekundi se šalju (ako je veza neaktivna). |
| `net.ipv4.tcp_keepalive_probes` | **9 komada** | Maksimalno koliko puta se šalju *keepalive* poruke. |



*Iako je **TCP keepalive** konfiguriran na sustavu, stvar je aplikacije da koristi (ili ne) ove postavke. Naime, Linux sâm od sebe neće slati **keepalive** pakete, ako to konkretna aplikacija ne podržava.*

*Ako se aplikacija pokreće s opcijom **SO_KEEPALIVE** odnosno ako je s njom otvorila mrežni socket, samo i isključivo tada će se i koristiti **TCP keepalive** mehanizam na razini aplikacije, što je potom vidljivo i iz Linuxa.*



U nedostatku vremenskog ograničenja određenog aplikacijom, TCP specifikacija [**RFC793**] definira zadano vremensko ograničenje TCP sesije od 5 minuta, kojeg se operativni sustav može, ali i ne mora držati (ovisno o implementaciji).

## TCP protokol i vremenska ograničenja trajanja veze

TCP protokol nema mehanizam unutar protokola za signaliziranje mijenjanja vremenskih ograničenja trajanja veze, prema drugoj strani komunikacijske veze. To uzrokuje da su lokalne promjene neučinkovite kod promjena vremenskih ograničenja na trajanje TCP sesije jer će udaljena strana u komunikaciji zatvoriti vezu nakon isteka svog postavljenog vremena (svog *TCP timeout* vremena).

Nadalje [**RFC1122**] proširuje ovu definiciju uvodeći dva praga u radu:
***R1*** i ***R2*** (*R2* > *R1*), koji kontroliraju broj pokušaja ponovnog slanja (*retransmisije*) za pojedini mrežni segment. Sugerira se da TCP treba obavijestiti aplikacije kada se dosegne R1 za mrežni segment, te zatvoriti vezu kada se dostigne R2. [RFC1122] također definira preporučene vrijednosti za R1 (3 ponovna prijenosa) i R2 (od 100 sekundi), uz napomenu da R2 za SYN segmente treba biti najmanje 3 minute.

Ove vrijednosti su u Linuxu definirane kao sljedeće **sysctl** varijable:
`net.ipv4.tcp_retries1` → odnosi se na ***R1***
`net.ipv4.tcp_retries2` → odnosi se na ***R2***, ali je iskazan u posebnoj vrijednosti (1-16).



Srećom [**RFC 5482**] dokument uvodi novu TCP opciju - TCP User Timeout Option (**UTO**) koja omogućuje oglašavanje trenutne vrijednosti vremenskog ograničenja tj. TCP timeout vrijednosti. Ova vrijednost pruža informaciju drugom kraju veze da prilagodi svoje postavke prema njoj. Ipak, druga strana u komunikaciji ostaje slobodna zanemariti ovaj savjet o promjeni, ako lokalna pravila to sugeriraju.

Ipak povećanje vremenskih ograničenja na oba kraja TCP veze dopuštaju da veza preživi duža razdoblja bez slanja podataka kroz TCP vezu.

Sada pogledajmo našu SSH konekciju s naredbom `ss` (skratili smo ispis):

```
ss -i | tail
tcp   ESTAB  0      64   192.168.1.129:ssh       192.168.1.112:iwb-whiteboard

cubic wscale:8,7 rto:238 rtt:37.447/4.838 ato:80 mss:1260 pmtu:1500 rcvmss:1168
advmss:1460 cwnd:10 bytes_sent:81085 bytes_acked:81021 bytes_received:22661
segs_out:427 segs_in:610 data_segs_out:372 data_segs_in:335 send 2.69Mbps lastsnd:2
lastrcv:3 lastack:3 pacing_rate 5.38Mbps delivery_rate 41.1Mbps delivered:372
app_limited busy:11154ms unacked:1 rcv_space:14600 rcv_ssthresh:64076 minrtt:0.091
snd_wnd:2105344
```

Vidimo da je za našu trenutnu SSH vezu prema poslužitelju izračunato ***RTO** (Retransmission Time Out)* vrijeme `238`(ms).
Važno je razumjeti da Linux u ovom izračunu vodi računa da se izračunato vrijeme mora nalaziti unutar okvira:
**TCP_RTO_MIN** (200 ms) i **TCP_RTO_MAX** (120 sekundi) koji su zapisani u kernelu (`TCP_RTO_MAX` i `TCP_RTO_MIN`).



Pogledajte poglavlje:
**24.2.15 TCP keepalives.**

Naime, ako nema protoka podataka na vezi klijent - poslužitelj, TCP/IP koristi takozvani mehanizam za održavanje aktivnosti veze (*keepalive*) kako bi potvrdio da je takva neaktivna veza (u stanju mirovanja) i dalje funkcionalna nakon unaprijed definiranog vremenskog razdoblja (`tcp_keepalive_time`). U međuvremenu, ako se radi o aplikaciji koja to podržava (spomenuta ***SO_KEEPALIVE*** opcija) ona udaljenoj strani šalje poruku o održavaju veze (*keepalive*). Ako je udaljena strana još uvijek dostupna i radi, ona će potvrditi ovakvu poruku za održavanje veze i veza će ostati otvorena. Na Linuxu ovaj mehanizam kontroliraju mrežni atributi `tcp_keepalive_time` (zadano je 2 sata [7200 sekundi]**OVISNO O INAČICI KERNELA**), `tcp_keepalive_probes` (zadano je 9 ponavljanja) i `tcp_keepalive_intvl` (zadano je 75 sekundi).
Zadane vrijednosti ovih mrežnih atributa određuju da se neaktivna veza zatvara nakon otprilike 2 sata i 11 minuta ako se ne potvrde probe za održavanje.

Pogledajmo računicu:

$$\mathbf{7200}s + (\mathbf{9}\,x\,\mathbf{75}s) = \mathbf{7850}s = \mathbf{2}.\,sata\ \mathbf{11}.\,minuta$$

## Istek vremena uspostavljanja veze klijenta prema poslužitelju

Kada se klijent pokušava spojiti na poslužitelj, šalje mu TCP SYN poruku te očekuje SYN-ACK potvrdu, koja može biti ili izgubljena u mreži ili se može dogoditi da ju poslužitelj uopće me pošalje. Klijent tada zapisuje poruku o pogrešci i vraća pogrešku procesu povezivanja u Linuxu. Kada se to dogodi, pokušaj povezivanja nije uspio te se pokušava ponovno.

Prilikom slanja SYN poruka, zadnji pokušaj povezivanja ističe nakon vremenskog razdoblja određenog vrijednošću mrežnog atributa `tcp_syn_retries`. Zadana vrijednost za `tcp_syn_retries` je `6`, što odgovara 63 sekunde do zadnjeg ponovnog prijenosa s trenutnim početnim **RTO** vremenom od 1 sekunde. Konačno vrijeme nakon kojeg će se TCP veza zatvoriti (ako i dalje nema odgovora) je nakon 127 sekundi.

Pogledajmo računicu za 6 pokušaja i istek konačnog vremena:

$$\mathbf{1}s\ (+\mathbf{2}) = \mathbf{3}s\ (+\mathbf{4}) = \mathbf{7}s\ (+\mathbf{8}) = \mathbf{15}s\ (+\mathbf{16}) = \mathbf{31}s\ (+\mathbf{32}) = \mathbf{63}s\ (+\mathbf{64}) = \mathbf{127}s$$

Ova varijabla (atribut) je dostupna kao *sysctl* varijabla (`net.ipv4.tcp_syn_retries`):

| **sysctl naziv** varijable | **Standardna vrijednost varijable** |
|---|---|
| `net.ipv4.tcp_syn_retries` | **6** *(ponavljanja)* |

## Istek vremena zatvaranja veze

Postoji i još jedna opcija koja utječe na konačno vrijeme trajanja TCP veze. Za prekid veze uobičajeno se šalje TCP FIN paket (iako postoje i druge metode). Prekid veze obično počinje tako da jedna strana signalizira da želi zatvoriti vezu kako bi se osiguralo da se veza uredno ugasi. U slučaju kada klijent šalje poslužitelju FIN paket on je u stanju ***FIN_WAIT_1***. Poslužitelj dobiva FIN paket i prelazi u stanje ***CLOSE_WAIT***, te šalje paket potvrde natrag na klijenta. Kada klijent dobije taj paket, on prelazi u stanje ***FIN_WAIT_2.*** Iz perspektive klijenta, veza je sada zatvorena, ali mrežne utičnice (*mrežni socket*) ostaju još neko vrijeme u stanju: ***FIN-WAIT-2***, prije nego se mrežni *socket* ne zatvori.
Mrežni atribut u kojem se definira ovo vrijeme čekanja na zatvaranje *socketa* je `tcp_fin_timeout`.
Dulje vrijeme znači veće zauzeće memorije i portova u upotrebi, ali i procesorskog vremena.

Ova varijabla (atribut) je dostupna kao *sysctl* varijabla (`net.ipv4.tcp_fin_timeout`):

| **sysctl naziv** varijable | **Standardna vrijednost varijable** |
|---|---|
| `net.ipv4.tcp_fin_timeout` | **60** *(sekundi)* |



Pogledajte poglavlje:
**24.2.11. Kako se zatvara TCP veza.**

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju.

Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj vezanih za održavanje TCP veze:

| Naziv brojača | Opis |
|---|---|
| **`TcpExtTCPKeepAlive`** | Ova statistika prati broj *TCP Keep-Alive* paketa poslanih od strane sustava. *TCP Keep-Alive* paketi koriste se za održavanje veze aktivnom ako nema prometa. Ovo je korisno kako bi se izbjeglo zatvaranje veze zbog neaktivnosti, što može biti posebno važno u nekim aplikacijama. |
| **`TcpExtTCPTimeouts`** | Ova statistika prati broj TCP veza koje su istekle zbog vremenskog ograničenja. To može biti zbog različitih razloga, uključujući nedostatak odgovora od druge strane ili problema u mreži. Praćenje ovih timeouta može pomoći u dijagnosticiranju problema s mrežnim vezama. |
| **`TcpExtTCPAbortOnTimeout`** | Ova statistika prati broj TCP veza koje su prekinute (*abortirane*) zbog isteka vremenskog ograničenja. Ovo se događa kada sustav odluči prekinuti vezu nakon što istekne određeno vremensko razdoblje bez odgovora od druge strane. |
| **`TcpExtTcpTimeoutRehash`** | Ova statistika prati broj puta kada se dogodilo prestrukturiranje *timeouta* TCP veza. To se može dogoditi kada se dinamički prilagođavaju timeouti kako bi se bolje nosili s promjenjivim uvjetima mreže ili opterećenjem sustava. |

**Izvor informacija: (325),(642),(1398),(1399),(1400),(1401),(1406),(1477),(1650),(1651),(K-6),(K-10),** **`man sysctl`**, **`man netstat`**, **`man nstat`**, **`man ss`**, **`man 7 tcp`**, **RFC 793, RFC 1122, RFC 5482, RFC 6298.**

### 24.2.5. Standardne potvrde (*ACK*)

U standardnom načinu rada, nakon što je primatelj zaprimio paket, isti paket, ako je zaprimljen ispravan, dužan je potvrditi s porukom `ACK`. Međutim i ovdje postoje dvije opcije za primatelja:

- Odmah poslati `ACK` poruku (`quick acknowledge`) jer smo primili podatke odnosno paket koji je došao u ispravnom slijedu (nîzu) ili ako smo zaprimili podatke (paket) izvan nîza pa je potrebno odmah poslati `ACK` poruku kako bi se ponovno poslali podaci (paketi) od zadnjeg ispravnog broja paketa u nîzu.
- Nakratko odgoditi slanje potvrde (`ACK`) odnosno staviti ju u red za slanje (*schedule*) (`Delayed ACK`). Ovo se događa u slučajevima kada smo primili prvi paket u ispravnom nîzu te čekamo na drugi ili više njih, pa ćemo pričekati još neko vrijeme, te potvrditi zadnji u ispravnom nizu, standardno ili s takozvanom **SACK** porukom.

Prva metoda slanja potvrde odmah (`quick acknowledge`) je standardna pa ćemo objasniti `Delayed ACK` koji je definiran u standardu **RFC 1122**. Kako je definirano ovim standardom primatelj može odgoditi slanje `ACK` poruke za maksimalno 500ms. Ali isto tako primatelj sa standardnom `ACK` porukom može potvrditi svaki drugi paket (odnosno segment na TCP razini). Dakle može čekati primitak dva paketa prije nego potvrdi zadnji. `Delayed ACK` daje aplikaciji primatelja vremena, ali i mogućnost da promijeni `TCP receive window` pomoću **TCP Window scale** mehanizma, ako je to potrebno.
Dakle moguće je uz `ACK` poruku kojom odgovaramo pošiljatelju, poslati i promjenu veličine *TCP prozora (TCP Window size)* sve u jednoj poruci, što za neke primjene može uštedjeti broj poslanih paketa čak do tri (3) puta.

U operativnom sustavu Linux (ovisno kako je kernel kompiliran) moguće je optimizirati obje vrijednosti (uz oprez):

- Standardni (*quick*) *TCP ACK timeout* definiran je u: `net.ipv4.tcp_ato_min`
- *Delayed ACK timeout* definiran je u: `net.ipv4.tcp_delack_min`

Postoji i još jedna *sysctl* opcija vezana za ***TCP SYN ACK***: `net.ipv4.tcp_synack_retries` – njena zadana vrrijednost jest 5 (komada) – ona definira broj puta koliko se ide u ponovno slanje (*retransmit)* s porukom ***SYN, ACK*** koja je odgovor na inicijalnu ***SYN*** poruku ako je došlo do greške te paket nije potvrđen sa **SYN,ACK**. Druga strana tada **SYN** poruke ponavlja standardno 6 puta, definirano u: `net.ipv4.tcp_syn_retries`.

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju. Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju TCP potvrda (ACK):

| Naziv brojača | Opis |
|---|---|
| **TcpExtSyncookiesRecv** | Broj primljenih TCP paketa gdje je ACK redni broj važeći syncookie.. |
| **TcpExtSyncookiesFailed** | Broj primljenih TCP paketa gdje ACK redni broj nije važeći syncookie |
| **TcpExtTCPHPAcks** | Ovaj brojač prati broj TCP ACK paketa koji su potvrđeni (*acknowledged*) unaprijed (prediktivno). **TCP HP** (*High Performance*) ACKs se koriste u situacijama kada primatelj očekuje sljedeće pakete u toku prijenosa podataka, te unaprijed potvrđuje prijem (*acknowledge*) tih očekivanih podataka. To može poboljšati učinkovitost i brzinu prijenosa podataka. |
| **TcpExtTCPPureAcks** | Ovaj brojač prati broj čistih TCP ACK paketa koji nisu kombinirani s drugim podacima. Čisti ACK paketi obično sadrže samo potvrdu prijema prethodno primljenih podataka, bez dodatnih podataka koji se šalju s ACK-om. |
| **TcpExtDelayedACKs** | Ovaj brojač prati broj odgođenih (*delayed*) ACK paketa. ***TCP Delayed ACK*** mehanizam omogućuje da TCP prijemnik odgodi slanje potvrde prijema (ACK) ako očekuje dolazak dodatnih podataka. Ovo može smanjiti broj ACK paketa u mreži i poboljšati učinkovitost komunikacije. |
| **TcpExtDelayedACKLocked** | Ovaj brojač prati broj situacija u kojima je odgoda ACK paketa blokirana (*locked*). To se može dogoditi kada postoji problem s resursima ili kada postoje ograničenja u mrežnoj konfiguraciji koji sprječavaju odgodu ACK paketa. |
| **TcpExtDelayedACKLost** | O vaj brojač prati broj izgubljenih odgođenih ACK paketa. Izgubljeni ACK paketi mogu uzrokovati ponovno slanje podataka i usporiti komunikaciju između TCP klijenta i poslužitelja. |

**High Performance ACK**
High Performance ACK-ovi su posebna vrsta potvrde prijema u TCP protokolu koja se koristi kako bi se poboljšala učinkovitost i brzina prijenosa podataka. Umjesto da čeka dolazak podataka prije nego što pošalje potvrdu (**ACK**), TCP primatelj unaprijed potvrđuje prijem očekivanih podataka. Ovo se obično radi kada primatelj očekuje dolazak sljedećih paketa u toku prijenosa podataka i ima povjerenja da će oni stići bez problema. Prednosti TCP HP ACK-ova uključuju smanjenje latencije u komunikaciji, povećanje protoka podataka i smanjenje broja potvrda koje se moraju poslati.

***TCP HP ACK***-ovi su implementacijski detalj koji ne mora biti prisutan u svakoj implementaciji TCP-a.

**Izvori informacija: (325),(1406),(1477),(1651),(K-6), `man 7 tcp`,`man nstat`, RFC 1122, RFC 1213.**

## 24.2.6. Selektivne potvrde (*SACK*)

Selektivna potvrda primitka odnosno *Selective Acknowledgement* (**SACK**) TCP paketa (segmenta) definirana je u standardima **RFC 2018** i **RFC 2883**. Ova nova funkcionalnost omogućuje potvrđivanje primitka *TCP* mrežnih paketa individualno.
Naime u klasičnoj implementaciji TCP protokola kada je paket odnosno segment izgubljen, potrebno ga je ponovno poslati, dakle dolazi do ponovnog slanja odnosno *retransmisije*. Međutim potrebno je ponovno poslati i sve pakete/segmente poslije njega koji su ispravno zaprimljeni. Standardno se zaprimaju sâmo paketi koji dolaze u točnom redoslijedu (1,2,3,4,5,…) odnosno u nizu, a svi nakon tog niza koji su uredno primljeni se odbacuju te se traži retransmisija zadnjeg neispravnog paketa iz ispravnog niza uz cijeli nîz paketa koji slijedi nakon njih. Ovaj mehanizam radi tako jer se s `TCP ACK` porukom potvrđuje primitak paketa u nizu, odnosno nîza podataka koje paket nosi, pa se tako potvrđuje primitak podataka sve do recimo 4800 *bajta*. Recimo da smo to primili u prvih 5 paketa. Ako su nakon toga došli ispravni paketi broj:10,11,12,13,14 pa sve do 50 oni će biti odbačeni i zatražit će se *retransmisija* od paketa broj 6.

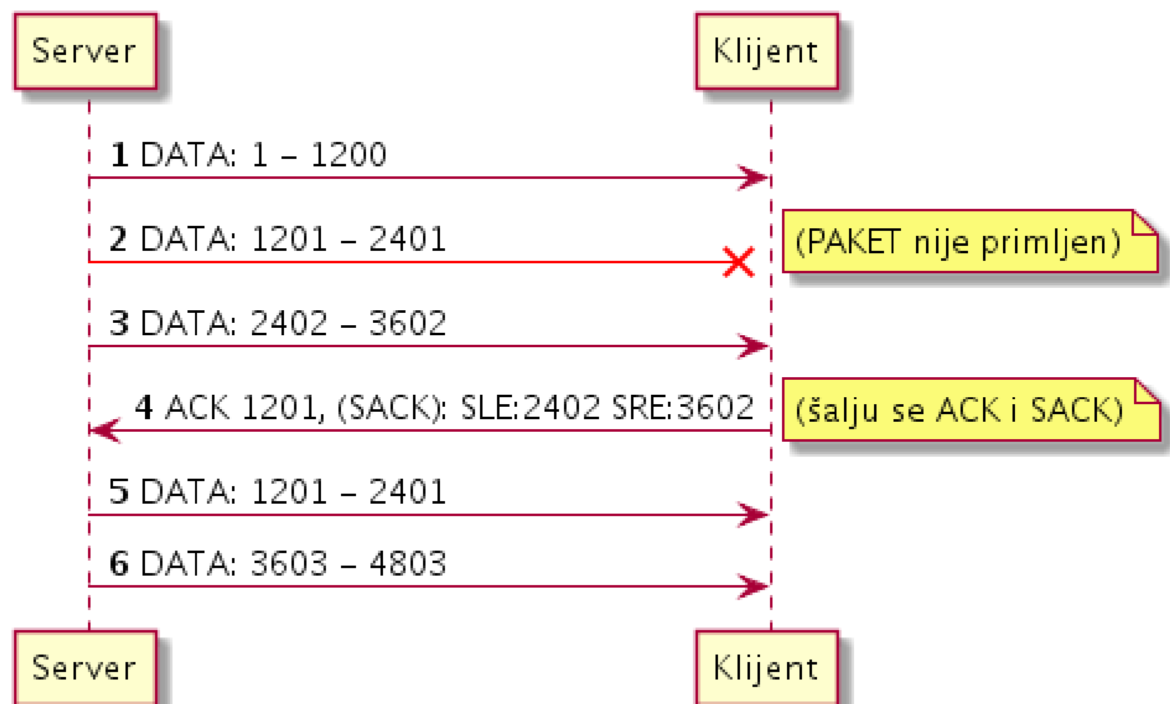
Dakle svi paketi koji su došli izvan ispravnog redoslijeda se odbacuju. Ovakav rad naravno može usporiti mrežu jer se paketi koji su ispravno zaprimljeni, a izvan su redoslijeda, proglašeni su neispravnima te se ponovno moraju slati preko mreže. Selektivno potvrđivanje je napravljeno upravo kako bi se riješilo ovaj problem. Ako obje strane u komunikaciji podržavaju **SACK** opciju, tada je moguće selektivno potvrđivati primitak mrežnih paketa.



**Mrežni segment na TCP sloju često pogrešno nazivamo mrežnim paketom, ali ćemo ga tako i nazivati zbog jednostavnosti i lakšeg razumijevanja.**

Pogledajmo kako izgleda ovakva komunikacija na slici 204.

**Slika 204. TCP komunikacija sa SACK porukama.**



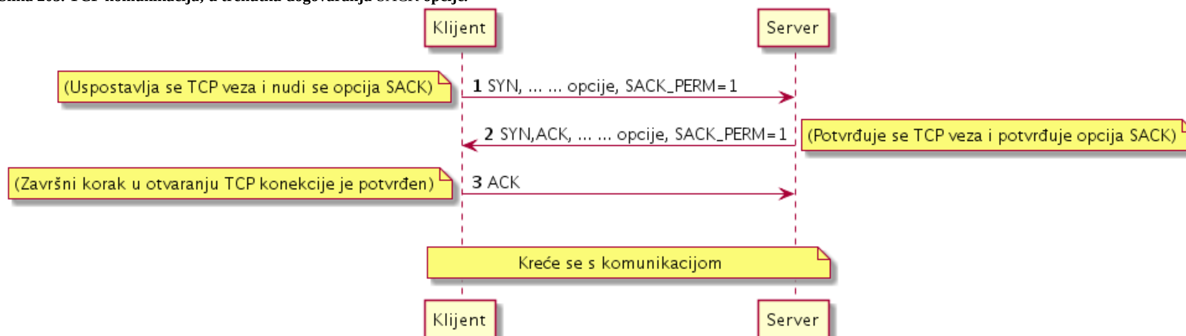

**1.** Paket na slici (**1**) je prvi paket s podacima, nakon što se odradilo otvaranje *TCP* komunikacijskog kanala porukama: `SYN`, `SYN ACK` i `ACK`. Šalje ga poslužitelj (*server*) prema klijentu. Ovaj paket sadrži podatke od 1 *bajta* do 1200 *bajta* i došao je do klijenta.
**2.** Paket (**2**) na slici šalje server prema klijentu, i on sadrži sljedeći blok podataka, od 1201 *bajta* do 2401 *bajta*, ali ovaj paket se izgubio na mreži ili je zaprimljen neispravan. U svakom slučaju nema ga (u ovom primjeru).
**3.** Paket server šalje prema klijentu i on sadrži sljedeći blok podataka od 2402 *bajta* do 3602 *bajta* i klijent ga uredno zaprima.
**4.** Paketom (**4**) klijent potvrđuje primitak sve do 1201 *bajta* (poruka `TCP ACK 1201`) što bi bilo normalno i bez *SACK* mehanizma, ali šalje se i `SACK` poruka unutar paketa (`SLE:2402 SRE:3602`) kojom se potvrđuje i primitak podataka od 2402 *bajta* do 3602 *bajta* što znači i sljedećeg paketa (paketa broj **3**.). Dakle *SACK* poruka sadrži ***SLE*** i ***SRE*** vrijednosti koje označavaju pakete izvan slijeda, od kojih ***SLE*** (*Left Edge*) označava početni *bajt* podataka (unutar paketa) a *SRE* (*Right Edge*) završni *bajt*.
**5.** Paket koji server prvo šalje je onaj izgubljeni paket broj **2** odnosno blok podataka od 1201 *bajta* do 2401 *bajta* kao odgovor na poruku poruka `TCP ACK 1201`.
**6.** Potom server nastavlja sa slanjem podataka od bloka 3603 *bajta* do 4803 *bajta* dakle prelazi se odmah na **6**. paket.

U stvarnoj komunikaciji između paketa broj **2** do paketa broj **3** moglo je biti poslano i primljeno na stotine ili tisuće paketa, te bi upotrebom *SACK* mehanizma prvo dobili neispravan paket broj **2**, a onda bi se nastavilo sa slanjem i primanjem paketa broj stotinu (ili tisuću) i nešto, s čime bi se uštedjelo na propusnosti odnosno opterećenju mreže.

***SACK*** je opcija koja se dogovara u trenutku uspostavljanja *TCP* komunikacijskog kanala. Klijent pri uspostavljanju TCP komunikacijskog kanala nudi opciju `SACK`, a poslužitelj (*server*) odgovara, podržava li ju, ili ne. Ako ju podržava, to potvrđuje u sljedećoj poruci (`SYN ACK`) te se u daljoj komunikaciji ova opcija odnosno funkcionalnost može početi koristiti.

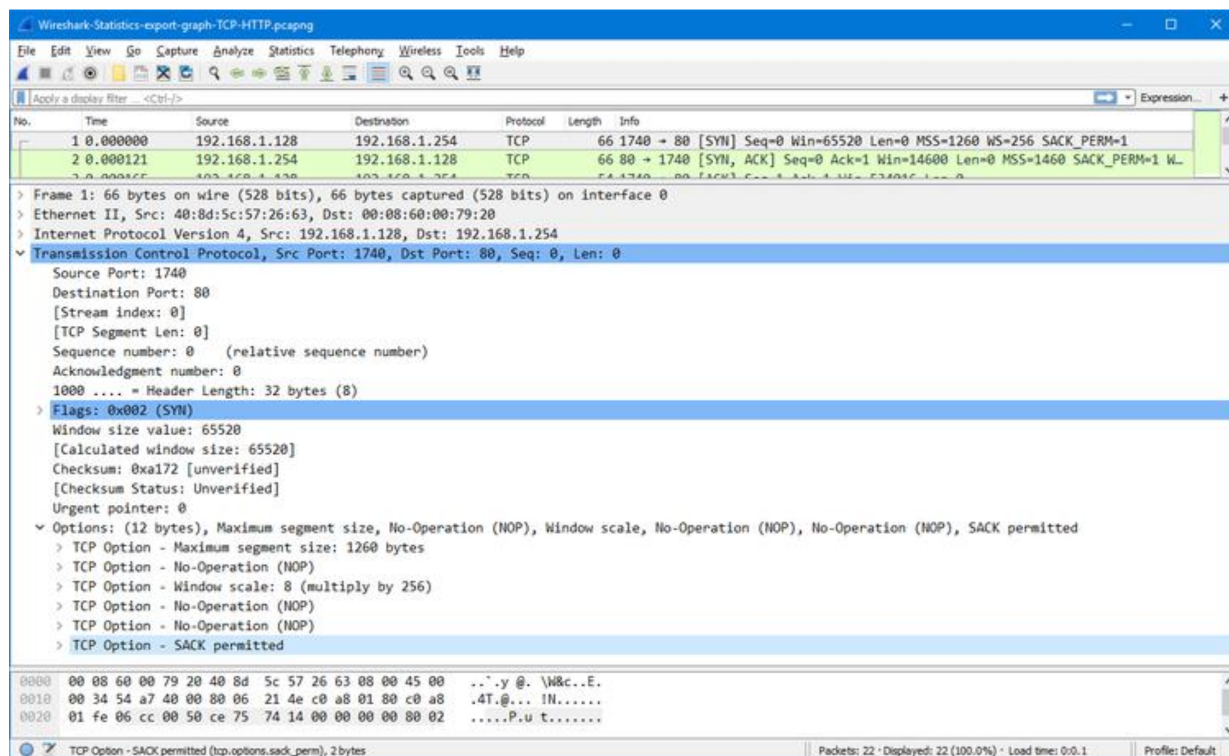
Pogledajmo dijagram toka, na slici 205 u trenutku dogovaranja `SACK` opcije.

**Slika 205. TCP komunikacija; u trenutku dogovaranja** SACK **opcije.**




Pogledajmo jedan stvarni `SYN` paket (za otvaranje TCP veze) snimljen programom **Wireshark** koji sadrži `SACK_PERM` odnosno zahtjev za korištenjem opcije `SACK` , kako je vidljivo, kao zadnja opcija na slici 206.

**Slika 206. TCP komunikacija: pogled na mrežni paket koji sadrži zahtjev za korištenjem** SACK **opcije.**




Pogledajmo kako ova `SACK` opcija odnosno ono što vidimo kao `SACK_PERM=1` izgleda kod `TCP SYN` paketa:

```
TCP Option - SACK permitted
    Kind: SACK Permitted (4)
```

Ista opcija slijedi u odgovoru s kojim nam druga strana potvrđuje otvaranje TCP veze (poruka: `TCP SYN, ACK`):

```
TCP Option - SACK permitted
    Kind: SACK Permitted (4)
```

U ranim implementacijama *SACK* funkcionalnosti prema **RFC 2018** pojavili su se određeni problemi kod slučajeva u kojima je primatelj mogao zaprimiti paket izvan niza (ispravnog redoslijeda), zbog problema u mreži, kada bi pošiljatelj zbog mehanizma brze retransmisije (*TCP Fast Retransmission*) mogao ponovno poslati paket bez čekanja na potvrdu, sličnu kao što je kod standardnog `TCP duplicate ACK` slučaja.



Pogledajte poglavlje:
**24.2.13 Mehanizam TCP retransmisije.**

U novom standardu **RFC 2883** uvedeno je duplicirano selektivno potvrđivanje **DSACK** (engl. *duplicate selective acknowledgment*) odnosno poruka: `D-SACK` koja rješava navedeni problem na sličan način kao i `TCP duplicate ACK`.

Pogledajmo postavke Linuxa (*sysctl*) s kojima možemo optimizirati ovo ponašanje odnosno `SACK` mehanizam:

- `net.ipv4.tcp_sack` – (*zadano je 1*) – definira je li **SACK** uključen (1) ili isključen (0).
- `net.ipv4.tcp_dsack` – (*zadano je 1*) – definira je li **DSACK** uključen (1) ili isključen (0).
- `net.ipv4.tcp_fack` – (*zadano je 1*) – definira je li **FACK** uključen (1) ili isključen (0).

**Napomena:** Aktivacijom sysctl varijable `net.ipv4.tcp_fack` uključuju se **FACK** (*Forward ACKnowledgement*) *congestion* i *fast retransmission*. **FACK** je posebna funkcionalnost (algoritam) koji radi pod **SACK** mehanizmom, a zadužen je za tzv. congestion control mehanizme za nadzor zagušenja.

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju. Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju TCP potvrda (**ACK**, **SACK** i **DSACK**):

| Naziv brojača | Opis |
|---|---|
| **TcpExtTCPSackShifted** | Ovaj brojač bilježi broj TCP SACK blokova (*Selective Acknowledgment*) koji su premješteni (*shifted*) prema početku TCP veze. SACK blokovi se koriste kako bi se označili segmenti koji su primljeni izvan redoslijeda i kako bi se omogućilo ispravljanje redoslijeda tih segmenata. |
| **TcpExtTCPSackMerged** | Ovaj brojač bilježi broj TCP SACK blokova koji su spojeni (*merged*) u jedan zajednički blok. Ovo se događa kada se dva ili više uzastopnih SACK blokova mogu spojiti u jedan blok, što može smanjiti složenost ACK paketa. |
| **TcpExtTCPSackShiftFallback** | Ovaj brojač bilježi broj situacija u kojima se TCP prebacuje na manje učinkovit mehanizam potvrde (ACK) nakon neuspjelog pokušaja uporabe SACK-a. To se obično događa kada SACK blokovi nisu dostupni ili nisu prihvaćeni od strane druge strane komunikacije. |
| **TcpExtTCPSackRecovery** | Ovaj brojač bilježi broj puta kada se koristi TCP mehanizam oporavka (*recovery*) pomoću SACK-a. SACK omogućuje TCP-u da učinkovitije reagira na izgubljene segmente tijekom komunikacije i da ih ponovno pošalje. |
| **TcpExtTCPSackRecoveryFail** | Ovaj brojač bilježi broj neuspjelih pokušaja TCP-a da se oporavi (recovery) korištenjem SACK-a. To se može dogoditi kada SACK blokovi nisu dostupni ili nisu pouzdani, ili kada postoje problemi s primanjem potvrda od strane druge strane komunikacije. |
| **TcpExtTCPSACKReneging** | Ovaj brojač bilježi broj puta kada se TCP povlači (*reneging*) na SACK opciju tijekom komunikacije. Reneging se događa kada TCP odustane od korištenja SACK opcije nakon što je već bila aktivirana. |
| **TcpExtTCPSACKReorder** | Ovaj brojač bilježi broj puta kada su TCP podaci primljeni izvan redoslijeda (reordered) tijekom SACK oporavka. To se može dogoditi kada SACK omogućuje ponovno slanje izgubljenih podataka u različitim redoslijedima. |
| **TcpExtTCPDSACKRecv** | Ovaj brojač bilježi broj primljenih TCP DSACK (*Duplicate Selective Acknowledgment*) potvrda. DSACK potvrde koriste se kako bi se označili duplikati podataka koji su primljeni. |
| **TcpExtTCPDSACKUndo** | Ovaj brojač bilježi broj puta kada TCP poništava (*undo*) primljenu DSACK potvrdu. Ovo se obično događa kada TCP primi DSACK potvrdu koja više nije valjana ili je već obrađena. |
| **TcpExtTCPDSACKOldSent** | Ovaj brojač bilježi broj puta kada su stariji (*old*) podaci ponovno poslani (*resent*) na temelju primljene DSACK potvrde. To se može dogoditi kada TCP primi DSACK potvrdu koja ukazuje na problem s prethodno poslanim podacima. |
| **TcpExtTCPSackFailures** | Ovaj brojač bilježi broj neuspjelih pokušaja korištenja TCP SACK opcije tijekom komunikacije. Neuspjeli pokušaji mogu se dogoditi zbog različitih problema, uključujući nedostupnost SACK blokova ili nedostatak podrške za SACK opciju na drugoj strani komunikacije |
| **TcpExtTCPSACKDiscard** | Ovaj brojač bilježi broj TCP SACK potvrda koje su odbačene ili ignorirane. To se može dogoditi primjerice zbog nedostatak resursa, neispravne SACK potvrde ili problema u procesiranju SACK opcija na strani primatelja. |

**Izvori informacija: (732),(1406),(1477), (1651),(K-6), `man 7 tcp`,`man nstat`, RFC 2018, RFC 2883. RFC 1213..**

## 24.2.7. TCP mehanizmi

TCP protokol koristi nekoliko mehanizama u radu, pogledajmo neke od osnovnih u sljedećim cjelinama.

### 24.2.7.1. Vrijeme prolaska paketa (*Round-Trip Time*)

*Round-trip time* (**RTT**) znan i kao *round-trip delay* je vrijeme koje je potrebno kako bi signal odnosno konkretno mrežni paket stigao od pošiljaoca do primatelja. Za *TCP* protokol ovo vrijeme odnosno izračun vremena je definiran u RFC 6298 standardu. Na ovo vrijeme utječe cijeli nîz parametara, poput:

- Brzine prijenosa podataka od izvora prema odredištu.
- Vrsti i prirodi medija kroz koji se prenose podaci: bakar, optika, bežična veza poput klasičnog *wirelessa*, satelitske komunikacije i slično.
- Fizička udaljenost između izvora i odredišta (zbog fizikalnih zakona).
- Broj mrežnih uređaja (preklopnika, usmjerivača, vatrozida i sl.) između izvora i odredišta kao i:
  - Broj drugih upita od i prema mrežnim uređajima i opterećenje istih, koji se nalaze na komunikacijskom putu između izvora i odredišta.
  - Brzine i kašnjenja mreže između navedenih mrežnih uređaja.
- Količine prometa i opterećenosti lokalne (*LAN*) mreže.
- Smetnje u svakoj komunikacijskoj točki između svih uređaja na mreži, uključenih u komunikaciju.

U raznim mrežama, a poglavito u **WAN** (*wide-area network*) mrežama preko kojih se ostvaruju veze od i prema internetu, dolazi do kašnjenja između slanja i primanja signala/podataka odnosno paketa. *WAN* mreže su posebno osjetljive zbog činjenice što povezuju veće udaljenosti poput onih između više lokacija tvrtki ili između zgrada, kampusa, gradova, država, kontinenata i slično. Naime vremena kašnjenja signala u lokalnim mrežama su obično nekoliko milisekundi za 1Gbps mreže, a za 10Gbps mreže čak nekoliko desetaka mikro sekundi. Dok su vremena kašnjenja na *WAN* vezama već u granicama milisekundi za veze unutar jednog grada, do desetak milisekundi između gradova, sve do stotinjak ili više milisekundi za veze između kontinenta, pa čak do nekoliko sekundi za “najudaljenije” veze.



Vezano za kašnjenje (latenciju) pogledajte poglavlje:
**19.7. Latencija odnosno kašnjenje.**

### RTT i (TCP) RTO vremena

Ovo vrijeme prolaska se naziva **RTT** (*Round-trip time*) vrijeme. Ovo vrijeme se može mijenjati i mijenja se svako malo, ovisno o problemima u mreži, odabiru drugog puta, opterećenju mrežnih uređaja kroz koje prolaze podaci i slično. Stoga je potrebno konstantno pratiti **RTT** vrijeme jer je ono jedan od važnih mehanizama i u ***TCP*** komunikaciji između dvije krajnje točke. Primjerice u *TCP* komunikaciji je važno vrijeme koje se čeka nakon što je mrežni paket poslan, do potvrde da je zaprimljen (s `ACK` porukom) s druge strane. Dakle TCP koristi **RTT** vrijednost za podešavanje vrijednosti svog vremenskog brojača ponovnog slanja (tzv. **RTO**). To znači da se vrijednost **RTO** koristi za određivanje koliko dugo TCP treba čekati prije ponovnog slanja paketa koji primatelj nije potvrdio. Dakle, ako ovaj vremenski okvir istekne, pošiljatelj kreće s retransmisijom paketa (podataka) koji nisu potvrđeni. Ovaj vremenski okvir se zove **RTO** (*Retransmittion Timeout*) te se konstantno i ponovno izračunava na osnovu izračunatog trenutnog **RTT** vremena, koje se stalno mijenja, zbog navedenih pomjena na mreži.



Pogledajte i poglavlje:
**24.2.13 Mehanizam TCP retransmisije**.

Naime **RTT** i **RTO** nisu isti, jer **RTO** osim vremena prolaska paketa uzima u obzir određena odstupanja i druge parametre, koja se u konačnici izračunavaju kao **RTO** vrijeme. Vezano za izračun **RTO** vremenskog okvira postoji više metoda odnosno algoritama za njegov izračun. Svaki od ovih algoritama ima svoje prednosti i mane, kao i najčešću ili preporučenu primjenu i ovise o operativnom sustavu, a na Linuxu ovise i o inačici kernela jer su se formule za izračun s vremenom mijenjale odnosno prilagođavale novim spoznajama i preporukama.



O ovim algoritmima govorit ćemo u poglavlju:
**24.2.9. Nadzor zagušenja (*Congestion control*).**

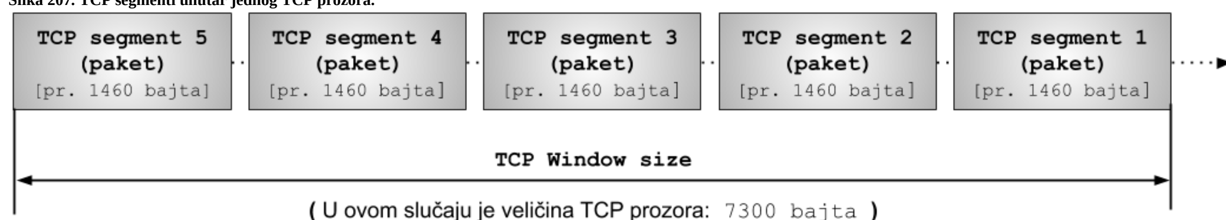
**Izvor informacija:** `man 7 tcp`, **RFC 6298.**

## 24.2.8. Kontrola protoka (*TCP Window* i *Window scaling*)

Sjetimo se kako je *TCP* transportni protokol taj koji se brine o vezi između dvije točke u komunikaciji. Pri tome oba kraja komunikacijske veze prate stanje svih prenesenih podataka, tako da se bilo koji izgubljeni ili prekinuti segmenti podataka mogu ponovno poslati radi održavanja pouzdanog prijenosa. Dodatno *TCP* koristi i mehanizme kontrole protoka (*flow control*) kako bi se izbjeglo da pošiljatelj šalje podatke prebrzo za prijemnu stranu. Pošto ih ona tada ne bi stigla obraditi jer bi došlo do zagušenja prijemne strane, primjerice jer joj je trenutno preopterećen CPU ili slično. Dakle mehanizmi za kontrolu protoka su vrlo važni, a pogotovo u okruženjima u kojima komuniciraju strojevi različitih brzina i različite propusnosti mreže.
Konkretno *TCP* koristi mehanizam za kontrolu protoka pomoću takozvanog kliznog prozora znanog kao TCP klizajući prozor tj. *TCP sliding window* odnosno *TCP window*. Drugim riječima ovaj *TCP* klizni prozor odnosno pojednostavljeno ***TCP prozor*** definira koliko je prostora dostupno međumemoriji (*bufferu)* prijemnika, za spremanje pristiglih paketa. Možemo to promatrati i ovako: *TCP prozor* odnosno klizni prozor je prozor unutar kojega se šalje više mrežnih paketa za koje je od strane primatelja potrebna sâmo jedna potvrda (s porukom *TCP ACK*). Ovaj TCP prozor se koristi od strane prijemnika i kako bi obavijestio pošiljatelja da treba ubrzati ili usporiti slanje podataka preko mreže. Ako prijemnik želi da pošiljatelj zaustavi potpuno odašiljanje, može postaviti vrijednost nula (0) za veličinu *TCP prozora* i pošiljatelj će privremeno zaustaviti slanje podataka na mrežu, prema tom konkretnom primatelju.

Pogledajmo kako logički izgleda jedan *TCP prozor (TCP window)* i segmenti (paketi) koji se šalju unutar njega (slika 207).

**Slika 207. TCP segmenti unutar jednog TCP prozora.**




U primjeru na slici veličina *TCP prozora* je 7300 *bajta* pa unutar njega stane maksimalno pet (5) segmenata odnosno mrežnih paketa, veličine 1460 *bajta*.
Osim sâme propusnosti mreže važan parametar u komunikaciji je kašnjenje odnosno latencija između poslanih i primljenih paketa na mreži. Ovo kašnjenje znatno utječe na pouzdane protokole usmjerene na vezu kao što je *TCP*. *TCP* je protokol temeljen na sesiji koji osigurava isporuku paketa i minimizira gubitak paketa. Kao rezultat takvog rada stvarna propusnost često može biti znatno niža od kapaciteta veze koju koristimo. Naime za razliku od *UDP*-a, *TCP* zahtijeva potvrdu uspješno primljenih paketa. Međutim neučinkovito je čekati uspješno potvrđivanje svakog pojedinačnog paketa, prije slanja sljedećeg paketa. To posebno vrijedi za udaljene mreže s velikim kašnjenjem odnosno latencijom. *TCP* koristi pojam odnosno metodu *veličine prozora* kako bi se utvrdilo koliko paketa u jednom nîzu odnosno takozvanom *prozoru* se može poslati bez gubitaka.
*TCP* pri tome čeka potvrdu primitka (*TCP ACK* poruku) kako bi se nastavilo sa slanjem novog nîza paketa odnosno *prozora* s paketima. Ako paketi nedostaju odnosno nisu zaprimljeni na odredištu, *TCP* na odredištu šalje *TCP ACK* poruku prema pošiljatelju s potvrdom zadnjeg zaprimljenog paketa, kako bi zatražio ponovni prijenos (*retransmisiju*) odnosno slanje od pošiljatelja. Kako bi uopće bio moguć efikasan prijenos i primanje veće količine paketa, sa sâmo jednom potvrdom sa strane primatelja, *TCP* koristi tehniku zvanu *TCP* ***windowing*** u kojoj se polako povećava veličina prozora odnosno broj paketa koji se šalju u nîzu unutar jednog *TCP prozora*, a za koji se čeka na *TCP ACK* potvrdu sa strane primatelja.

Potom se polako povećava brzina prijenosa, tako da se povećava veličina TCP prozora i sâmim time broj paketa unutar jednog *TCP prozora*, kako bi u konačnici odgovarao raspoloživoj širini komunikacijskog pojasa. Isto tako brzina prijenosa odnosno *TCP prozor* se brzo smanjuje kada su paketi izgubljeni i nisu potvrđeni (s *TCP ACK porukom*) unutar očekivanog vremenskog okvira. Za to su pak zaduženi algoritmi za nadzor zagušenja (Engl. *Congestion control*).
Kao rezultat toga, na propusnost uvelike utječe kašnjenje odnosno latencija kao i gubitak paketa u vezi.
Za iste mehanizme, kako bi se nadoknadio ograničeni prostor memorijskih među spremnika gdje su primljeni podaci privremeno pohranjeni dok ih odgovarajuća aplikacija ne može obraditi, *TCP* točke u komunikaciji pristaju ograničiti količinu nepotvrđenih podataka koji mogu biti u tranzitu u bilo kojem trenutku. Taj mehanizam se naziva veličina prozora (*Window size*) i komunicira se putem 16-bitnog polja u *TCP* zaglavlju. Pošto je ovo 16. bitni broj ($2^{16}$) maksimalna veličina ovog prozora koja se može definirati je 65.535 *bajta* odnosno 64 kB. Međutim postoji i proširenje ovog standarda za mreže koje zahtijevaju velike performanse ili imaju neke druge specifičnosti. Standard u kojemu je definirano navedeno proširenje je RFC 1323, a o njemu ćemo govoriti u drugom dijelu ovog poglavlja. Naime u *TCP* komunikaciji koristi se takozvani ***TCP Receive Window*** odnosno ***RWIN*** koji definira količinu podataka koje računalo može prihvatiti bez potvrđivanja paketa sa *TCP ACK* porukom, prema pošiljatelju. Ako pošiljatelj nije primio potvrdu (*TCP ACK*) o prvom paketu koji je poslao, zaustavit će se i čekati, a ako ovo čekanje premaši određenu granicu, paket će biti ponovno poslan. Tako *TCP* ostvaruje pouzdani prijenos podataka.



*Mehanizam TCP prijemnog prozora (kod* **TCP Receive Window**-*a) radi od primatelja prema pošiljatelju.*

Čak i ako nema gubitka paketa u mreži, *TCP* prijemni prozor (*TCP Receive Window*) može ograničiti propusnost. Budući da *TCP* prenosi podatke do veličine *prozora* prije čekanja potvrđivanja sa *TCP ACK* porukom, cijela širina pojasa mreže možda neće uvijek biti iskorištena.

Ograničenje stvarne propusnosti mreže zbog veličine prozora može se izračunati na sljedeći način:

$$\boldsymbol{Propusnost\ mre\check{z}e} \leq \frac{Receive\ Window}{Round\ Trip\ Time}$$

Dakle propusnost je manja ili jednaka veličini prijemnog prozora ***RWIN*** (*Receive Window*) podijeljenoj s izračunanim ***RTT*** (*Round-Trip Time*) vremenom prolaska paketa do odredišne točke, u datom trenutku.
Dakle u *TCP* komunikaciji sa strane primatelja svako malo oglašava se veličina prijemnog prozora koja se naziva *TCP Receive Window* ili samo *TCP Window*. U bilo kojem trenutku prozor koji oglašava prijemna strana *TCP konekcije* mora odgovarati količini slobodne memorije koja je rezervirana za tu namjenu i koju je sustav dodijelio za ovu vezu. Dakle strana koja prima podatke prvo rezervira dio memorije za *receive window* odnosno za primanje veće količine paketa to jest podataka.

Tek potom oglašava strani koja šalje pakete (podatke), koliko podataka može primiti, pomoću oglašavanja *TCP receive window*a odnosno prijemnog TCP prozora unutar kojeg može primati navedene podatke.
Kada ovaj mehanizam ne bi postojalo, riskiralo bi se odbacivanje primljenih paketa odnosno podataka zbog: nedostatka memorijskog prostora, eventualnog zagušenja aplikacije, procesora ili slično. Na strani koja šalje, također se treba dodijeliti jednaka količinu memorije kao i na prijemnoj strani. To je zato što čak i nakon slanja podataka na mrežu, strana koja šalje mora ih zadržati u memoriji, sve dok se ne potvrdi da su uspješno primljeni, jer ako nisu morat će doći do njihovog ponovnog slanja (*retransmisije*) iz te iste memorije.

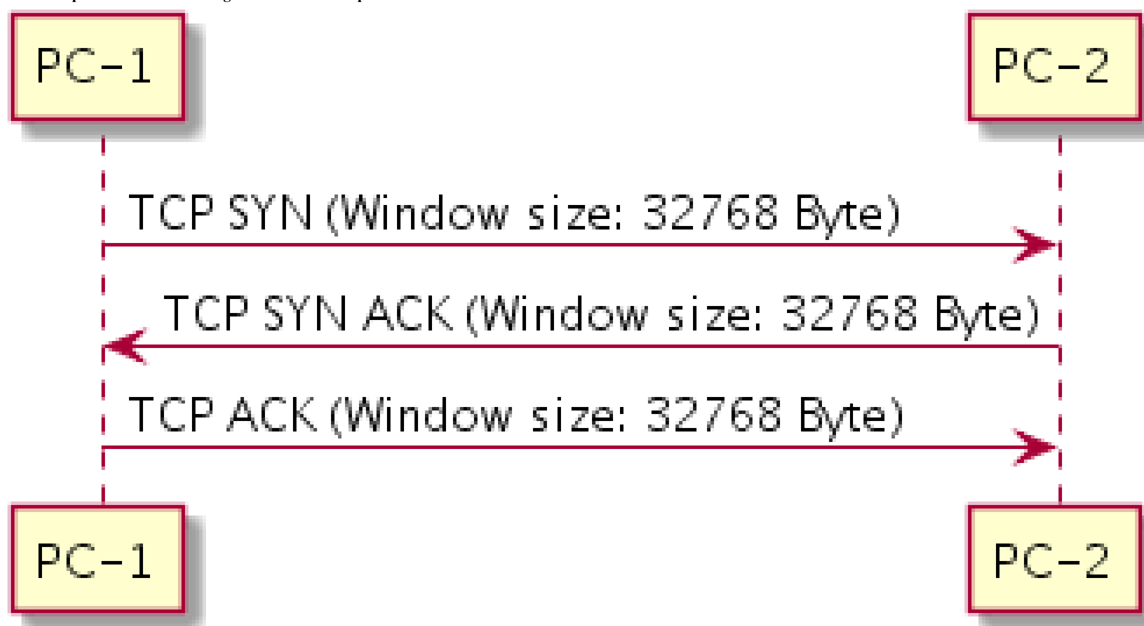
Ako je prijemna strana daleko, potvrdama će jasno trebati mnogo više vremena za dolazak do pošiljatelja.
U slučajevima kada je pak memorija za slanje mala, može se prepuniti i blokirati slanje na neko vrijeme.
Jednostavan izračun daje istu optimalnu veličinu memorije za slanje kao i gore navedenu veličinu memorije za primanje.
Dakle cijelo vrijeme se u TCP komunikaciji, od prijemne strane prema predajnoj, šalje veličina prijemnog prozora kojoj se predajnik treba stalno prilagođavati, kako se prijemna strana ne bi zagušila. Naime standardno u svakoj potvrdnoj poruci (*TCP ACK*), ali i bilo kojoj drugoj povratnoj poruci prema pošiljatelju, šalje se i nova veličina *TCP* prozora na koju se pošiljatelj mora prilagoditi.

Ne zaboravimo kako je *TCP* komunikacija dvosmjerna u dva kanala, što znači kako postoji po jedan komunikacijski kanal:

- Klijenta prema poslužitelju.
- Poslužitelja prema klijentu.

Čak i u trenutku otvaranja TCP veze dolazi do razmijene veličine *TCP prozora (TCP Window)*. Pogledajmo pojednostavljenu sliku (208) u kojoj obje strane za oba komunikacijska kanala dogovaraju *TCP prozor* veličine 32.768 *bajta* odnosno 32 kB.

**Slika 208. Uspostava TCP veze i dogovor veličine TCP prozora.**




Ovdje se dogodilo sljedeće. Prvo su oba računala u komunikaciji alocirala 32.768 *bajta* odnosno 32 kB memorije za slanje:

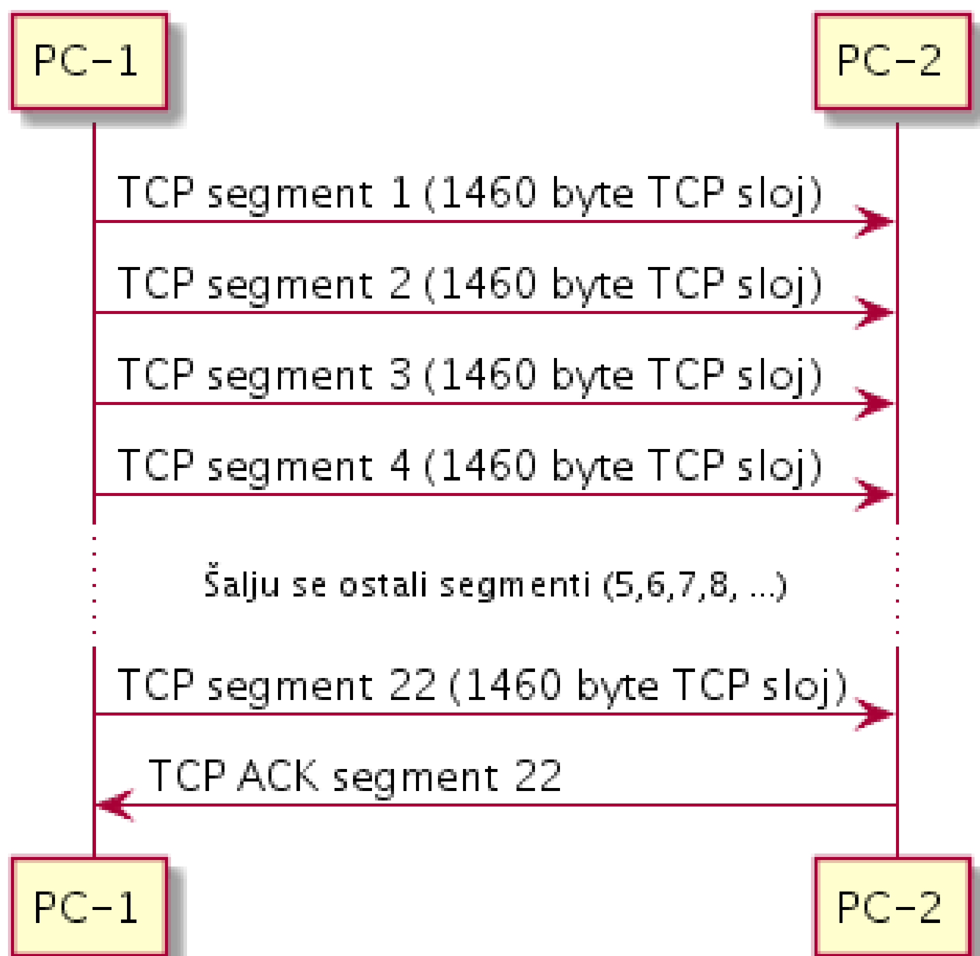
1. **PC-1** šalje prvi paket `TCP SYN` u kojemu kao opciju ima `tcp.window_size_value=32768` i šalje ju prema računalu **PC-2.**
2. **PC-2** potvrđuje prvi paket i šalje `TCP SYN ACK` u kojemu kao opciju šalje istu veličinu `tcp.window_size_value=32768` i šalje ju prema računalu **PC-1.**
3. **PC-1** potom šalje odgovor na drugi paket te šalje `TCP ACK` u kojemu kao opciju isto šalje `tcp.window_size_value=32768` i šalje ju prema računalu **PC-2.**

S ovime se ostvario *TCP* komunikacijski kanal u dva smjera, a preko kojega će krenuti TCP promet. Pošto oba računala sada imaju potvrdu da je veličina *TCP* prozora 32kB, prvo računalo može krenuti sa slanjem podataka. To znači kako će pošiljatelj moći poslati pakete koji zbrojno (na TCP sloju) mogu sadržavati podatke do 32.768 *bajta* prije nego ih primatelj potvrdi.

Krećemo sa slanjem, kako je vidljivo na slici 209:

1. **PC-1** kreira paket, standardno maksimalne ukupne veličine 1500 *bajta* jer je to maksimalna veličina koju može poslati na mrežu odnosno to je ***MTU*** veličina. Od ovog MTUa se dio paketa od TCP protokola zove MSS (*Maximum Segment Size*) odnosno ono što ubacuje TCP sloj, bez *IP* sloja. A on je za ovaj ***MTU*** maksimalne veličine 1460 *bajta*. Dakle govorimo o dijelu paketa na *TCP* razini koji je ovdje veličine 1460 *bajta*.
2. **PC-1** šalje još 21 paket što ih ukupno čini 22. Zbog toga što je 22 x 1460 = 32.120 što je malo manje od maksimalne veličine prozora koju može odjednom poslati, a koja je 32.760 *bajta* i kolika je *buffer* memorija na prijemnoj strani.
3. **PC-2** sada šalje prvi potvrdni paket TCP ACK prema **PC-1**. On s ovim paketom potvrđuje prijem svih 22 paketa koja su stala unutar jednog dogovorenog *TCP* prozora (*TCP Window*). Da budemo tehnički precizni, on potvrđuje svih 22 segmenta podataka koje je primio u svoju (*buffer)* memoriju koja je trenutno veličine 32.768 *bajta*.

**Slika 209. TCP segmenti u komunikaciji.**




Pogledajmo i kako izgleda jedan TCP segment snimljen s programom ***Wireshark*** (ili s programom/naredbom. `tcpdump`). Fokusirajmo se na OSI Sloj **4** (TCP) i to opciju `Window size value`. Dakle naš mrežni paket izgleda ovako:

```
1.  Frame 47: 147 bytes on wire (1176 bits), 147 bytes captured (1176 bits
2.  Ethernet II, Src: Fujitsu_b3:5f:b6 (2c:d4:44:b3:5f:b6), Dst: Cisco_85:4c:c6
(00:12:00:85:4c:c6)
3.  Internet Protocol Version 4, Src: 10.50.200.182, Dst: 10.50.247.69
4.  Transmission Control Protocol, Src Port: 38762, Dst Port: 84, Seq: 434, Ack:
3191, Len: 93
        Source Port: 38762
        Destination Port: 84
        [Stream index: 7]
        [TCP Segment Len: 93]
        Sequence number: 434    (relative sequence number)
        [Next sequence number: 527    (relative sequence number)]
        Acknowledgment number: 3191    (relative ack number)
        0101 .... = Header Length: 20 bytes (5)
        Flags: 0x018 (PSH, ACK)
          Window size value: 32768
          [Calculated window size: 32768]
          [Window size scaling factor: -2 (no window scaling used)]
        Checksum: 0x5c2c [unverified]
        [Checksum Status: Unverified]
```

**Postoje i dodatne varijacije**
Ako je u bilo kojem trenutku *(buffer)* memorija primatelja potpuno prepunjena jer nije uspio obraditi prethodne podatke i isprazniti ju, on u pravilu šalje strani koja mu šalje podatke potvrdu primljenih segmenata u kojoj vrijednost opcije `tcp.window_size_value` postavlja na nulu (0), što odašiljačkoj strani indicira da više *ništa* ne šalje.
Moguće je i drugo među stanje u kojemu se primateljeva *(buffer)* memorija nije u potpunosti prepunila, već je aplikacija koja obrađuje podatke uspjela recimo preuzeti ili obraditi samo polovicu podataka, odnosno 16.384 *bajta*. Tada primatelj može pošiljatelju poslati potvrdu uz uključenu opciju `tcp.window_size_value` postavljenu na 16.384, što znači kako može primiti segmenata, samo koliko stane u 16.384 kB što bi u gornjem slučaju bilo **11** novih segmenata (paketa) koje može primiti.

Sljedeća varijacija bi bila kada je primatelj povećao *buffer* memorije, odnosno ***Receive window*** na maksimalno, odnosno na 65.535 *bajta (*64 kB*)*. Tada bi on u sljedećoj potvrdi poslao i opciju `tcp.window_size_value` postavljenu na 65.535 *bajta*. Potom bi pošiljatelj trebao prilagoditi i svoju veličinu *buffera* odnosno svoj ***Send Window***, a nakon toga bi mogao krenuti sa slanjem više segmenata preko mreže, koji stanu u ovu novu povećanu veličinu *TCP* prozora.

U ovom slučaju bi se mogla poslati 44 segmenta koja bi u *TCP* dijelu nosila 1460 *bajta* podataka, na koje bi primatelj nakon što ih ispravno primi, za sva 44 segmenta poslao jednu jedinu potvrdu `TCP ACK`, a s kojom bi ponovno mogao promijeniti svoj *TCP Receive Window* odnosno *buffer* memorije, ako je to potrebno. Pomoću ovih varijacija, indirektno se radi kontrola protoka slanja paketa, odnosno ubrzavanje, usporavanje ili zaustavljanje slanja istih, od primatelja prema pošiljatelju.

Jedan od primjera upotrebe ovih mehanizama kontrole toka je *SSH* protokol za udaljeni pristup. Komunikacija *SSH* protokola s transportnim slojem je definirana u RFC 4253. Naime *SSH* protokolom se obično spajamo na udaljeni terminal nekog računala, kako bismo "*tipkali*" neke naredbe odnosno nešto radili na tom udaljenom računalu. Stoga je za svaku stisnutu tipku (ili nekoliko njih), potrebno poslati po jedan *TCP* paket prema udaljenoj strani i dobiti brzi odgovor od nje (je li prihvaćena stisnuta tipka), a između kojih je potrebno potvrditi primljene pakete (s `TCP ACK` porukom).

Stoga *SSH* s aplikacijske razine, za *SSH* komunikaciju, nalaže da se ovaj *SSH* komunikacijski kanal preko *TCP* transportnog protokola odrađuje tako, da se *TCP Window* smanji do te granice, kako bi se spriječilo slanje velikog broja *TCP* segmenata preko mreže unutar jednog *TCP windowa*.

To je tako jer ova komunikacija mora biti dvosmjerna i sa što bržim odzivom s druge strane.
Dakle ovdje se koristi kontrola protoka pomoću *TCP Windowa* i to za svaku stranu TCP komunikacije:

- Klijent → poslužitelj (*SSH*).
- Poslužitelj → klijent (*SSH*).

**Izvor informacija za SSH komunikaciju i *TCP prozor (TCP Window):* (264)**

Pogledajmo tablicu s varijablama koje postavljaju memorijska ograničenja po pitanju TCP memorije:

| Naziv `sysctl` varijabli | Opis sysctl varijabli |
|---|---|
| **`net.ipv4.tcp_rmem`**<br>**`net.ipv4.tcp_wmem`** | Definiraju veličinu RAM memorije (**R**/**W**) za svaki pojedini *TCP socket* pa su stoga ograničenje i za maksimalnu veličinu *TCP prozora*.<br>Pogledajte poglavlje:<br>**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).** |

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju. Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju TCP prozora:

| Naziv brojača | Opis |
|---|---|
| **`cpExtTCPFromZeroWindowAdv`** | Ovaj brojač bilježi broj puta kada TCP primatelj oglašava (*advertises*) svoj prozor veličine nula (*zero window*) prema TCP pošiljatelju. Prozor veličine nula signalizira da TCP primatelj trenutno ne može primiti nove podatke jer mu je prijemna međumemorija ispunjena ili je sustav preopterećen. |
| **`TcpExtTCPToZeroWindowAdv`** | Ovaj brojač bilježi broj puta kada TCP pošiljatelj oglašava (*advertises*) svoj prozor veličine nula (*zero window*) prema TCP primatelju. To se obično događa kada TCP pošiljatelj ima podatke za slanje, ali primatelj nije spreman primiti nove podatke zbog prozora veličine nula. |
| **`TcpExtTCPWantZeroWindowAdv`** | Ovaj brojač bilježi broj puta kada TCP primatelj želi oglašavati (*advertise*) prozor nula veličine (*zero window*) prema TCP pošiljatelju, ali to nije moguće zbog nekih ograničenja ili problema na strani primatelja. |
| **`TcpExtTCPZeroWindowDrop`** | Ovaj brojač bilježi broj TCP paketa koji su odbačeni zbog toga što je primatelj oglašavao prozor veličine nula (*zero window*). Kada primatelj oglašava prozor veličine nula, to znači da trenutno nije spreman primiti nove podatke od pošiljatelja jer mu je prijemna međumemorija ispunjena ili je sustav preopterećen. |

**Izvori informacija: (264),(265),(266), (732),(1406),(1477),(1651),(1770),(1771),(K-6), `man 7 tcp`,`man nstat`, RFC 1323, RFC 4253.**

## 24.2.8.1. Veza između latencije, propusnosti i TCP Window size parametra

Iako kašnjenje (latencija), propusnost i veličina TCP prozora naizgled nisu povezani, veza između njih je vrlo opipljiva. Naime na kašnjenje odnosno latenciju ne možemo puno utjecati zbog zakona fizike. Jedino na što bi eventualno mogli utjecati je kašnjenje uzrokovano sporijom mrežnom opremom od strane telekoma (prema internetu), ali u praksi su razlike prilično male. U svakom slučaju važno je znati kako postoje određena ograničenja koja nisu toliko očita. Pogledajmo o čemu se radi.
Upotrebom TCP/IP protokola, odnosno konkretnije TCP protokola kao transportnog protokola svi današnji operativni sustavi koriste funkcionalnost TCP prozora (*TCP Window)* za povećanje propusnosti.
Standardna veličina ovog TCP prozora je obično 64kB.
Prisjetimo se da je:

- 64kB = 65.535 *bajta* = 524.280 bitova.
- 1Gbps = 1.000.000.000 bitova u sekundi.

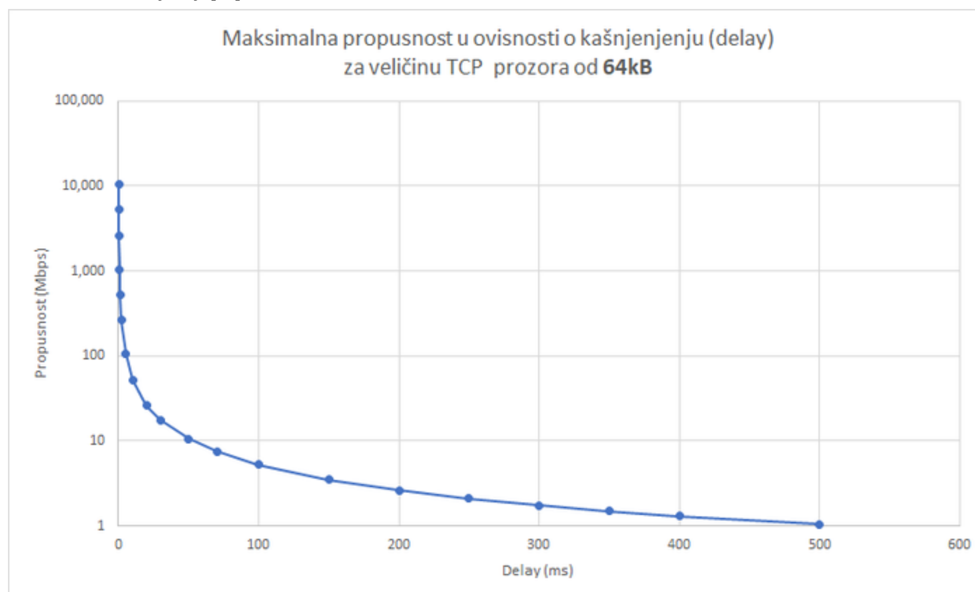
$$\boldsymbol{Maksimalno\ dozvoljeno\ kašnjenje\ (latencija) = \frac{TCP\ Window\ size}{Širina\ pojasa\ mreže}}$$

Prema tome za 1Gbps vezu možemo reći sljedeće:

$$\boldsymbol{Maksimalna\ dozvoljena\ latencija = \frac{524.280}{1\,000.000.000} = 0,00052428\ sekundi = 524\ mikro\ sekunde}\ (\mu s)$$

To znači kako za mrežu koja radi na 1Gbps, te se u komunikaciji oslanja na *TCP* protokol, a koji koristi 64kB veličinu *TCP prozora*, kako bi se postigla puna propusnost mreže, kašnjenje između dvaju strana u komunikaciji ne smije biti veće od 524 mikro sekunde ($\mu s$) odnosno 0,524 mili sekunde (*ms*).
Kod 10Gbps mreža vrijeme je deset puta manje, dakle 52,4 mikro sekunde ($\mu s$) odnosno 0,0524 mili sekunde (*ms*).
Sve ovo isto tako znači i kako se povećanjem kašnjenja (*latencije*) uz istu veličinu *TCP prozora*, smanjuje i efektivna propusnost mreže. Dakle vrlo udaljene veze, između gradova ili država i kontinenata zbog fizike medija i rasprostiranja signala kroz njega, kao i mrežne opreme, unose kašnjenja signala koja utječu na efektivnu propusnost mreže.
Jedino na što možemo utjecati je veličina *TCP prozora*. Pogledajmo kako bi izgledala propusnost mreže ako bi *TCP prozor* ostao 64kB (65.535 bajta), a mijenjalo bi se kašnjenje (*delay/latencija*) kako je prikazano na slici 210.

**Slika 210. Rezultati mjerenja propusnosti.**




Jasno je vidljivo kako se povećanjem kašnjenja (latencije) odnosno *delaya*, uz istu veličinu *TCP prozora* drastično smanjuje stvarna propusnost mreže.

Pogledajmo i drugi primjer izračuna ispravne veličine *TCP prozora*, ovisno o kašnjenju (*delay*), kako bi se maksimalno iskoristila dostupna propusnost odnosno ono što nazivamo *brzinom* mreže.

U ovom drugom primjeru zamislimo da imamo tvrtku s dva ureda, jedan u Zagrebu, a drugi u Osijeku te unajmljenu međuvezu propusnosti 1Gbps. Pri tome je:

- Latencija (kašnjenje) između naših usmjerivača u Zagrebu i Osijeku oko 22ms.

Osnovna formula nam govori:

$$\boldsymbol{Veličina\ TCP\ prozora = Širina\ pojasa\ mreže\ \times kašnjenje\ (latencija)}$$

Prema tome je za navedeni primjer:
**Veličina TCP prozora = 1.000.000.000 bps x 0,022 sekunde = 22.000.000 bitova**

Kako bismo sve dobili u bajtima umjesto u bitovima, podijelit ćemo ovu vrijednost s osam:

$$\boldsymbol{Veličina\ TCP\ prozora = \frac{22.000.000\ bitova}{8} = 2.750.000\ bajta}$$

Odnosno kako bismo sve dobili u megabajtima (MB): $\boldsymbol{Veličina\ TCP\ prozora = \frac{2.750.000\ bajta}{1.000.000} = 2,75\ MB}$

Ovdje smo izračunali kako je za komunikaciju preko ovakve udaljene veze potrebna veličina TCP prozora od: *2.750.000 bajta*, podijelimo s 1.000, kako bismo dobili kB, te to sve podijelimo ponovno s 1.000, kako bismo dobili vrijednost u MB, te konačno dobivamo **2,75 MB** što naravno uopće mora podržavati operativni sustav, jer ova veličina nadilazi osnovni standard (definirana je tek u proširenju **RFC 1323**). Dodatno potrebno je i poigrati se s drugačijim načinom slanja *TCP ACK* poruka, koji je efikasniji za ovakve ogromne veličine TCP prozora. Dakle moramo imati i podršku za Tzv. ***TCP selective acknowledgement*** (*SACK*) odnosno selektivno TCP potvrđivanje koje može selektivno potvrđivati pristigle pakete unutar *TCP prozora*. Tako da se u slučaju greške ne bi morala slati cijela gomila paketa koji su ispravni i sâmo nekoliko onih neispravnih, već se upotrebom ove *TCP* opcije onda zatraže samo neispravni paketi unutar TCP prozora.
Ova mogućnost je definirana u **RFC 2018**. U ovu priču dodatno dolazi i ***window scaling*** mehanizam, jer kako smo ranije govorili, maksimalna standardna veličina *TCP* prozora (*TCP window*) je 16 bitni broj ($2^{16}$) što znači 65.535 bajta odnosno 64 kB, što je u ovom primjeru nedovoljno. Srećom pa imamo proširenje ove veličine, o kojem ćemo govoriti u sljedećoj cjelini.

**Pogled s druge strane**
Kako smo već govorili postoji jasna poveznica između latencije, veličine TCP prozora i propusnosti mreže.
Pogledajmo malo matematike, za već navedenu standardnu veličinu *TCP prozora* od 64kB te latenciju, koju ćemo zaokružiti na 20ms. Dakle ovdje ćemo promatrati utjecaj latencije i veličine *TCP prozora*, na maksimalnu moguću propusnost mreže koju možemo dobiti ako se koristi *TCP* kao transportni protokol. U našem slučaju imamo sljedeće stanje: 64 kB = 65.535 *bajta* x 8 = 524.288 bitova. Pogledajmo kolika će biti maksimalna propusnost ove mreže ako se koristi TCP protokol bez obzira kolika je propusnost koju smo dobili:

$$\boldsymbol{Maksimalna\ propusnost\ mreže} = \frac{\boldsymbol{TCP\ window\ size\ (bitova)}}{\boldsymbol{RTT\ (sekundi)}} = \frac{\mathbf{524.288}}{\mathbf{0.02}} = \mathbf{26.214.400} = \frac{\mathbf{26.214.400}}{\mathbf{1.000.000}} = \mathbf{26,21}\ \boldsymbol{Mbps}$$

Dakle vidimo kako je maksimalna moguća propusnost upotrebom *TCP* protokola koji koristi veličinu *TCP* prozora od 64kB uz latenciju od 20ms (0,02 sekunde) samo **26,21** Mbps, bez obzira kolika je propusnost međuveze između krajnjih točaka u komunikaciji. Dakle uz ovu latenciju (kašnjenje) i veličinu *TCP prozora* možemo imati i **100 Gbps** međuvezu, ali nećemo imati veću propusnost od samo **26,21 Mbps**.
S druge strane nepotrebnim povećanjem *TCP prozora* izvan izračunatih okvira dobiti ćemo suprotan efekt, odnosno usporavanje veze. Stoga treba biti oprezan, a ako niste sigurni što radite najbolje je ništa ne dirati.

**Izvori informacija: (K-6), RFC 1323 i RFC 2018.**

## 24.2.8.2. Skaliranje TCP prozora (*Window Scaling*)

Postoje i slučajevi u kojima upotrebom *TCP prozora* ne možemo dobiti potrebnu propusnost mreže odnosno dolazimo do granica u kojima ta mreža nije optimalno iskorištena. Problem je u tome što kod udaljenih veza kao i onih koje imaju veće kašnjenje signala (satelitske, neke kablovske i druge veze) maksimalna veličina *TCP prozora* od 65.535 bajta odnosno 64 kB nije dovoljna.
Mehanizam skaliranja prozora (*Window scaling*) uveden je prvo u RFC 1072 te proširen u RFC 1323.
U osnovi skaliranje prozora jednostavno proširuje 16-bitno polje prozora na 32 bita. Naravno nemoguće je sâmo umetnuti dodatnih 16 bitova u *TCP* zaglavlje što bi ga učinilo potpuno nespojivim s postojećim implementacijama TCP protokola.
Rješenje je definiranje nove *TCP* opcije za određivanje broja pomoću kojeg bi se polje za zaglavlje *TCP*-a trebalo pomaknuti (Engl. *logical bitwise shift*) kako bi se dobila ova veća vrijednost. Ova nova opcija u TCP zaglavlju se zove: `tcp.window_size_scalefactor`. Ona predstavlja proširenje veličine TCP prozora.
Naime ova vrijednost govori na koju potenciju broja dva ($2^n$) proširujemo veličinu standardnog *TCP prozora*.
Međutim implementacija je takva da je iskorišteno 14 bîtova jer su zadnja dva bîta rezervirana:

| `tcp.window_size_scalefactor` | **Faktor povećanja (n x veličina TCP prozora)** |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 |
| 11 | 2048 |
| 12 | 4096 |
| 13 | 8192 |
| 14 | 16384 |

To znači da, ako je vrijednost `tcp.window_size_scalefactor` postavljena na **1** da se omogućava povećanje klasičnog *TCP prozora* dva (2) puta, odnosno ako je postavljena na **2** kako se omogućava povećanje za četiri (4) puta i tako dalje.
Dakle povećanje veličine TCP prozora je: $\mathbf{2^n}$ puta.

S ovom tehnologijom smo dobili maksimalno povećanje do 16.384 puta ($2^{14}$) na postojeću veličinu *TCP prozora* odnosno:

- 65.535 x 16.384 = 1.073.725.440 odnosno $2^{30}$ ili drugim riječima: $2^{16}$ x $2^{14}$ = $2^{30}$.

Ova opcija skaliranja TCP prozora (*Window Scaling*) šalje se u inicijalnom *TCP* paketu (u `TCP SYN` poruci) tijekom ostvarivanja nove *TCP* konekcije, te traje tijekom cijelog života određene *TCP* konekcije, naravno ***sâmo i ako*** ju obje strane podržavaju. Srećom sve moderne implementacije TCP/IP protokola automatski dogovaraju obje vrijednosti:

- Veličinu TCP prozora (***TCP window size***).
- Vrijednost skaliranja TCP prozora (***TCP scale factor***).

Pogledajmo gdje je u Linuxu definirana upotreba *TCP window scaling* mehanizma:

| /proc | sysctl | Standardna vrijednost |
|---|---|---|
| /proc/sys/net/ipv4/tcp_window_scaling | net.ipv4.tcp_window_scaling | 1 |

Standardno je u Linuxu uključen *TCP window scaling*, a možemo ga (u posebnim slučajevima) i isključiti, postavljanjem ove varijable na nulu (0). Dodatno vezano za prijašnju vrijednost, potrebno je povećati i maksimalnu veličinu memorijskog *buffera* u i iz kojeg se šalju odnosno primaju paketi sa i na mrežu. Dakle govorimo o međumemorijama zaduženim za primanje i slanje paketa na mrežu odnosno *Receive* (***rmem***) i *Send* (***wmem***) *bufferu*.

Ovdje (***rmem*** i ***wmem***) se radi o maksimalnoj mogućoj vrijednosti *TCP (buffera)* međumemorije koja se može koristiti na razini cijelog sustava. Pogledajmo gdje se oni mogu definirati odnosno mijenjati u Linuxu:

| /proc varijabla | sysctl varijabla | Standardna vrijednost | Opis |
|---|---|---|---|
| /proc/sys/net/core/rmem_max | net.core.rmem_max | 16777216 | *Receive buffer* |
| /proc/sys/net/core/wmem_max | net.core.wmem_max | 11796480 | *Send buffer* |

Ovdje se radi o maksimalnim mogućim vrijednostima, a ne onima koje se trenutno koriste.
Postoji i donja granica ispod koje Linux neće ići kod alociranja i/ili dogovaranja *TCP prozora* te standardna i vršna vrijednost. Pri tome se vršne vrijednosti čitaju iz gore navedenih varijabli.

Pogledajmo i varijablu i njenu vrijednosti za međumemoriju za **primanje** paketa; **RMEM** (*Receive*):

| /proc varijabla | sysctl varijabla | Vrijednost 1 | Vrijednost 2 | Vrijednost 3 |
|---|---|---|---|---|
| /proc/sys/net/ipv4/tcp_rmem | net.ipv4.tcp_rmem | 4096 | 131072 | 11796480 |

Opis polja:

- Vrijednost **1**: *4.096* je minimalna vrijednost memorije za primanje (*receive window buffera)* koja će se postaviti za svaku novu *TCP* konekciju.
- Vrijednost **2**: *131.072* je standardna vrijednost memorije za primanje (*receive window buffera)* koja će se postaviti za svaku novu *TCP* konekciju.
- Vrijednost **3**: *11.796.480* je maksimalna vrijednost memorije za primanje (*receive window buffera)* koja se može postaviti za bilo koju *TCP* konekciju.

Međutim, prostor međuspremnika za primanje podataka s mreže dijeli se između aplikacije i kernela. Kernel dio za TCP koristi dio međuspremnika za veličinu TCP prozora; ovo je veličina prozora primanja (`net.ipv4.tcp_rmem`) koja se i inače oglašava drugoj strani u komunikaciji. Naime, ranije spomenuta postavka koja se zove `net.ipv4.tcp_rmem` sastoji se od tri vrijednosti, a sada govorimo o njenoj trećoj vrijednosti koja definira ukupnu veličinu ove međumemorije.
Ostatak međuspremnika (memorije) koristi se kao aplikacijski međuspremnik. Za kontrolu odnosa, koliko postotaka ove međumemorije se može koristiti kao međumemorija za aplikacije, postoji nova varijabla imena: `net.ipv4.tcp_adv_win_scale`.

Primjerice zadana vrijednost `net.ipv4.tcp_adv_win_scale` od **1** podrazumijeva da je prostor koji se koristi kao dio međuspremnika za ***aplikacije***, označava polovinu ukupno dostupnog međuspremnika. Pogledajmo opis drugih vrijednosti:

| Sysctl varijabla | Vrijednost = postotak memorije | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| net.ipv4.tcp_adv_win_scale | **4** = $\frac{15}{16}$ | **3** = $\frac{7}{8}$ | **2** = $\frac{3}{4}$ | **1** = $\frac{1}{2}$ | **0** | **-1** = $\frac{1}{2}$ | **-2** = $\frac{1}{4}$ | **-3** = $\frac{1}{8}$ |

Izračun je: *postotak međuspremnika za aplikacije* = `tcp_rmem` - $\left(\frac{tcp_rmem}{2^{adv_win_scale}}\right)$.



U slučajevima kada može doći do problema u propusnosti mreže ili drugih problema, pogledajte i poglavlje: **25.7.4. *Naredba netstat*** → pogledajte napredni primjer, nakon primjera 2. (obratite pažnju na ***Recv-Q*** i ***Send-Q***).

Zatim pogledajmo i stanje za međumemoriju za **slanje** paketa odnosno **WMEM** (*Send*) međumemoriju:

| /proc varijabla | sysctl varijabla | Vrijednost 1 | Vrijednost 2 | Vrijednost 3 |
|---|---|---|---|---|
| /proc/sys/net/ipv4/tcp_wmem | net.ipv4.tcp_wmem | 4096 | 16384 | 11796480 |

Opis polja:

- Vrijednost **1**: *4.096* je minimalna vrijednost memorije za slanje (*send window buffer*) koja će se postaviti za svaku novu *TCP* konekciju.
- Vrijednost **2**: *16.384* je standardna vrijednost memorije za slanje *(send window buffera)* koja će se postaviti za svaku novu *TCP* konekciju.
- Vrijednost **3**: *11.796.480* je maksimalna vrijednost memorije za slanje (*send window buffera)* koja se može postaviti za bilo koju *TCP* konekciju.

**Vratimo se na prvotni izračun kašnjenja/latencije (**Engl. *network delay/latency***) ovisno o brzini mreže.**
Kako smo izračunali, ovo su maksimalna kašnjenja odnosno *latencije* prema brzini mreže za koje je standardna veličina *TCP prozora* (64 kB) zadovoljavajuća:

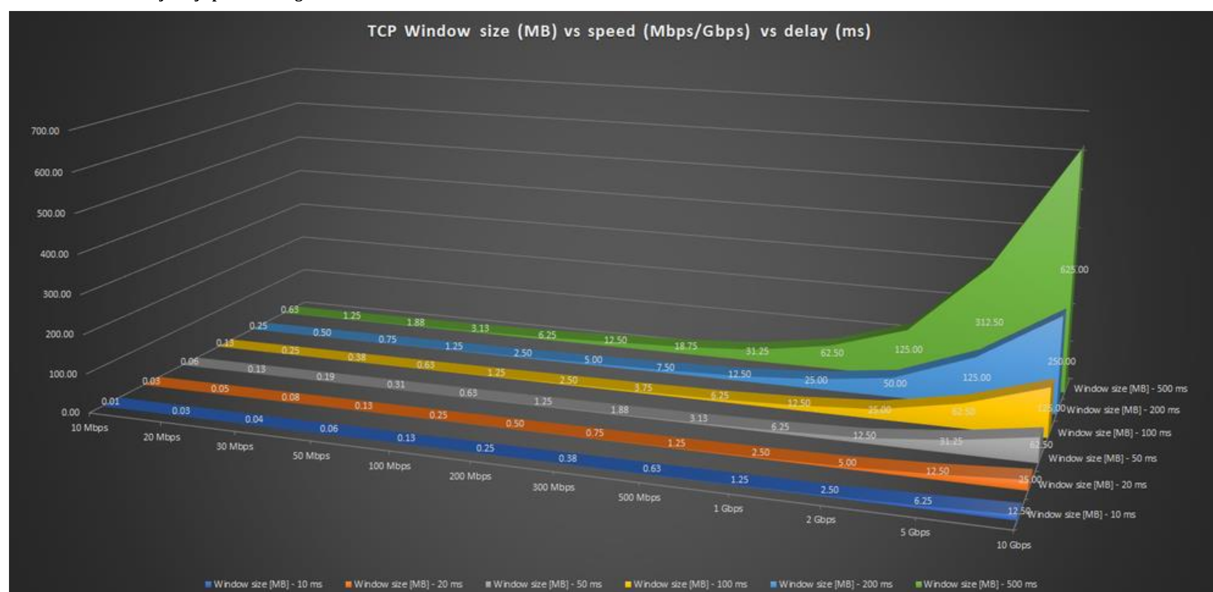
| Brzina mreže | Maksimalna dozvoljena latencija (kašnjenje) |
|---|---|
| 10 Mbps | 52.428 µs (52,42 mili sekunde (ms)) |
| 100 Mbps | 5.242 µs (5,24 mili sekundi (ms)) |
| 1 Gbps | 524 µs (0,524 mili sekunde (ms)) |
| 10 Gbps | 52,4 µs (0,0524 mili sekunde (ms)) |
| 20 Gbps | 26,2 µs (mikro sekundi) |
| 40 Gbps | 13,1 µs (mikro sekundi) |
| 50 Gbps | 10,48 µs (mikro sekundi) |
| 100 Gbps | 5,24 µs (mikro sekundi) |

Pogledajmo i koliki bi trebao biti *TCP prozor* za dolje definirane brzine, ovisno o kašnjenju odnosno latenciji mreže:

| Brzina mreže (Mbps) | Veličina TCP prozora za 10 ms kašnjenje | Veličina TCP prozora za 20 ms kašnjenje | Veličina TCP prozora za 50 ms kašnjenje | Veličina TCP prozora za 100 ms kašnjenje |
|---|---|---|---|---|
| 10 Mbps | 13 kB | 25 kB | 63 kB | 125 kB |
| 20 Mbps | 25 kB | 50 kB | 125 kB | 250 kB |
| 30 Mbps | 38 kB | 75 kB | 188 kB | 375 kB |
| 50 Mbps | 63 kB | 125 kB | 313 kB | 625 kB |
| 100 Mbps | 125 kB | 250 kB | 625 kB | 1,25 MB |
| 200 Mbps | 250 kB | 500 kB | 1,25 MB | 2,5 MB |
| 300 Mbps | 375 kB | 750 kB | 1,88 MB | 3,75 MB |
| 500 Mbps | 625 kB | 1,25 MB | 3,13 MB | 6,25 MB |
| 1 Gbps | 1,25 MB | 2,5 MB | 6,25 MB | 12,5 MB |
| 2 Gbps | 2,5 MB | 5 MB | 12,5 MB | 25 MB |
| 5 Gbps | 6,25 MB | 12,5 MB | 31,25 MB | 62,5 MB |
| 10 Gbps | 12,5 MB | 25 MB | 62,5 MB | 125 MB |

Pogledajmo i usporedni grafikon koji pokazuje ovisnost:
**Brzina mreže - kašnjenje/latencija (***delay***) - veličina TCP prozora (***TCP Window***).** Grafikon smo proširili s kašnjenjima tako da su sada pokrivena vremena od 10ms sve do 500 ms, kako je sve zajedno vidljivo na slici 211.

**Slika 211. Rezultati mjerenja prikazani u grafikonu.**




Iz grafikona na slici 211 je vidljivo koliko bi se trebao povećati *TCP prozor* ovisno o kašnjenju odnosno latenciji (*delay*-u) i brzini mrežne međuveze.

**Izvori informacija: (267),(268),(269),(270),(271),(272),(1406),(K-6),** `man 7 tcp`**, RFC 1072 i RFC 1323.**

## Gubitak podataka

U svakoj mreži dolazi do gubitka podataka koje *TCP* uspješno rješava ponovnim slanjem (retransmisijom) izgubljenih paketa. Bez obzira koristi li se standardni *TCP prozor* od 64kB ili prošireni upotrebom W*indow scaling* mehanizma (veći od 64kB) u trenutku kada primatelj ne primi određeni niz paketa unutar *TCP* prozora, odnosno ne potvrdi ih pošiljatelju, pošiljatelj ih mora ponovno poslati. Naime ovime se osim mehanizma za retransmisiju bavi i *TCP* mehanizam za zagušenje (Engl. *congestion*). Dakle i ovaj mehanizam *TCP*-a bavi se problemima kada određeni paket ili nîz paketa poslanih unutar *TCP prozora*:

- Nije potvrđen od strane primatelja.
- Ili je isteklo vrijeme unutar kojega je potvrda morala biti poslana (odnosno zaprimljena).

U oba slučaja paket se smatra izgubljenim, a prozor *TCP zagušenja* (Engl. *TCP congestion window*) se smanjuje za polovicu, a sâmim time u tom trenutku se smanjuje i efektivna propusnost mreže.
Za sada ćemo reći samo toliko o mehanizmu detekcije zagušenja, jer ćemo o njemu govoriti kasnije.
U svakom slučaju gubitak paketa imat će dva učinka na brzinu prijenosa podataka:

- Paketi će morati biti ponovno poslani, čak i ako je samo paket potvrde (*TCP ACK*) bio izgubljen, a paketi su bili isporučeni.
- Veličina prozora TCP zagušenja (*TCP congestion window*) neće dopustiti optimalan protok podataka kroz mrežu.

Kod povremenog gubitka podataka, veza se usporava te se s vremenom ponovno ubrzava.
U slučajevima kada postoje konstantni gubici na mreži doći će do znatnijih usporavanja koja će postati konstantnija; ovisno o tome koliko su i gubici konstantni. Naravno što je veća latencija mreže to će propusnost u svakom slučaju biti manja, povećanjem gubitaka paketa na mreži. Naime već kod konstantnog gubitka paketa koji u postotku možda izgledaju kao mali (primjerice 2%) to u praksi izgleda drastično loše, a pogotovo ako se povećava i kašnjenje odnosno latencija.
Pogledajmo i koliko je to loše odnosno kako se mijenja propusnost mreže u odnosu na povećanje gubitaka paketa na mreži:

| **Kašnjenje (latencija/delay)** | **TCP propusnost bez gubitaka** | **TCP propusnost uz stalni gubitak od 2%** |
|---|---|---|
| 0 ms | 93,5 Mbps | 3,72 Mbps |
| 30 ms | 16,20 Mbps | 1,63 Mbps |
| 60 ms | 8,07 Mbps | 1,33 Mbps |
| 90 ms | 5,32 Mbps | 0,85 Mbps |

Izvor tablice/mjerenja: **(1023)** → http://smutz.us/techtips/NetworkLatency.html

**Izvori informacija: (325),(1023),(K-6), `man 7 tcp`.**

## 24.2.9. Nadzor zagušenja (*Congestion control*)

*TCP Congestion control* znan i kao mehanizam nadzora zagušenja odnosno zaštite od gomilanja prometa, osigurava nas od zagušenja klijenta odnosno krajnjeg računala koje prima mrežni promet (pakete). Dodatno ovaj mehanizam se uključuje i nakon detekcije problema u kojima paketi nisu dostavljeni primatelju iz bilo kojeg od dva razloga:

- Primatelj ih nije potvrdio s porukom `TCP ACK`.
- Istekao je vremenski okvir u kojem ih je primatelj morao potvrditi.

Ovaj mehanizam definiran je prvi puta u standardu RFC 2581 koji je nadograđen s novim standardom RFC 5681.
Ovdje se zapravo radi o nekoliko osnovnih mehanizama koji nam daju ovu funkcionalnost.

*Transmission Control Protocol* (**TCP**) može koristiti sâmo jedan od algoritama za izbjegavanje zagušenja mreže.
Navedenih algoritama postoji veći broj, od kojih svaki operativni sustav preferira samo jedan od njih. Tako Linux standardno koristi algoritam **`cubic`** u kojem su implementirani navedeni mehanizmi, ali na njemu specifičan način.
Dakle svaki od algoritama ima navedene mehanizme implementirane na specifičan način pa se stoga svaki od njih ponaša malo drugačije na različitim vrstama veza:

- Brzim LAN mrežama, s vrlo malim kašnjenjem (latencijom).
- Brzim WAN mrežama s većim ili velikim kašnjenjem (latencijom).
- Sporijim WAN mrežama s većim ili velikim kašnjenjem (latencijom).
- Raznim vezama s manjim ili većim postotkom gubitaka paketa i slično.

Kako bi se izbjeglo zagušenje, *TCP* koristi strategiju kontrole zagušenja (*congestion control*). Za svaku vezu *TCP* održava takozvani prozor za zagušenje ograničavajući ukupan broj nepotvrđenih paketa koji mogu biti u tranzitu. To je analogno mehanizmu *TCP prozora* koji se koristi za kontrolu protoka, ali u ovom slučaju za potrebu nadzora zagušenja (*congestion control*).



**Mehanizam nadzora zagušenja se aktivira na strani pošiljatelja, a ne od strane primatelja!**

*TCP* koristi mehanizam nazvan usporeni početak (Engl. ***slow start***) za povećanje prozora zagušenja, nakon inicijalizacije veze ili nakon vremenskog ograničenja. Naime kada se detektira zagušenje, prozor zagušenja (*congestion window)* se namješta na veličinu do dva puta maksimalne veličine segmenta (**MSS**). Ovaj prozor zagušenja je zapravo nama već poznati *TCP prozor*, ali koji sada inicijalno mijenja pošiljatelj, a ne primatelj kao što je slučaj s klasičnim TCP prozorom.
Promjene se rade na osnovi njegovih izračuna te praćenja prometa kao i trenutnog kašnjenja odnosno latencije u mreži.
Iako je početna vrijednost prozora za zagušenja u ovom trenutku niska, stopa porasta je vrlo brza; za svaki potvrđeni paket prozor za zagušenja povećava se za **1 *MSS***, tako da se prozor za zagušenje učinkovito udvostruči za svako prolazno vrijeme kašnjenja (latenciju) odnosno **RTT** *vrijeme*.

Kada prozor zagušenja prelazi prag polaganog pokretanja (Tzv. ***slow start***) algoritam ulazi u novo stanje nazvano izbjegavanje zagušenja (Engl. ***congestion avoidance***). U ovom stanju sve dok se ne pojave duplicirane *ACK* poruke, prozor za zagušenje dodatno se povećava za jedan *MSS*, svaki puta za svako prolazno vrijeme kašnjenja (latenciju) odnosno ***RTT*** *vrijeme*. O ovim stanjima i mehanizmima, govorit ćemo kasnije. Algoritmi detekcije i kasnijeg reagiranja na zagušenje se zovu ***AIMD*** (Engl. *Additive-Increase/Multiplicative-Decrease*) zbog njihovog načina rada.

Naime svi ovi algoritmi kod detekcije zagušenja prvo rade:

- *Multiplicative decrease* odnosno smanjivanje TCP prozora, koji ovdje, odnosno u ovom slučaju nazivamo prozor zagušenja, i to obično na pola trenutne vrijednosti, mada točan iznos smanjivanja ovisi o sâmom algoritmu.
- A potom *additive increase* odnosno postepeno povećanje prozora zagušenja, čiji faktor povećanja opet ovisi o sâmom algoritmu.

Pogledajmo koji algoritam je trenutno u upotrebi na našem Linuxu, pomoću naredbe `sysctl` i iščitavanja varijable (`net.ipv4.tcp_congestion_control`) koja nam daje ovu vrijednost. Dakle pokrenimo sljedeću naredbu:

```
sysctl net.ipv4.tcp_congestion_control
```

Dobivamo odgovor iz kojeg vidimo kako se na našem računalu (Linux) koristi `cubic` algoritam:

```
net.ipv4.tcp_congestion_control = cubic
```

Na istom mjestu odnosno s istom metodom možemo i permanentno (trajno) promijeniti standardni algoritam koji će se koristiti za cijeli sustav. Dakle u konfiguracijsku datoteku: `/etc/sysctl.conf` dodajmo varijablu za čiju vrijednost postavljamo algoritam koji želimo (u primjeru je to **`reno`** algoritam), a koji dodajemo dodavanjem sljedećeg retka teksta:

```
net.ipv4.tcp_congestion_control=reno
```

Naravno algoritam koji smo postavili mora biti dostupan odnosno njegov kernel modul učitan tijekom pokretanja sustava. Pogledajmo i koji su sve algoritmi *kompilirani* za naš trenutni linux kernel, a koji dolaze kao kernel moduli:

```
cat /boot/config-`uname -r` | grep CONFIG_TCP_CONG

CONFIG_TCP_CONG_ADVANCED=y
CONFIG_TCP_CONG_BIC=m
CONFIG_TCP_CONG_CUBIC=y
CONFIG_TCP_CONG_WESTWOOD=m
CONFIG_TCP_CONG_HTCP=m
CONFIG_TCP_CONG_HSTCP=m
CONFIG_TCP_CONG_HYBLA=m
CONFIG_TCP_CONG_VEGAS=m
CONFIG_TCP_CONG_SCALABLE=m
CONFIG_TCP_CONG_LP=m
CONFIG_TCP_CONG_VENO=m
CONFIG_TCP_CONG_YEAH=m
CONFIG_TCP_CONG_ILLINOIS=m
CONFIG_TCP_CONG_DCTCP=m
```

Opis ispisa koji smo dobili:

- Svi kernel moduli koji maju oznaku `=y` su već dostupni odnosno nalaze se unutar kernela.
- Svi kernel moduli koji maju oznaku `=m` su dostupni samo kao kernel moduli i prethodno ih je potrebno učitati sa `insmod` ili `modprobe`. Informacije o njima možemo dobiti s naredbom `modinfo`.

Konkretni kernel moduli se nalaze u standardnom direktoriju za kernel module trenutno aktivnog kernela.
Za naš trenutno aktivni kernel, možemo ih izlistati sa sljedećom naredbom:

```
ls -al /lib/modules/`uname -r`/kernel/net/ipv4/tcp_*
```

Ako pak želimo vidjeti koji su algoritmi trenutno učitani i odmah dostupni (za sve korisnike), to možemo vidjeti s naredbom:

```
cat /proc/sys/net/ipv4/tcp_available_congestion_control
```

Ako smo *root* korisnik, naravno možemo učitati i druge algoritme i koristiti ih, ali sâmo one koji su kako smo vidjeli gore, kompilirani s našim aktivnim kernelom.

Objasnit ćemo neke od ovih algoritama, prema nazivima njihovih kernel modula (uz naziv algoritma):

- `tcp_bic` (***bic***) - *BIC* je kratica za *Binary Increase Congestion control*. *BIC* koristi jedinstvenu funkciju rasta TCP prozora. U slučaju gubitka paketa, TCP prozor se smanjuje multiplikativnim faktorom. Veličina TCP prozora prije i poslije redukcije se zatim koristi kao parametar za binarnu pretragu za novu veličinu prozora. *BIC* je prije korišten kao standardni algoritam u Linux kernelu.
- `tcp_bbr` (***BBR***) - *Bottleneck Bandwidth and Round-trip propagation time* - moderni je algoritam za kontrolu i izbjegavanje zagušenja, dostupan je od kernela 4.9. od 2016.g. a noviji i optimiziraniji BBR v.2.od 2020.g. Dok se većina algoritama temelji na gubicima (oni se oslanjaju na gubitak paketa za otkrivanje zagušenja i niže brzine prijenosa), BBR se temelji na mehanizmu korištenja maksimalne širine pojasa i vremena prolaska paketa (*round-trip*) za izgradnju modela sposobnosti mreže na temelju **BDP** (izračun kašnjenja i propusnosti). Standardno ga koriste *Google* i *Youtube* kao zadani algoritam.
- `cubic` (***cubic***) - on se već nalazi u našem kernelu (`y` vrijednost u konfiguraciji kernela) i u svim novijim kernelima, kao standardan. *CUBIC* je manje agresivna varijanta *BIC*a što znači kako ne uzima toliko propusnosti od paralelnih *TCP* tokova kao *BIC*. Od 2011.g. u ovaj algoritam ugrađeni su i mehanizmi *HyStart* algoritma, koji je riješio neke od njegovih prijašnjih slabosti. Ovaj algoritam prati gubitke paketa u mreži i na osnovu njih se prilagođava. Dakle on je tzv. *loss-based* algoritam i često je upravo on zadani algoritam na Linuxu.
- `tcp_dctcp` - ***Datacenter TCP*** (*DCTCP*) je poboljšanje algoritma za kontrolu zagušenja *TCP*-a koji iskorištava eksplicitnu obavijest zagušenja (*ECN*) u mrežama podatkovnih centara, kako bi se pružila višekratna povratna informacija krajnjim točkama u komunikaciji. Svi preklopnici na mreži podatkovnih centara koji koriste *DCTCP* moraju podržavati *ECN* obilježavanje paketa i biti konfigurirani za obilježavanje pri dostizanju određenog praga međuspremnika (*buffera*). Informacije: https://www.kernel.org/doc/Documentation/networking/dctcp.txt
- `tcp_htcp` - (***htcp***). *H-TCP* je predložio institut *Hamilton*, za transmisije koje se brže oporavljaju nakon događaja zagušenja. Također je dizajniran za veze s visokom propusnošću i ***RTT*** vremenima odziva.
- `tcp_highspeed` - (***highspeed***). *High speed* algoritam je opisan u RFC 3649. Glavna upotreba je u vezama s velikom propusnošću i velikim ***RTT*** vremenima, kao što je propusnost od barem 1 Gbit uz 100 ms *RTT vrijeme*.
- `tcp_hybla` - (***hybla***). *TCP Hybla* se koristi kako bi učinkovito prenosio podatke preko satelitskih veza i "obranio" prijenos *TCP* tokova iz drugih paralelnih izvora.
- `tcp_illinois` - (***illinois***). Ovaj algoritam razvijen je za vrlo brze mreže s velikim latencijama. Kao okidač koristi gubitak paketa i kašnjenje (latenciju). Za više detalja pogledajte: https://en.wikipedia.org/wiki/TCP-Illinois.
- `tcp_lp` -(*lp*). ***TCP Low priority*** je algoritam koji koristi višak propusnosti za nove TCP tokove prijenosa. Može se koristiti za prijenose podataka s niskim prioritetom bez "uznemiravanja" drugih TCP prijenosa, a koji vjerojatno ne koriste *TCP Low Priority* algoritam.
- `tcp_scalable` (***scalable***). *TCP scalable* je još jedan algoritam za *WAN* veze s visokom širinom pojasa i ***RTT*** vremenima/latencijom. Jedan od njegovih dizajnerskih ciljeva je brz oporavak veličine TCP prozora nakon događaja zagušenja. To postiže tako da vraća prozor na višu vrijednost od standardnog *TCP*-a.
- `reno` (***reno***). *TCP Tahoe/Reno* - ovo su klasični modeli koji se koriste za kontrolu zagušenja. Oni rade s tipično sporim početkom prijenosa. Propusnost se povećava postupno dok ne ostane stabilna. S druge strane smanjuje se, čim u prijenosu dođe do zagušenja, a potom se stopa polako diže. TCP prozor se povećava dodavanjem fiksnih vrijednosti. *TCP Reno* koristi algoritam multiplikativnog smanjenja za smanjenje veličine prozora. *TCP Reno* je najčešće korišten algoritam i u drugim operativnim sustavima.
- `tcp_vegas` - (***vegas***). *TCP Vegas* uvodi mjerenje ***RTT*** vremena za procjenu kvalitete veze. Koristi povećanje aditiva i smanjenje aditiva u opciji prozora zagušenja (*congestion window*-a).
- `tcp_veno` - (***veno***). *TCP Veno* je varijanta optimizirana za bežične mreže, jer je dizajnirana za rješavanje slučajnog gubitka paketa. Pokušava pratiti prijenos i pogađa ako se kvaliteta smanjuje zbog zagušenja ili slučajnih pogrešaka u paketu.
- `tcp_westwood` - (***westwood***). *TCP Westwood* (+) radi dobro s velikim propusnostima i ***RTT*** vrijednostima, kao i slučajnim gubitkom paketa, zajedno s dinamičkim promjenama mrežnih opterećenja. Analizira stanje prijenosa pregledavanjem *TCP ACK* (potvrdnih) paketa. *Westwood +* je modifikacija *TCP Reno* algoritma.
- `tcp_yeah` - (***yeah***). *YeAH TCP* je algoritam za kontrolu zagušenja koji je svjestan latencije i gubitka paketa, što znači kako ovaj algoritam prati gubitke paketa u mreži, ali i latenciju i na osnovu njih se prilagođava. Dakle on je *loss-delay-based* algoritam. Kroz mjerenje kašnjenja i gubitaka kada je mreža neopterećena, brzo će iskoristiti raspoloživi kapacitet pokušavajući zadržati iskoristivost mrežnog *buffera* uvijek malo nižeg od praga koji je standardno izračunat. On je dizajniran da bude za unutarnji promet (*LAN*), ali je s druge strane dobar i s većim ***RTT*** kašnjenjem, te elastičan na veze u kojima dolazi do gubitka podataka/paketa.

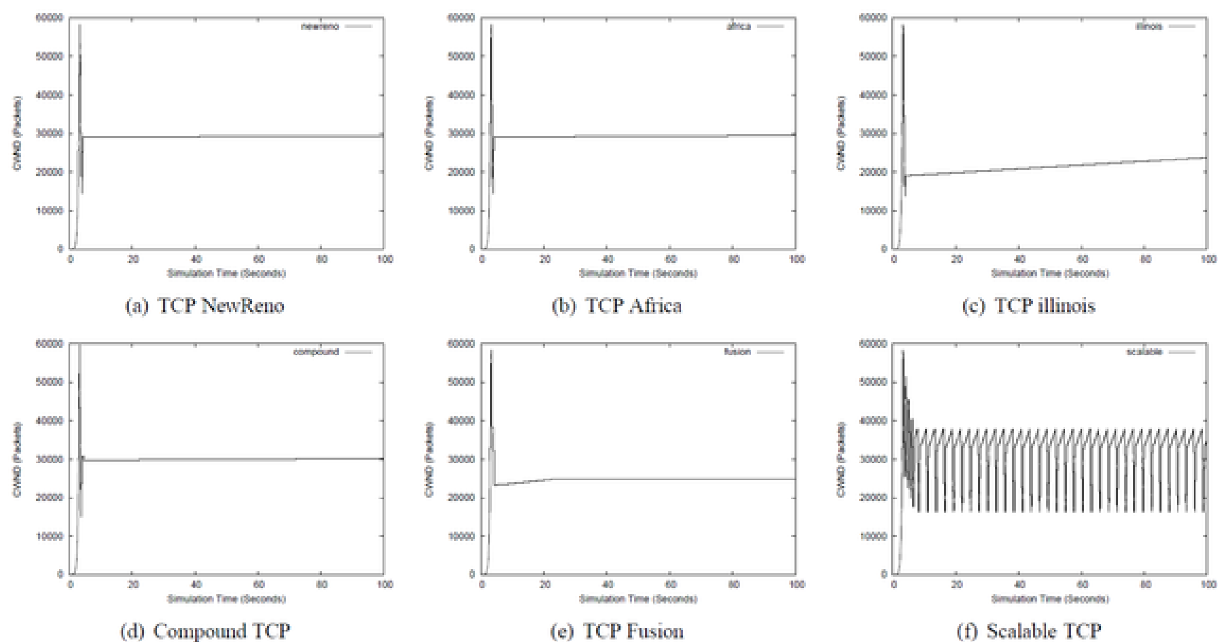
Pogledajmo rezultate testiranja u kojima su namjerno izazvani gubici paketa na mreži propusnosti 1Gbps s latencijom: od 1ms prema preklopnicima te sa 100ms između usmjerivača koji su bili povezani sa 1Gbps vezom. Mijenjale su se i veličine *buffera*, od 100 do 5.000 paketa.
Veličina paketa je bila 1.000 *bajta* a trajanje svakog testiranja 100 sekundi.
Linux kernel na svim računalima je bio v.3.4.28. Logička shema testa je vidljiva na slici 212.

**Slika 212. Shema testnog sustava.**



Pogledajte rezultate ovih mjerenja, prema veličini prozora zagušenja, a ovisno o algoritmu, vidljivih na slici 213.

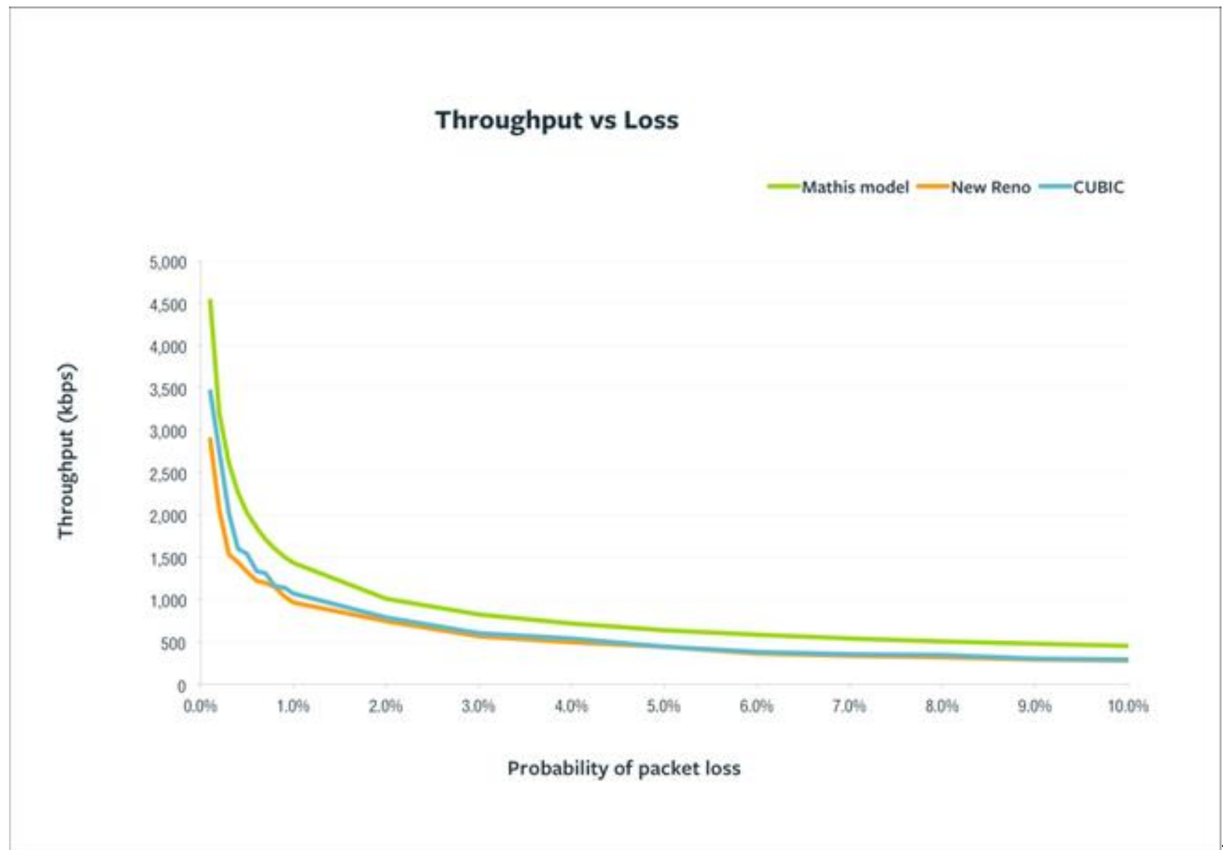
**Slika 213. Rezultati mjerenja**




**Izvor slike, informacija i mjerenja: (1024) → Comparative study of High-speed Linux TCP Variants over High-BDP Networks**

Sada možemo pogledati i još jedan graf u kojemu je uključen matematički teoretski model (***Mathis***) te dva algoritma: ***New Reno*** i ***Cubic***, u mreži između dva računala u kojoj je latencija 100 ms, a gubitak paketa između 1% i 10%. Dakle na slici 214 vidljivo je ponašanje algoritama za isto kašnjenje (latenciju), ali s povećanjem gubitaka paketa sve do 10%.

**Slika 214. Prikaz rezultata mjerenja**




**Izvor navedenih informacija i mjerenja: https://blog.thousandeyes.com/a-very-simple-model-for-tcp-throughput/**

**Vratimo se na odabir i upotrebu navedenih algoritama.**
Za probu možemo učitati cijeli nîz dostupnih algoritama, koji nam dolaze kao kernel moduli:

```
modprobe tcp_bic
modprobe tcp_dctcp
modprobe tcp_diag
modprobe tcp_highspeed
modprobe tcp_htcp
modprobe tcp_hybla
modprobe tcp_illinois
modprobe tcp_lp
modprobe tcp_scalable
modprobe tcp_vegas
modprobe tcp_veno
modprobe tcp_westwood
modprobe tcp_yeah
```

Sada pogledajmo koje sve algoritme imamo dostupne:

```
cat /proc/sys/net/ipv4/tcp_available_congestion_control
```

Isto možemo dobiti i sa slijedećom naredbom: `sysctl net.ipv4.tcp_allowed_congestion_control`
I dobiti ćemo listu dostupnih algoritama (koje smo upravo učitali):

```
cubic reno illinois bic dctcp highspeed htcp hybla lp scalable vegas veno westwood
yeah
```

Vidimo kako ih je ovdje sada cijeli nîz. Naravno za njihovo permanentno (trajno) učitavanje tijekom svakog pokretanja računala, potrebno ih je učitati tijekom svakog pokretanja sustava. Morate biti svjesni činjenice kako nisu svi algoritmi najbolji odabir za sve vrste veza, što je i očito. Međutim za pravilan odabir algoritma potrebna su prvo laboratorijska, a potom i praktična testiranja u što realnijim uvjetima kako bi se odabrao najbolji algoritam za određenu vrstu međuveze.



***Za većinu lokalnih mreža standardni* cubic *algoritam je dobar odabir!***

Za osnovno testiranje propusnosti preporučujemo upotrebu programa `iperf` odnosno njegove zadnje i optimizirane inačice `iperf3` jer s njime lako možemo mijenjati algoritam i veličinu *TCP* prozora. Dodatno `iperf3` standardno dinamički mijenja veličinu *TCP prozora* što je vidljivo u stupcu: ***Cwnd*** (*congestion control*), a što dodaje realnije stanje rada TCP protokola. Naime u staroj inačici programa (`iperf`) veličina TCP prozora je ostala fiksna što se doduše može forsirati i na `iperf3` programu, pomoću prekidača `-w` što ipak ne želimo.
Program ***iperf*** radi prema principu: **klijent → poslužitelj**.

Na prvo računalu koje testiramo pokrećemo poslužitelj; IP adresa ovog računala je: 192.168.1.100.
Sada pokrenimo *iperf* poslužitelj upotrebom naredbe `iperf3` na sljedeći način:

```
iperf3 -s
```

A na drugom računalu s kojega testiramo, pokrećemo klijentski dio programa (njegova IP je:192.168.1.200).
Pri tome *iperf* klijent generira promet.

**1.** Pokrenimo generiranje *TCP* prometa maksimalnom mogućom brzinom korištenjem standardne veličine *TCP prozora* (64kb) te standardnog algoritma na sljedeći način:

```
iperf3 -c 192.168.1.100
```

**2.** Sada promijenimo algoritam u `YeAH` na strani klijenta:

```
iperf3 -c 192.168.1.100 -C yeah
```

**3.** Dodatno možemo povećati veličinu *TCP prozora* s prekidačem `-w` a povećat ćemo *TCP prozor* na 2.75MB uz algoritam `reno` s programom *iperf3*:

```
iperf3 -c 192.168.1.100 -C reno -w 2.75M
```

Trenutna veličina *TCP prozora* je vidljiva u stupcu: **`Cwnd`** na strani klijenta.

**Testiranje kašnjenja** (*latencije*).
Ako sve radimo u testnom okruženju moguće je simulirati i kašnjenje (latenciju) mreže pomoću naredbe `tc` (*Trafic control*) i **netem** funkcionalnosti koje standardno dolaze uz Linux. Pri tome je potrebno da je ***`netem`*** (*Network emulator*) funkcionalnost kompilirana u kernelu. ***NetEm*** je kernel modul koji nudi proširenje funkcionalnosti Linux sustava kontrole prometa (*traffic control*) koji omogućava dodavanje kašnjenja, gubitka paketa, dupliciranja paketa, krivog redoslijeda slanja i još nekih drugih obilježja, paketima koji odlaze s odabranog mrežnog sučelja.
Provjerimo je li nam dostupan *netem* kernel modul, sa sljedećim nîzom naredbi:

```
cat /boot/config-`uname -r` | grep -i netem

CONFIG_NET_SCH_NETEM=m
```

Pošto imamo vrijednost `=m` to znači kako je dostupan kao kernel modul a `=y` bi značilo kako se već nalazi u našem kernelu.

Naziv ovog kernel modula je `sch_netem`. Sada ga možemo ručno učitati na sljedeći način:

```
modprobe sch_netem
```

Probajmo *pingati* naš usmjerivač (192.168.1.1) s tri *ICMP* paketa, prije promjena:

```
ping 192.168.1.1 -c3

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=255 time=1.18 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=255 time=1.20 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=255 time=1.20 ms
```

Vidimo kako je kašnjenje odnosno latencija oko 1ms.
Sada možemo u našem testnom laboratoriju konfigurirati kašnjenje za naše mrežno sučelje (pr. **eth0**) na 100ms.
Dakle nadodat ćemo kašnjenje od 100ms na svaki mrežni paket koji dođe ili ode sa mrežnog sučelja `eth0` pomoću naredbe `tc` (*traffic controll*). Pri tome moramo aktivirati odnosno koristiti `netem` network *scheduler* kako bi to uopće mogli raditi.

```
tc qdisc add dev eth0 root netem delay 100ms
```



Za detalje o ***Network scheduleru***, pogledajte poglavlje:
**25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa.**

Probajmo sada isti *ping* s tri *ICMP* paketa kako bismo vidjeli novo dodano kašnjenje (latenciju) mreže:

```
ping 192.168.1.1 -c3

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=255 time=101 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=255 time=101 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=255 time=101 ms
```

Vidimo kako se vrijeme kašnjenja povećalo za 100 ms. Uspjeli smo što smo željeli.
Pogledajmo i što smo sve promijenili s prethodnom naredbom `tc` ako ju sada pozovemo na sljedeći način:

```
tc -s qdisc

qdisc netem 8001: dev eth0 root refcnt 2 limit 1000 delay 100.0ms
                  Sent 5010 bytes 39 pkt (dropped 0, overlimits 0 requeues 0)
                  rate 0bit 0pps backlog 0b 0p requeues 0
```

Sada možemo pokretati testove u kojima je važna latencija.
Tek kada sve završimo, možemo obrisati dodanu latenciju sa sljedećom naredbom:

```
tc qdisc del dev eth0 root netem
```

Nakon ove naredbe sve vraćamo u stanje kako je bilo i prije.

**Sada kada smo naučili dodavati latenciju, pogledajmo i kako namjerno simulirati greške na mreži zajedno s latencijom.**
Dodat ćemo latenciju od 50ms te namjerne greške odnosno gubitak mrežnih paketa od 2%, isto s naredbom `tc`:

```
tc qdisc add dev eth0 root netem latency 50ms loss 2%
```

Na drugoj strani (drugom računalu) sada možemo također dodati samo latenciju (kašnjenje) od 50ms:

```
tc qdisc add dev eth0 root netem latency 50ms
```

I sada možemo pokrenuti naše testove.

Mi ćemo ipak napraviti malo detaljnije testiranje, prema sljedećoj ideji:
Konkretno pokrenuti ćemo iste testove koje su napravljeni i ucrtani u graf na slici 214. Test će biti:

- Na prijemnoj strani dodajemo latenciju (namjerno kašnjenje) od 50ms.
- Na strani pošiljatelja ćemo također dodati latenciju od 50ms, uz normalno povećanje TCP prozora odnosno *congestion control* mehanizam koji će se aktivirati kod gubitka paketa, a koji će ovisiti o algoritmu.

Pri tome ćemo mijenjati:
  - Postotak gubitka paketa:
    - 0% gubitaka odnosno bez gubitaka.
    - 0,01% gubitaka.
    - 0,02% gubitaka.
    - 0,1% gubitaka.
    - 1% gubitaka.
    - 2% gubitaka.
  - Algoritam za sve gore navedene postotke gubitaka.

Na strani primatelja dodajemo samo latenciju od 50ms:

```
tc qdisc add dev eth0 root netem latency 50ms
```

A na strani pošiljatelja, prvo dodajemo isto latenciju od 50ms i dodajemo gubitak paketa (prvo 0.01%):

```
tc qdisc add dev eth0 root netem latency 50ms loss 0.01%
```

Potom na istoj strani radimo test performansi prema primatelju.
Na strani primatelja koristimo pokretanje programa sâmo na točno definiranom mrežnom sučelju odnosno IP adresi: 192.168.1.100:

```
iperf3 -s -B 192.168.1.100
```

Pošiljatelj: prvo učitavamo sve algoritme s kojima ćemo raditi testove:

```
modprobe tcp_bic
modprobe tcp_dctcp
modprobe tcp_diag
modprobe tcp_highspeed
modprobe tcp_htcp
modprobe tcp_hybla
modprobe tcp_illinois
modprobe tcp_lp
modprobe tcp_scalable
modprobe tcp_vegas
modprobe tcp_veno
modprobe tcp_westwood
modprobe tcp_yeah
```

Zatim krećemo s testovima prema klijentu s IP: 192.168.1.100 (koristimo `-Z` zbog veće brzine kreiranja paketa):

```
iperf3 -c 192.168.1.100 -Z -t 30 -C cubic
```

Nakon testa mijenjamo algoritam (`-C cubic`) na slijedeći te ponovno pokrećemo test. Zatim sve ponavljamo za veće greške:

```
tc qdisc add dev eth0 root netem latency 50ms loss 0.02%
```

Potom ponavljamo sve testove od prije te ponovno povećavamo pogrešku i ponavljamo sve testove. Sve testove smo ponavljali pet puta te zapisivali srednju vrijednost svakog testa. Ako je neko mjerenje iz grupe odstupalo više od 20% unutar pet testova, testiranje je ponovljeno još pet puta kao i u slučajevima kada je neka grupa odstupala od druge grupe za 20% ili više postotaka.

Kada sve završimo možemo obrisati dodane latencije i gubitak paketa (ovo treba napraviti na oba računala) s `tc` na sljedeći način:

```
tc qdisc del dev eth0 root netem
```

Važno je i promatrati kako se veličina *TCP prozora* (**`Cwnd`**) mijenja tijekom testiranja, a pogotovo koliko brzo se vraća na staro stanje nakon detekcije grešaka. Za što realnije testiranje potrebno je duže vrijeme testa kao i dinamičnije i promjenjivije simuliranje gubitaka koje ovdje nećemo obraditi jer bi izlazni graf bio previše kompleksan.

Ovdje nam je želja sâmo okvirno vidjeti utjecaj algoritma u određenom postotku gubitaka na konačnu propusnost mreže (uz željenu latenciju odnosno kašnjenje).

**Uvjeti mjerenja** uz specifikaciju oba poslužitelja su sljedeći:

- CPU: 2 x CPU Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz - 24 CPU jezgre; ukupno 48 CPU jezgri – **NUMA.**
- RAM: 128 GB.
- LAN: 2 x *Emulex Corporation OneConnect NIC* (*Skyhawk*) - 2 x 10Gbps.
  - Mreža: LAN - LAN : latencija: 0,03 ms.
- Linux OS: *RedHat Enterprise Linux* v. 7.3.
  - kernel: 3.10.0-514.21.2.el7.x86_64.
  - LAN driver: *be2net* (v. 11.0.0.0r).

**Slika 215. Prikaz kreiranja i zaprimanja mrežnog prometa**



**Slika 216. Prikaz rezultata mjerenja.**

## Rezultati testiranja/mjerenja koja smo dobili

TCP: utjecaj gubitaka paketa na propusnost mreže, ovisno o algoritmu

Propusnost : Mbps

Gubici paketa (%)

cubic, reno, bic, highspeed, htcp, hybla, illinois, lp, scalable, vegas, veno, westwood, yeah

Iz rezultata mjerenja sa slike 216 na navedenoj opremi i opisanim uvjetima vidljivo je ponašanje raznih algoritama u slučajevima gubitaka podataka kao i direktan utjecaj na propusnost mreže. Ova mjerenja možete uzeti kao polaznu točku za vlastita mjerenja na osnovu kojih možete optimirati vašu mrežu, ovisno o uvjetima na njoj.

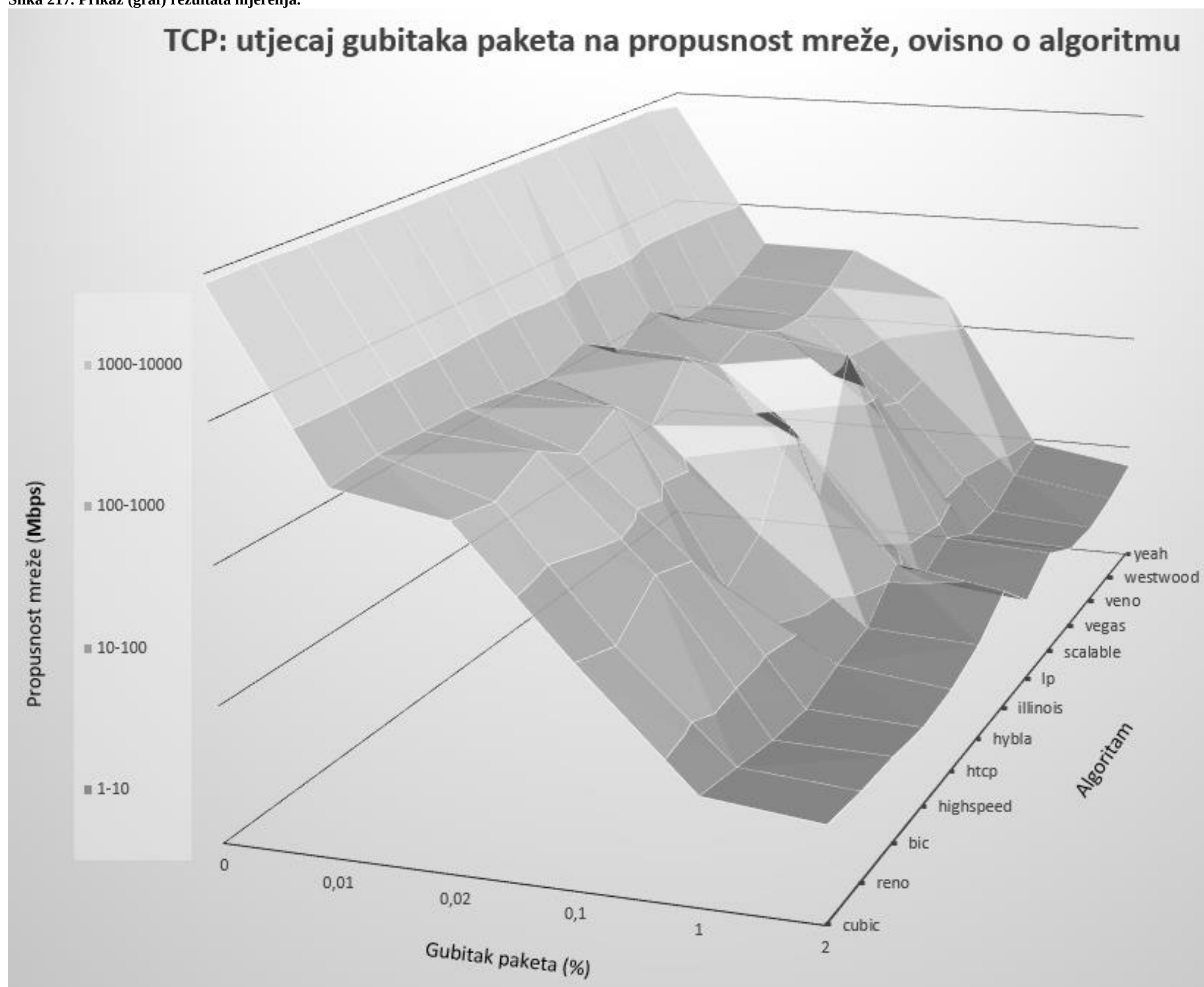
Ako uvrstimo i `iperf3` mjerenja između oba poslužitelja, za sve algoritme bez dodatne latencije i bez namjerno dodanog gubitka paketa, dobivamo propusnosti u opsegu od: **8.5 Gbps** do **8.8 Gbps**.

Međutim uvrstimo li sve navedene gubitke paketa moramo preći u logaritamsko mjerilo u obje osi. Na slici 217 je vidljivo kako se povećanjem gubitka paketa na horizontalnoj osi (**gubici: 0%, 0,01%, 0,02%, 0,1%, 1% i 2%**) drastično smanjuje propusnost mreže na vertikalnoj osi: 1Mbps – 10.000 Mbps (10Gbps).



Najdrastičnije smanjenje brzine je od trenutka kada nema gubitaka (0%) do gubitaka od 0,01% gdje propusnost pada sa prosječnih 8.500 Mbps (**8,5 Gbps**) strmo prema vrijednosti između **400 Mbps** i **500 Mbps**.

Slika 217. Prikaz (graf) rezultata mjerenja.






Za algoritme koji također utječu na propusnost, ali i način na koji se mrežni paketi na nižoj razini prihvaćaju, klasificiraju (**QoS**) i obrađuju, pogledajte poglavlje: **25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa.**



U stvarnim mrežama u odnosu na ova mjerenja u laboratorijskim uvjetima, a pogotovo jer je u našem testu veza između uređaja koji generira promet i onoga koji ga prima, direktna, rezultati mogu biti drastičniji. Naime zamislimo normalnu Internet vezu između dvaju točaka u komunikaciji, između kojih se nalaze i desetci usmjerivača i cijeli nîz mrežnih preklopnika, vatrozida i drugih uređaja. Važno je razumjeti da svaki od navedenih mrežnih uređaja utječe na propusnost, uvodi dodatno kašnjenje: ako ništa drugo zbog sâme činjenice da im je potrebno određeno vrijeme za dohvaćanje, obradu i ponovno prosljeđivanje mrežnih paketa, ali i eventualno može biti uzrok gubitka paketa (pr. zbog preopterećenosti ili drugih uzroka).

Ne zaboravimo da svaki mrežni uređaj mora minimalno imati mehanizam za raspodjelu mrežnih paketa, pri čemu je čak vrlo vjerojatno da će većina tih uređaja imati odabrane poprilično različite navedene mehanizme. Već sâma činjenica da eventualno neki od uređaja u nîzu ima u upotrebi nešto drugačiji mehanizam, definitivno može utjecati i utječe na konačnu propusnost, ali i kašnjenje (latenciju) u mrežnoj komunikaciji između krajnjih točaka u komunikaciji.
Naime različiti algoritmi unutar mehanizama za raspodjelu mrežnih paketa na drugačiji način prihvaćaju mrežne pakete, spremaju ih u svoje međumemorije te ih kasnije odabiru za prosljeđivanje te prosljeđuju dalje. Ako u to uključimo i razne međumemorije unutar svakog od uređaja: od prstenaste memorije mrežnog sučelja, *txqueue* memorije, i drugih, dolazimo i potencijalno do problema zajedničkog imena ***bufferbloat***. ***Bufferbloat*** je problem upotrebe prekomjerne veličine međumemorije, koja efektivno pomaže za povećanje propusnosti, ali može drastično djelovati na kašnjenje odnosno latenciju.



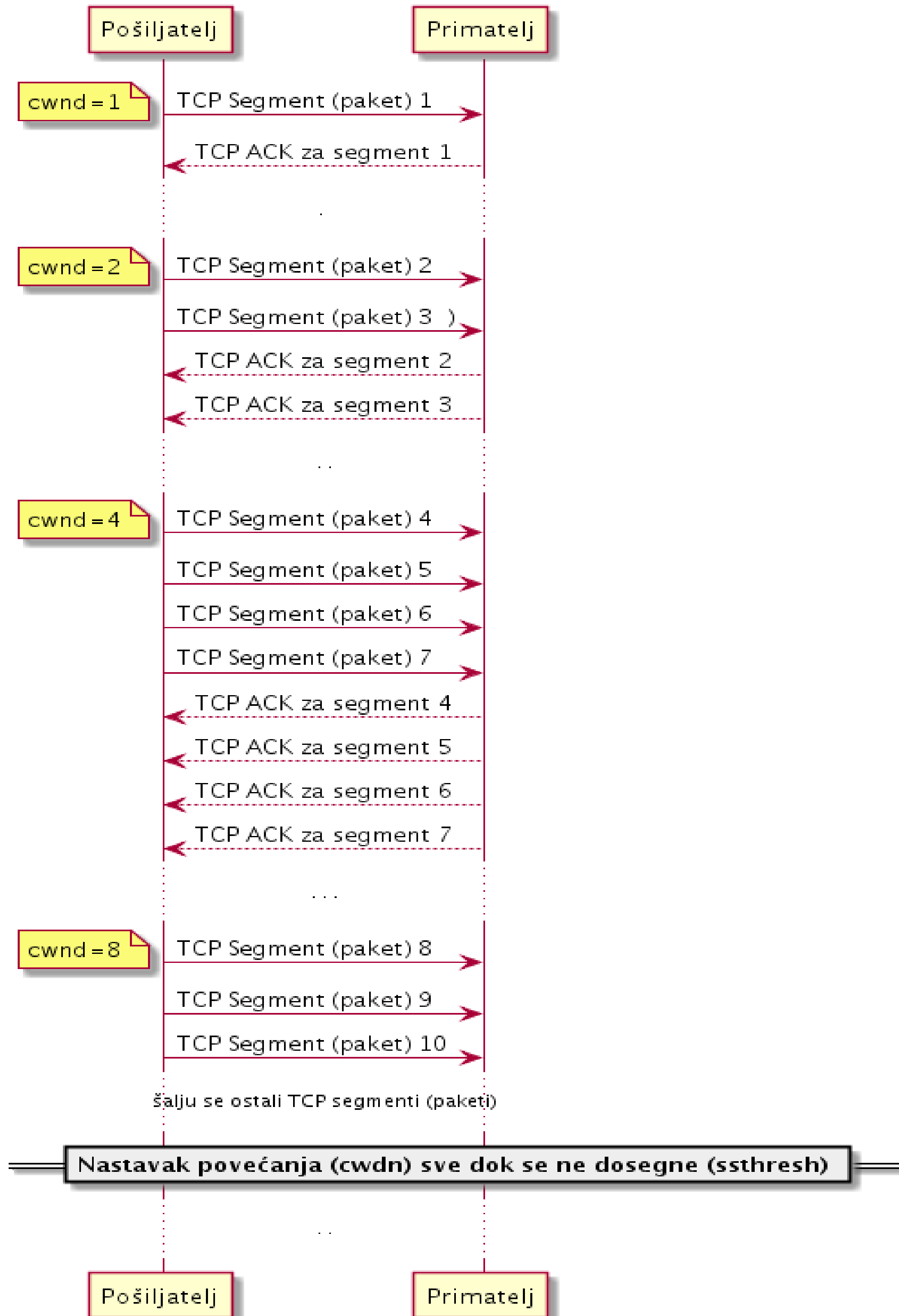
Za ***bufferbloat*** pogledajte poglavlje: **25.1.4.1. Bufferbloat.**

**Izvori informacija: (273),(274),(275),(276),(277),(278),(279),(280),(281),(1024), `man 7 tcp`, RFC 2581 i RFC 5681**

#### 24.2.9.1. Usporeni početak i izbjegavanje zagušenja

*Mehanizam* usporenog početka odnosno mehanizam zvan „***Slow start***" dio je strategije nadzora zagušenja koju koristi *TCP* protokol za prijenos podataka. Usporeni početak upotrebljava se zajedno s drugim mehanizmima kako bi se izbjeglo slanje više podataka nego što je mreža sposobna prenijeti, odnosno izbjeći zagušenje mreže. Ovaj mehanizam je definiran u **RFC 5681**. Kada se detektira zagušenje mreže „*usporeni početak*" započinje sa slanjem paketa s manjom veličinom prozora zagušenja; tzv. *congestion window size* (**cwnd**) od 1, 2, 4, 8 ili sve do 10 veličina ***MSS***. Dakle veličine prozora zagušenja, odnosno *congestion window* je zapravo postavljanje vrijednosti *TCP* prozora (*TCP Window*) na vrijednost koju je odredio odnosno izračunao pošiljatelj, a ne onu koju je zahtijevao primatelj, kao što je slučaj kod klasičnog povećanja *TCP* prozora koje zahtjeva primatelj kod upotrebe mehanizma kontrole protoka (*flow control*).

**Slika 218. Prikaz rada TCP protokola kod promijene vrijednosti prozora zagušenja.**



Vrijednost *congestion windowa* nadalje će se povećati za jedan, sa svakom potvrdom s druge strane (*ACK*), efektivnim udvostručavanjem veličine prozora svakih ***RTT*** vremena. U kontroli zagušenja sa sporim početkom *TCP* brzo povećava veličinu prozora (*TCP window size*) kako bi dosegao maksimalnu brzinu prijenosa što je brže moguće. Ova izračunata veličina prozora raste kako *TCP* potvrđuje sposobnost mreže da prenosi podatke bez pogrešaka. Međutim to može ići samo do maksimalne veličine prozora (**`rwnd`**). U svakom slučaju, brzina prijenosa bit će povećana ovim mehanizmom polaganog pokretanja sve dok se ne otkrije gubitak ili ako je veličina prijemnog prozora: *receiver's advertised window* (**`rwnd`**) prijemnika ograničavajući čimbenik. Ako se pak dogode gubitci (paketa), *TCP* pretpostavlja da je to zbog zagušenja mreže i poduzima korake za smanjenje trenutnog opterećenja na mreži. Jednom kada se dostigne određena granica (ovisna o algoritmu) koja se zove ***slow start threshold value*** (***ssthresh***) ponašanje *TCP* se mijenja iz mehanizma za sporo pokretanje, do algoritma linearnog rasta koji se zove ***izbjegavanje zagušenja***. U ovom trenutku, prozor zagušenja se povećava za jedan segment za svako *RTT* prolazno vrijeme. *Slow start threshold value* (***ssthresh***) je vrijednost koju je izračunao TCP algoritam za trenutno stanje veze, a vidljiv je u *TCP* paketu kao: `Calculated Window size`. Mjerenja ovise o korištenom konkretnom algoritmu izbjegavanja zagušenja *TCPa*. Stoga prvo pogledajmo kako funkcionira mehanizam ***usporeni početak*** (***slow start***) nakon detektiranja zagušenja mreže, kako je vidljivo na slici **218**.

U fazama nakon detekcije zagušenja, sve dok se ne dođe do veličine prozora koja je izračunata (***ssthresh***) sve se odvija na gore opisan način. Mogli bi se zapitati zbog čega nakon zagušenja mreže odmah ne prihvatiti veličinu *TCP* prozora primatelja (*TCP Receive Window*); koju podržava primatelj. Tada bi se preskočilo ovo naizgled sporo povećanje veličine TCP prozora s mehanizmom ***slow start***. Međutim u praksi su se ipak ovakvi mehanizmi pokazali puno pouzdaniji, a i povećanje koje oni rade nije baš sporo kao što može izgledati, jer ono ipak raste vrlo brzo, mada ne i eksponencijalno cijelo vrijeme. Iako se ova strategija naziva ***usporeni početak*** (***slow start***) njegov rast veličine prozora zagušenja je prilično agresivan. Ipak prije nego što je u *TCP*-u uveden usporeni početak, početna faza izbjegavanja zagušenja bila je još brža. *Detaljno ponašanje u trenutku gubitka paketa ovisi o konkretnom algoritmu izbjegavanja zagušenja* TCP-a *koji se koristi.*

## Izbjegavanje zagušenja (Congestion Avoidance)

Iako je u formulu po kojoj raste *veličina prozora zagušenja* uračunata vjerojatnost da se propusnost u međuvremenu mogla i povećati ili smanjiti, rast nije eksponencijalno brz. To je zato što bi se tada brzo moglo dogoditi da se ponovno zaguši mreža i da se sve vrati na početak. Naime veličina *prozora zagušenja* prvo raste vrlo brzo, sve dok se ne dosegne izračunata veličina znana kao (***ssthresh***), odnosno ciljana veličina *TCP prozora (ne i konačna)*. Vrijednost ***ssthresh*** se ažurira nakon svakog aktiviranja mehanizma sporog pokretanja.
Ona tada nastavlja polagano rasti za po jednu MSS vrijednost, sve dok se ne pojave duplicirane *TCP ACK* poruke (koje indiciraju probleme) za svako RTT vrijeme, odnosno trenutno "prolazno" vrijeme paketa. Dakle ova faza se zove *Congestion Avoidance* odnosno faza izbjegavanja (ponovnog) zagušenja. Međutim postoje i varijacije koje uvodi svaki od gore navedenih algoritama koji je optimiziran za određene uvjete na mreži: propusnost, latenciju i/ili gubitke.

Neke od *sysctl* varijabli ovdje u upotrebi su:

- `net.ipv4.tcp_reordering` - Ovaj parametar određuje prag (*threshold*) za toleranciju izvanrednih (*out-of-order*) segmenata u TCP komunikaciji.
- `net.ipv4.tcp_max_reordering` - Ovaj parametar određuje maksimalni broj dopuštenih izvanrednih (out-of-order) segmenata prije nego što se TCP primatelj odluči na poduzimanje radnji za reorganizaciju ili odbacivanje izvanrednih segmenata.



Za ***out-of-order*** pogledajte poglavlje**: 24.2.14. Primitak paketa izvan redoslijeda slanja (*Out-Of-Order*).**

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju. Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
```

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju zatvaranja TCP veze:

| Naziv brojača | Opis |
|---|---|
| **`TcpExtTCPSlowStartRetrans`** | Ovaj brojač bilježi broj TCP segmenta koji su retransmitirani (ponovno poslani) tijekom faze "*slow start*" u TCP protokolu. *Slow start* je jedan od mehanizama za kontrolu zagušenja (*congestion control*) u TCP-u, gdje se postupno povećava broj poslanih segmenata dok se ne dosegnu granice propusnosti mreže ili detektira zagušenje. Retransmisija tijekom "*slow start*" faze događa se kada se segmenti izgube ili ne stignu na odredište. |
| **`TcpExtTCPThresholdRetrans`** | Ovaj brojač bilježi broj TCP segmenta koji su retransmitirani tijekom faze "*Congestion Avoidance*". Kada TCP prelazi iz faze "*slow start*" u fazu "*Congestion Avoidance*", prag (*threshold*) za broj ACK potvrda koje su primljene prije nego što se podupla veličina prozora (*window*) smanjuje se na polovicu kako bi se sporije povećavao broj poslanih segmenata i spriječilo naglo zasićenje mreže. Retransmisija tijekom ove faze može se dogoditi kao odgovor na izgubljene segmente ili druge probleme u mreži. |

**Izvori informacija: (326),(327),(328),(329), (732),(1406), (1651),(K-6), `man 7 tcp`,`man nstat`, RFC 5681.**

### 24.2.9.2. Brzo slanje *(fast retransmit)* i brzi oporavak *(fast recovery)*

Oba mehanizma: brzo slanje (*Fast retransmit)* i brzi oporavak (*Fast recovery*) opisana su u RFC 5681 i to u poglavlju **3.2**. Njih koriste neki od navedenih algoritama, dok ih neki drugi ne koriste. Pogledajmo primjere njihove upotrebe (za par algoritama):

- **TCP Tahoe** algoritam koristi:
    - *Slow start.*
    - *Congestion Avoidance.*
    - *Fast Retransmit.*
- **TCP Reno** algoritam koristi:
    - *Fast Recovery.*
    - *Fast Retransmit.*
- **TCP SACK** funkcionalnost koristi:
    - *Fast Recovery* nakon tri duplicirane *TCP ACK poruke.*

## Brzo slanje *(Fast retransmit)*

Mehanizam brzog slanja odnosno *Fast retransmit* (**FRR**) mehanizam daje poboljšanje *TCP* protokola koje smanjuje vrijeme koje pošiljatelj čeka prije ponovnog slanja izgubljenog segmenta (*paketa*). *TCP* pošiljatelj koristi vremenski okvir (*timer)* za prepoznavanje izgubljenih segmenata. Ako se za određeni segment ne primi potvrda (*TCP ACK*) u određenom vremenu, a koje je funkcija procijenjenog (RTT) vremena, pošiljatelj će pretpostaviti da je segment izgubljen u mreži te da će ponovno morati poslati taj segment. Međutim ako prijemnik primi podatkovni segment koji nije u redu, odmah šalje dupliciranu potvrdu (*TCP ACK*) pošiljatelju. Ako pošiljatelj pak zaprimi tri duplicirane potvrde (*TCP ACK*), pretpostavlja se da se segment podataka koji je označen potvrdama (*TCP ACK*) izgubio te se odmah ponovno prenosi izgubljeni segment. Stoga vrijeme nije izgubljeno čekajući vremensko ograničenje kako bi se ponovno započeo prijenos izgubljenih (nepotvrđenih) segmenata.
Dakle dvostruko potvrđivanje (*TCP ACK*) je osnova za mehanizam brzog ponovnog slanja koji funkcionira na sljedeći način:

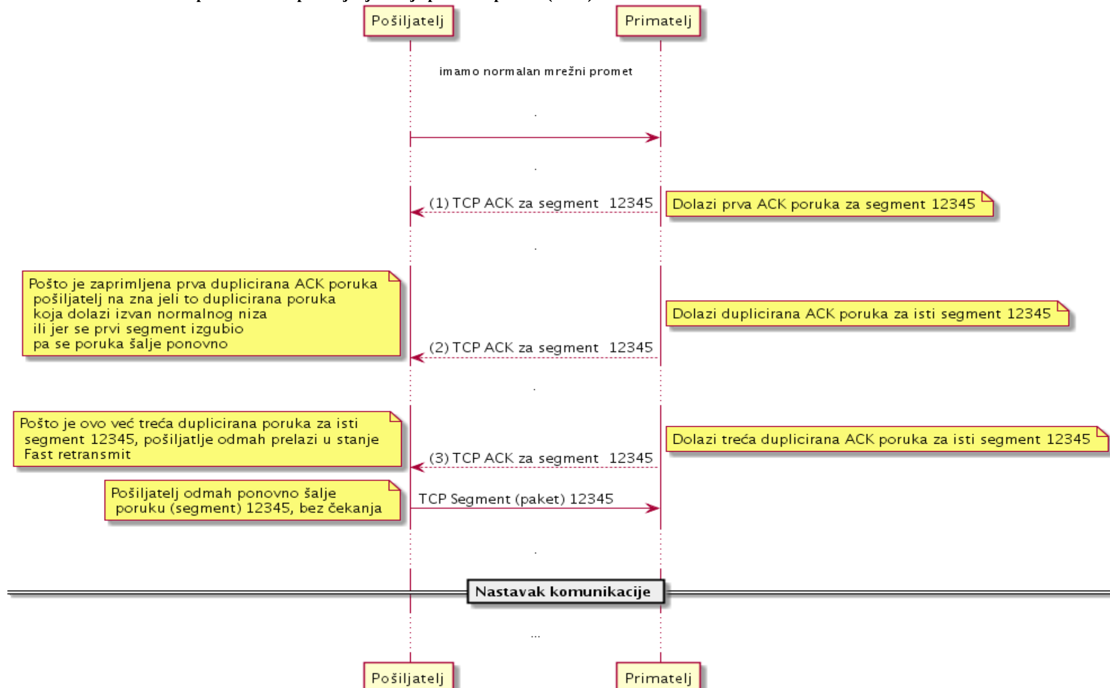
1. Nakon primanja paketa (npr. s rednim brojem 1) primatelj šalje potvrdu dodavanjem broja jedan na broj sekvence odnosno potvrđuje se sve do broja 2. To znači kako je primatelj primio paketni broj 1 i očekuje paket broj 2 od pošiljatelja. Pretpostavimo da su tri kasnija paketa izgubljena.
2. U međuvremenu primatelj prima pakete brojeva 5 i 6. Nakon primanja paketa broj 5 primatelj šalje potvrdu, ali još uvijek samo za redni broj 2. Kada primatelj prima paket broj 6, on šalje još jednu potvrdu za paket broj 2. Stoga pošiljatelj prima više od jedne potvrde s istim brojem paketa (dvije u ovom primjeru) što se naziva duplicirano potvrđivanje ili duplicirana *TCP ACK* poruka.

***Fast retransmit*** mehanizam funkcionira na sljedeći način:

1. Ako *TCP* pošiljatelj prima određeni broj potvrda (*TCP ACK*) koji je obično postavljen na tri duplicirane potvrde s istim potvrdnim brojem, pošiljatelj može biti uvjeren kako je segment sa sljedećim višim brojem paketa izgubljen, i neće stići izvan niza.
2. Pošiljatelj će zatim ponovo poslati paket koji je pretpostavio da je odbačen, bez čekanja na istek njegovog vremenskog okvira (brojača), nakon kojega bi trebao ponoviti slanje (u navedenom slučaju).

Ovaj mehanizam se aktivira, ako su zaprimljena tri potvrdna paketa, kojima se potvrđuje isti paket odnosno segment s istim brojem. Tada se automatski odmah kreće u ponovno slanje (*retransmisiju*), od paketa koji je zadnji potvrđen, bez čekanja na istek vremenskog okvira (*timera*). Pogledajmo kako izgleda ovakva komunikacija na slici 219.

**Slika 219. Prikaz rada TCP protokola kod ponavljanja slanja potvrdne poruke (ACK).**




**Izvor informacija:(330),(K-6), `man 7 tcp`, RFC 5681.**

### Brzi oporavak *(Fast recovery)*

Mehanizam brzog oporavka (*Fast recovery*) je prvi puta definiran u RFC 2001 te je kasnije proširen u RFC 2581 pa dodatno proširen s RFC 5681, a ponekad se kombinira s prethodno navedenim *Fast retransmit* mehanizmom (ovisno o algoritmu). Naime u slučaju o kojem smo prethodno govorili: kada se dogodilo da je pošiljatelj zaprimio tri *TCP ACK* poruke za isti segment, kreće se u *Fast retransmit* prema kojemu se ponavlja izgubljeni segment bez ikakvog čekanja. Međutim sada se obično ulazi u stanje koje se zove *stanje brzog oporavka*. Mada sve ponovno ovisi o algoritmu koji je trenutno u upotrebi.
U svakom slučaju, ako algoritam nakon *brzog slanja* prelazi u stanje *brzog oporavka* ili ako je iz nekog drugog stanja ušao stanje brzog oporavka u ovom koraku se događa sljedeće:

1. Pošiljatelj postavlja *TCP prozor* odnosno ***cwnd*** (*Congestion window*) na trenutno izračunati ***ssthresh* + 3** segmenta. Ovaj mehanizam stoga efektivno postavlja inicijalnu vrijednost na efektivnu vrijednost polovinu one koja je bila prije nego se ovakav događaj dogodio.
2. Primatelj šalje potvrdu (*TCP ACK*) koju pošiljatelj zaprima te se na osnovu nje izračunava nova veličina *TCP* prozora odnosno *congestion window*-a, a koja se sada još povećava.
3. Ako se zaprimi još jedna *TCP ACK* poruka koju može poslati primatelj, s njom se može potvrditi dodatno proširenje *TCP prozora* koji se tada može još malo više povećati. S time se dolazi do stanja u kojemu pošiljatelj može poslati traženi segment sa sada već još malo povećanom veličinom *congestion window*-a odnosno *TCP* prozora.

Na ovaj način se veličina *TCP* prozora prilično brzo povećala te se i samim time vrlo brzo vratilo u početno stanje brze komunikacije, zahvaljujući povećanom *TCP* prozoru odnosno *congestion window*-u. Sada se obično ulazi u stanje izbjegavanja zagušenja (*congestion avoidance)* u kojemu se oprezno može povećavati veličina *TCP* prozora kako se ne bi ponovno zagušila mreža. Kombinacija *Fast retransmit* i *Fast recovery* ne radi učinkovito, ako u kratkom vremenskom razdoblju dolazi do više gubitaka podataka. Stoga ***TCP Reno*** algoritam nije najbolji odabir za ovakve slučajeve jer on koristi isključivo samo navedena dva mehanizma.

**Izvori informacija: (331),(332), (K-6), RFC 2001, RFC 2581, RFC 5681.**

## 24.2.9.3. Explicit Congestion Notification (*ECN*)

***Explicit Congestion Notification*** (*ECN*) odnosno *eksplicitna obavijest zagušenja* je proširenje *IP* i *TCP* protokola te je definirana u RFC 3168. *ECN* dopušta obavijest o zagušenju mreže bez odbacivanja paketa. *ECN* je opcija koja se može koristiti između dvije *ECN* krajnje točke u komunikaciji, isključivo i sâmo kada ih i mrežna infrastruktura podržava. Konvencionalno TCP/IP mreže signaliziraju zagušenost: odbacivanjem paketa, ne potvrđivanjem primitka ili istekom vremena u kojem se potvrdni paket morao zaprimiti. Kada se *ECN* uspješno dogovori, *ECN* usmjerivač (*router*) može postaviti oznaku u IP zaglavlju (unutar *DSCP* polja) kao i u *TCP* zaglavlju (*CWR*, *ECE* i *Nonce*) umjesto da odbaci paket kako bi signalizirao predstojeće zagušenje. Dakle podrška za ECN je potrebna i u IP i TCP sloju komunikacije, što u današnje vrijeme (od cca. 2015.g.) podržavaju svi operativni sustavi. Na IP sloju imamo dva ECN bita, koji označavaju sljedeća moguća ECN stanja:

- `00` – Uređaj ne podržava ECN (*Non ECN-Capable Transport, Non-ECT*).
- `10` – Uređaj podržava ECN (*ECN Capable Transport, **ECT(0)***).
- `01` – Uređaj podržava ECN (*ECN Capable Transport, **ECT(1)***).
- `11` – Detektirano je zagušenje (*Congestion Encountered, CE*).

TCP podržava ECN koristeći dvije zastavice u TCP zaglavlju. Prva je: ***ECN-Echo (ECE)*** i koristi se za vraćanje indikacije zagušenja tj. signala pošiljatelja da smanji količinu informacija/podataka koje šalje. Druga je ona koja označava smanjenje prozora zagušenja (***CWR***), a koja potvrđuje da je primljen odgovor na indikaciju zagušenja. Kako bi se ECN mogao koristiti na TCP sloju, o njemu se mora pregovarati prilikom uspostavljanja TCP veze uključivanjem prikladnih opcija u segmente/pakete *TCP SYN* i *TCP SYN-ACK*. U svakom slučaju, prijemnik paketa šalje signal zagušenja prema pošiljatelju, što smanjuje brzinu prijenosa kao da je otkriven ispušteni/izgubljeni paket i pokrenuo se mehanizam detekcije zagušenja, ali znatno brže od bilo kojih od **AIMD** algoritama. Stoga se ***ECN*** intenzivno koristi u naprednijim mrežama (pr. telekomi, podatkovni centri i sl.).
Upotreba ***ECN*** se u Linuxu definira u dvije ***sysctl*** varijable: `net.ipv4.tcp_ecn` i `net.ipv4.tcp_ecn_fallback`.

**Izvori informacija: (333),(334),(K-6), `man 7 tcp`, RFC 3168.**

## 24.2.10. TCP Push (*PSH*) i Urgent (*URG*)

Unutar svakog *TCP* paketa u polju sa zastavicama (Engl. *Flags*) nalaze se zastavice koje označavaju vrstu poruke odnosno paketa. Prisjetite se početnog poglavlja o TCP-u. Ovdje ćemo govoriti o dvije posebne funkcionalnosti odnosno zastavice:

- **`PSH`** - prioritetno prosljeđivanje podataka unutar paketa, prema aplikacijskom sloju (aplikaciji).
- **`URG`** - urgentno prosljeđivanje podataka aplikaciji, slično kao **`PSH`**.

Dakle u bilo kojem TCP paketu može biti aktivirana bilo koja zastavica (ili više njih). *Aktivirana* znači kako određena zastavica ima vrijednost **1**, odnosno neaktivirana, ako je u vrijednosti **0**. O tome koje zastavice su aktivirane u kojem paketu ovisi i značenje paketa. Ako je primjerice aktivirana samo zastavica *`SYN`* to znači kako se radi o inicijalnom paketu za otvaranje TCP komunikacijskog kanala, što označava *TCP SYN* poruku. Ako potom uslijedi odgovor s druge strane, koji sadrži aktivirane zastavice *`SYN`* i *`ACK`* to znači kako je ovo odgovor na zahtjev za otvaranje TCP komunikacijskog kanala i tako dalje.
Međutim u ovoj cjelini upoznat ćemo se s *`PSH`* i *`URG`* zastavicama te njihovim funkcionalnostima.

**Izvori informacija: (K-6), `man 7 tcp`, RFC 793, RFC 6093 i RFC 9293.**

## 24.2.10.1. Urgent (*URG*)

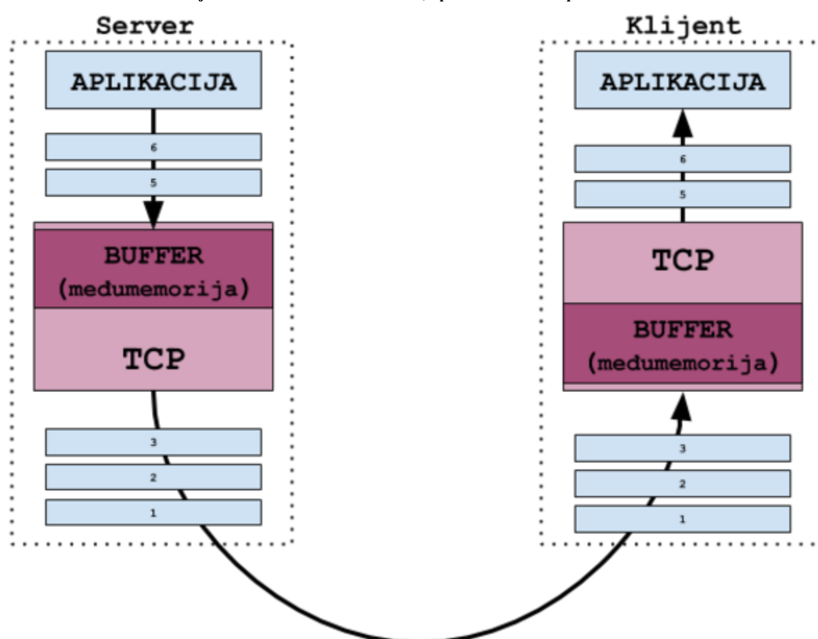
***Urgent*** funkcionalnost je definirana u RFC 6093. ***URG*** zastavica se koristi kako bi strani koja prima paket/segment dala do znanja kako se unutar paketa nalaze podaci ili dio podataka koje je važno hitno (Engl. *Urgent*) prihvatiti i obraditi. Ako je `URG` zastavica uključena, čita se polje: `Urgent Pointer` koje se nalazi u TCP zaglavlju. Ovo je 16 bitni broj. Polje: `Urgent Pointer` govori koliko *bajta* od podataka unutar paketa u kojem se nalazi, je hitno (urgentno). Tako je moguće da paket nosi 1200 *bajta* podataka, ali od kojih aplikaciji treba pod hitno dostaviti samo prvih 800 *bajta*. Tada će vrijednost `Urgent Pointer`a biti 800. Kada primatelj zaprimi ovakav paket TCP sloj će se pobrinuti da otvori poseban komunikacijski kanal prema aplikaciji te joj dostavi ove urgentne podatke. Ovo polje kao i `URG` zastavica danas se rijetko koriste, što zbog kompleksnosti, a što zbog češće upotrebe zastavice/funkcionalnosti `PSH` (***push***) o kojoj ćemo govoriti u sljedećoj cjelini.

**Izvor informacija: RFC 6093. i RFC 9293.**

## 24.2.10.2. Push (*PSH*)

***Push*** funkcionalnost je definirana u izvornom RFC 793 standardu. Kako bismo uopće razumjeli kako se podaci s aplikacije šalju do *TCP* sloja, te na mrežu, a potom drugoj strani u komunikaciji: kako se zaprimaju i dolaze ponovno do *TCP* sloja i potom do aplikacije, pogledajmo sliku **220**.

**Slika 220. Prikaz komunikacije između dva računala na mreži, upotrebom TCP/IP protokola.**




Dakle TCP radi na OSI sloju četiri, a ostale slojeve smo sâmo nacrtali, kako bi bilo jasno da su i oni uključeni u komunikaciju, iako nam za razumijevanje ove problematike nisu važni.
Naime podaci iz aplikacije se s poslužitelja (*server*) šalju do TCP međumemorije (Engl. *Buffer*) na poslužitelju, s kojom aplikacija jedino i ima vezu. Dakle aplikacija se spaja na mrežni TCP *socket* na koji može slati ili iz kojega može čitati podatke. Aplikaciju ništa drugo niti ne zanima, niti treba išta više od toga znati.
Pošto je aplikacija poslala podatke na *TCP socket*, oni su završili u međuspremniku TCP sloja. Sa svake strane postoje zapravo dva TCP međuspremnika: dolazni i odlazni, ali smo pojednostavili priču u sâmo jednom smjeru.

Kada se u međuspremnik nakupi dovoljno podataka za slanje, oni se šalju nižim slojevima (3-2-1) te potom na mrežu.
Druga strana (klijent) zaprima paket, koji ponovno prolazi kroz niže OSI slojeve i dolazi do ulaznog TCP međuspremnika (*TCP buffer*). Ovaj međuspremnik se inače popunjava s podacima iz više mrežnih paketa te kada se dovoljno napuni, podaci iz međuspremnika se šalju aplikaciji. Ovi TCP međuspremnici na prijemnoj strani su dobri jer efikasno spremaju podatke koji su vrlo vjerojatno bili razlomljeni u više mrežnih paketa, to jest koji nisu stali u jedan mrežni paket već su bili razlomljeni u više njih. Problem kod ovakvog načina rada događa se za aplikacije koje zahtijevaju što brži prijem s druge strane, poput aplikacija koje zahtijevaju rad u realnom vremenu (Engl. *real-time*). Ovdje uskače zastavica `PSH` odnosno *Push* funkcionalnost koja postavljena u svaki paket koji se šalje, govori primaocu paketa (klijentu), kako mora zaobići TCP međuspremnik i podatke odmah dostaviti aplikaciji. Dakle sâma aplikacija (serverska strana) može *TCP socketu* tijekom slanja podataka naznačiti da se koristi „*pushing*“ mehanizam za slanje. S time TCP sloj uključuje `PSH` zastavicu. Jedan od očitih primjera upotrebe `PSH` mehanizma su programi za udaljeni pristup poput programa `telnet` i `ssh`. Naime svakim pritiskom na neku tipku od klijenta prema poslužiteljskoj (serverskoj) strani, na čiji SSH servis se spajamo, paketi bi se prvo prikupljali u međumemoriju, a tek kasnije bi bili obrađeni.

Tek potom bi bio vraćen potvrdni paket, koji sadržava i potvrdu svake stisnute tipke koja se tek tada (tijekom potvrde) iscrtava na ekran. Jasno je kako ovakav način rada za SSH nije prikladan jer ne bi vidjeli koju tipku smo stisnuli, a koji izlaz bi dobili kao odgovor. Pogledajmo primjer spajanja klijenta (IP:192.168.1.128) na *SSH* servis (IP:192.168.1.254). Naime nakon što se uspostavi TCP komunikacijski kanal, kreće se u uspostavu SSH komunikacije.

Prvi *SSH* paket (zapravo četvrti gledajući od `SYN` paketa) šalje poslužitelj (*server*) prema klijentu. Taj paket izgleda ovako:

```
1. Frame 191: 97 bytes on wire (776 bits), 97 bytes captured (776 bits
2. Ethernet II, Src: Cisco_85:4c:c6 (00:12:00:85:4c:c6), Dst: Fujitsu_b3:5f:b6
(2c:d4:44:b3:5f:b6)
3. Internet Protocol Version 4, Src: 192.168.1.254, Dst: 192.168.1.128
4. Transmission Control Protocol, Src Port: 22, Dst Port: 16107, Seq: 1, Ack: 1, Len:
43
    Source Port: 22
    Destination Port: 16107
    [Stream index: 6]
    [TCP Segment Len: 43]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 44    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  + Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
         [TCP Flags: ·······AP···]

    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0x8374 [unverified]
    Urgent pointer: 0
    [SEQ/ACK analysis]
    TCP payload (43 bytes)
+ SSH Protocol
   Protocol: SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8
```

U zastavicama vidimo kako je odmah uz `ACK` uključena i `PSH` (*`Flags: 0x018 (PSH, ACK)`*) što znači kako se već u početku kreće s komunikacijom upotrebom *Push* metode. Svi paketi koji slijede u ovoj *SSH* komunikaciji, također koriste `PSH` metodu u komunikaciji, kako bi što brže došli do aplikacije (i nazad), zaobilazeći međumemoriju na TCP sloju.

**Izvor informacija:(K-6), RFC793 i RFC 9293.**

## 24.2.11. Kako se zatvara TCP veza

TCP veza se može zatvoriti na nekoliko načina:

- Zatvaranje putem takozvane *four-way handshake* procedure u četiri koraka (ovo je standardna procedura).
- Moguća je i varijanta zatvaranja veze u tri koraka.
- Te je moguća i varijanta zatvaranja veze u dva koraka.
- Veza se dodatno može zatvoriti i ekstremno brzo, slanjem *Reset* poruke (**RST**), koju ćemo opisati u sljedećem poglavlju (iako ova metoda nije standardna, ipak se koristi u određenim scenarijima za brzo zatvaranje veze).

Prvo ćemo objasniti standardno zatvaranje TCP veze u četiri koraka, prema slici dolje lijevo (221):

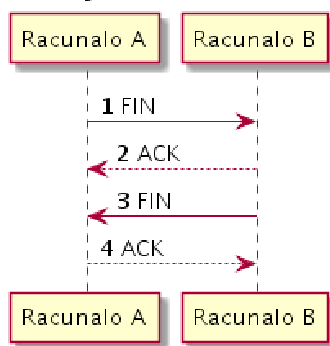
**1.** Kada jedna strana želi zatvoriti (svoju stranu) veze (**Računalo A**) ona prvo šalje `FIN` poruku. Ovdje nećemo ulaziti ponovno u detalje oko `sequence number` i `acknowledgment number` jer je sistem isti kao kod otvaranja veze i kao kod bilo koje druge TCP komunikacije.

**2.** Druga strana (**Računalo B**) odgovara slanjem poruke `ACK`. Nakon toga **Računalo A** čeka da `TCP Timeout` vrijeme istekne te potom zatvara svoju stranu veze. S ovom porukom je zatvoren jedan dio veze: **Računalo A → Računalo B**.
Od trenutka kada je primljena poruka `ACK` za zatvaranje veze, TCP port koji je otvoren za tu vezu još je otvoren, sve dok ne istekne TCP vremenski okvir (*TCP timeout*), i tek tada se on zatvara.

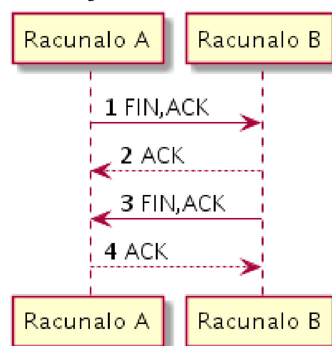
**3**. Sada druga strana (**Računalo B**) također šalje `FIN` paket prema prvoj strani (na **Računalo A**).

**4**. Potom druga strana ove komunikacije (**Računalo A**) potvrđuje prekid veze slanjem poruke `ACK` koju kada **Računalo B** primi, također čeka na TCP *timeout* (istek vremenskog okvira) te potom zatvara vezu (isto kao točka **2.**). Ovim korakom je zatvorena i druga strana veze: **Računalo B → Računalo A**, te sâmim time i cijela dvosmjerna TCP komunikacija tj. veza.

Moguće zatvaranje TCP veze u četiri koraka (slika 221) te zatvaranje TCP veze u četiri koraka na drugi način (slika 222).




Slika 221 Zatvaranje TCP veze u četiri koraka.



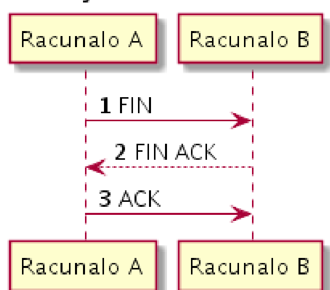

Slika 222. Zatvaranje TCP veze u četiri koraka.

**Pogledajmo i drugi scenarij zatvaranja TCP veze u četiri koraka, kako je vidljivo na slici 222.**
**1.** U prvom koraku prva strana (***Racunalo A***) šalje `FIN` za zatvaranje veze, ali i `ACK` poruku s kojom se potvrđuje primitak zadnjeg paketa, prije nego krećemo u zatvaranje.
**2.** Druga strana (***Računalo B***) potvrđuje zatvaranje s porukom `ACK` za primljeni paket iz koraka **1.**

**3.** Sada druga strana (***Računalo B***) također šalje zahtjev za zatvaranje svoje strane veze sa `FIN` za zatvaranje veze, ali i `ACK` s kojim potvrđuje zadnji primljeni paket iz koraka **2.**
**4.** Prva strana (***Računalo A***) potvrđuje zatvaranje sa `ACK`, čime potvrđuje i zadnji paket iz koraka **3**.

Postoji i brža metoda **zatvaranja veze u tri koraka**, koja je u praksi i najčešća metoda zatvaranja TCP veze.
Kod nje se događa slijedeće (kako je vidljivo na slici 223):

Slika 223. Zatvaranje TCP veze u tri koraka.




1. **Računalo A** šalje poruku `FIN` prema: **Računalo B.**
2. **Računalo B** odgovara sa porukom `FIN ACK` prema: **Računalo A.**
3. **Računalo A** odgovara sa porukom `ACK` prema: **Računalo B.**

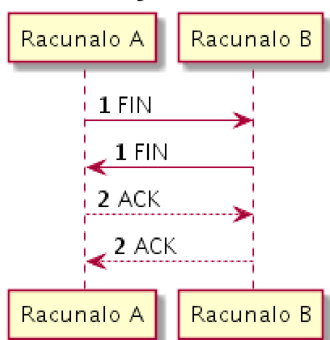
Nakon ovog koraka obje strane čekaju da TCP vremenski okvir (*timeout*) istekne te zatvaraju cijelu TCP vezu (obje strane komunikacije: A→B i B→A).

**Dodatno postoji i još brža metoda zatvaranja veze u dva koraka** kod koje oba računala istovremeno pošalju poruku `FIN` te nakon toga istovremeno potvrde drugoj strani zatvaranje, s porukom `ACK` kako je vidljivo na slici 224.



***U svakom slučaju, važno je znati da TCP veza može biti i napola otvorena odnosno napola zatvorena!.***

Slika 224. Zatvaranje TCP veze u dva koraka.




Naime moramo biti svjesni činjenice da je uspostavljena TCP veza uspostavljena zapravo s dvije strane, što se može promatrati kao dvije jednosmjerne veze:

- Strana A → strana B
- Strana B → strana A

Stoga je moguće da TCP veza ostane napola otvorena tj. "*half-open*" što se i događa kada jedna strana regularno zatvori vezu, a druga ne. Tada strana koja je zatvorila vezu ne može slati podatke na drugu stranu, ali druga strana može slati prema njoj, sva dok druga strana ne zatvori svoju stranu veze.

U svakom od slučajeva, nakon što je dobivena poruku o zatvaranju veze (**FIN**) od druge strane, čeka se određeni boj sekundi prije konačnog ***FIN*** paketa (obično 60.s.), prije prisilnog zatvaranja mrežnog *socketa* (tzv. *mrežne utičnice*). Ovo vrijeme definirano je u ***sysctl*** TCP varijabli: `net.ipv4.tcp_fin_timeout`.

**Pogledajte i sljedeće poglavlje s dodatnim primjerima!**

**Izvor informacija: (K-6),(1406), `man 7 tcp`, RFC793, RFC 9293.**

### 24.2.12. TCP reset

Transportni *TCP* protokol nudi nam mnoge mogućnosti održavanja i brige o konekcijama. Međutim ipak postoje slučajevi u kojima neka konekcija može ostati u polu zatvorenom stanju. Naime moguće je da jedna strana u komunikaciji zatvori svoju stranu komunikacije, bez da je o tome obavijestila drugu stranu. U ovakvom slučaju druga strana (koja nije svjesna problema) imat će otvorenu konekciju koja će biti u stanju `ESTABLISHED` dok će sa strane onoga koji je zatvorio konekciju bez obavijesti, konekcija biti u zatvorenom stanju (`CLOSED`) ili u nekom drugom prijelaznom stanju u trenutku zatvaranja. Ovakav slučaj se može dogoditi i ako je došlo do rušenja aplikacije, koja je zadužena za konekciju ili ako se aplikacija u tom trenutku restartala ili ostala "zaglavljena". Kako bi se *TCP* protokol mogao nositi s ovim problemom postoji posebna funkcija/opcija odnosno poruka za resetiranje konekcije (`TCP RST` poruka). Ova poruka radi tako da se u navedenom slučaju šalje *TCP* paket u kojem je uključena zastavica `RST` (*reset*).

Dakle poruka `TCP RST` se šalje kada se detektira nešto što ne pripada normalnom načinu funkcioniranja TCP komunikacije (**RFC 793**, **RFC 9293**), kao i cijeli niz drugih slučajeva:

- Ako je primatelj zaprimio *TCP SYN* poruku (poruku za otvaranje TCP komunikacije) na port na kojem nema aktivne *aplikacije* koja je odgovorna za komunikaciju. Primjerice zaprimljen je *TCP SYN* na portu 80 (*http*), a web poslužitelj koji bi trebao biti pokrenut na portu 80 nije pokrenut.
- Ako se zaprimi *TCP* segment (paket) od izvora s kojim nemamo ostvarenu *TCP* vezu, sve dok ta poruka nije *TCP SYN* poruka za otvaranje nove veze.
- Ako se poslužiteljska *aplikacija* u nekom trenutku srušila, a klijent i dalje šalje podatke prema njoj, poslužitelj šalje `RST`.
- Ako je primatelj zaprimio ispravan paket, ali u kojem su *Sequence Number* ili *Acknowledgment Number* brojevi koji ne odgovaraju tijeku komunikacije, odnosno koji:
  - Ili pripadaju nekoj drugoj TCP komunikaciji pa opće nemaju veze s ovom komunikacijom na bilo koji način.
  - Ili zbog primanja potvrdnog paketa (s *Acknowledgment Number*om) koji je već davno potvrđen.
- Slučaj u kojem je veza u stanju zatvaranja, ali je još u **Time-Wait** stanju i ako se tada zaprimi zakašnjeli ACK (ili čak SEQ) paket na strani poslužitelja, on će poslati TCP RST klijentu jer je veza s njegove točke gledišta u stanju zatvaranja (iako je još u ***Time-Wait*** stanju).
- Ako primatelj trenutno ne želi odgovoriti na zahtjev za novom vezom odnosno na *TCP SYN* poruku. Ovo se može dogoditi, ako je primatelj preopterećen na što će indicirati: da mu je pred memorija [***queue***] prepunjena ili mu je TCP pred memorija prepunjena (potencijalni ***TCP buffer overflow***) ili kod nekih specijaliziranih tranzitnih uređaja; primjerice *Proxy/Load Balancer/Web poslužitelj*, jer je detektirano preveliko zaglavlje paketa; primjerice ***HTTP zaglavlje*** (*HTTP header*).
- Slanjem TCP RST poruke zagušeni mrežni uređaji mogu učinkovito resetirati TCP vezu, uzrokujući prekid oba kraja veze, što pomaže u sprječavanju daljnjeg zagušenja komunikacije i omogućuje oporavak mrežne veze (**RFC 5681**).
- Ako poslužitelj ima vatrozid koji ne dozvoljava spajanje konkretnom klijentu; primjerice zbog IP adrese s koje dolazi (zbog prava pristupa [**ACL** liste]).
- Ako ispred računala imamo tranzitni uređaj koji radi ***NAT*** (usmjerivač ili vatrozid), a koji može odbaciti njemu zastarjele TCP veze slanjem TCP RST poruke. Uzrok može biti i zbog greške/*buga* ili problema s tranzitnim uređajem.
- Ako tranzitni uređaj, pr. usmjerivač/vatrozid/*Load balancer/IDS/IPS/Proxy*, radi preusmjeravanje na drugu adresu ili ako je broj ostvarenih konekcija dosegao definirano ograničenje (ako je to ograničenje postavljeno na uređaju [ili sustavu]).
- Slanje velikog broja TCP RST poruka indicira takozvani TCP Reset napad (slanjem enormnog broja TCP RST poruka).



*U svakom slučaju kada druga strana zaprimi `TCP RST` poruku ona odmah i bezuvjetno zatvara* TCP *vezu, bez potrebe za potvrdom o zatvaranju veze poput normalnog zatvaranja sa `TCP FIN` porukom ili bilo kakve potvrde sa `TCP ACK`, koju je potrebno potvrditi. U određenim slučajevima se i tijekom normalnog zatvaranja konekcije koristi `TCP RST` jer je brža metoda i od `FIN ACK` + `FIN ACK` metode zatvaranja* TCP *konekcije.*



***TCP RST** poruka može biti vidljiva i kao **TCP RST ACK** pri čemu se ovaj **ACK** ne odnosi na potvrdu **RST** dijela već se on odnosi na potvrdu prijašnjeg paketa, jer se **RST** poruka nikada ne potvrđuje s **ACK** porukom.*

Ako se **`TCP RST`** poruka s IP adrese: 192.168.1.162 šalje na IP:192.168.1.231, na port 80 (*HTTP*) to može izgledati ovako:

| *SRC IP* | *DST IP* | | *SRC PORT →* | *DST PORT* | *PORUKA:* |
|---|---|---|---|---|---|
| 192.168.1.162 | 192.168.1.231 | TCP | 27052 → | 80 | [RST, ACK] Seq=1 Ack=1 |



*Load Balanceri* poput primjerice: **HAProxy** ili **Keepalived** servisa, zbog provjere dostupnosti servisa na poslužiteljima koji se prate (jesu li dostupni ili nisu), prema njima mogu slati i takozvane ***TCP probe*** poruke. Ovim porukama *Load Balancer* šalje TCP paket: **TCP SYN** prema odredišnoj IP adresi i TCP portu, na koji suprotna strana, ako joj je servis živ na tom portu, odgovara s **TCP SYN ACK** te konačno *Load Balancer* potvrđuje s **TCP ACK**. S time se zapravo klasično otvara TCP konekcija. Međutim on tada može poslati ***TCP RST** kako bi brzo zatvorio tu TCP konekciju. Pošto mu ona više ne treba do sljedeće provjere, jer kroz nju ionako neće teći nikakav promet. *Load Balanceri* šalju ovakve provjere svakih nekoliko sekundi (konfigurabilno) te svaki puta nakon uspostavljene veze šalju **TCP RST** što je najbrža metoda zatvaranja TCP veze.

Komunikacija u kojoj *Load Balancer* šalje ***TCP probe*** poruke može izgledati ovako:

| *SRC IP* | *DST IP* | | *SRC PORT → DST PORT* | *PORUKA:* |
|---|---|---|---|---|
| 192.168.1.162 | 192.168.1.231 | TCP | 27052 → 80 | [SYN] Seq=0 |
| 192.168.1.231 | 192.168.1.162 | TCP | 80 → 27052 | [SYN, ACK] Seq=0 Ack=1 |
| 192.168.1.162 | 192.168.1.231 | TCP | 27052 → 80 | [ACK] Seq=1 Ack=1 |
| 192.168.1.162 | 192.168.1.231 | TCP | 27052 → 80 | [RST, ACK] Seq=1 ACK=1 |

Ovdje vidimo kako *Load Balancer* s IP adresom: 192.168.1.162 šalje ***TCP Probe*** (tzv. ***health check***) poruku otvaranjem TCP veze s porukom: TCP `[SYN]` (prva poruka) prema TCP portu 80 (HTTP servis) odredišnog web poslužitelja s IP: 192.168.1.231. Potom nam odredišni servis (konkretno HTTP poslužitelj sa TCP porta 80) s IP adrese: 192.168.1.231 odgovara s TCP `[SYN ACK]` (druga poruka u nîzu).

Tada, što je standardno za uspostavljenje TCP veze, mi kao *Load Balancer* s IP adrese: 192.168.1.162 odgovaramo s potvrdom o otvaranju ove TCP veze, s porukom TCP `[ACK]`. S time je TCP veza otvorena odnosno uspostavljena: između *Load Balancera* i Web poslužitelja na TCP portu 80. Odmah potom slijedi četvrta poruka u kojoj *Load Balancer* šalje poruku za resetiranje veze TCP `[RST, ACK]`. *S time da `ACK` dio poruke nema direktne veze s `RST`* porukom jer se `RST` ne potvrđuje, već se potvrđuje sâmo primitak zadnje TCP poruke (konkretno poruke broj tri).*

*Međutim moguć je i ovakav scenarij otvaranja tzv. poluotvorene (**TCP half open**) veze koja se brzo zatvara s **RST** porukom:*

| *SRC IP* | *DST IP* | | *SRC PORT → DST PORT* | *PORUKA:* |
|---|---|---|---|---|
| 192.168.1.162 | 192.168.1.231 | TCP | 27052 → 80 | [SYN] Seq=0 |
| 192.168.1.231 | 192.168.1.162 | TCP | 80 → 27052 | [SYN, ACK] Seq=0 Ack=1 |
| 192.168.1.162 | 192.168.1.231 | TCP | 27052 → 80 | [RST, ACK] Seq=1 Ack=1 |

Ovakvu metodu također često koriste uređaji poput *Load Balancera* za provjeru dostupnosti servisa, jer troši još manje resursa.

Pogledajmo sliku 224.1 snimljenih samo *TCP probe* poruka koje šalje *Load Balancer* prema navedenom *Web* poslužitelju:

**Slika 224.1. TCP Probe poruke.**

TCP Probe.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000 | 192.168.1.162 | 192.168.1.231 | TCP | 74 | 27014 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3793227854 TSecr=0 WS=128 |
| 2 | 0.000386 | 192.168.1.231 | 192.168.1.162 | TCP | 74 | 80 → 27014 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2291144973 TSecr=3793227854 WS=128 |
| 3 | 0.000419 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27014 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793227854 TSecr=2291144973 |
| 4 | 0.000478 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27014 → 80 [RST, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793227854 TSecr=2291144973 |
| 5 | 7.423247 | 192.168.1.162 | 192.168.1.231 | TCP | 74 | 27016 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3793235277 TSecr=0 WS=128 |
| 6 | 7.423635 | 192.168.1.231 | 192.168.1.162 | TCP | 74 | 80 → 27016 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2291152397 TSecr=3793235277 WS=128 |
| 7 | 7.423654 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27016 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793235278 TSecr=2291152397 |
| 8 | 7.423753 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27016 → 80 [RST, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793235278 TSecr=2291152397 |
| 9 | 17.425189 | 192.168.1.162 | 192.168.1.231 | TCP | 74 | 27020 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3793245279 TSecr=0 WS=128 |
| 10 | 17.425624 | 192.168.1.231 | 192.168.1.162 | TCP | 74 | 80 → 27020 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2291162399 TSecr=3793245279 WS=128 |
| 11 | 17.425649 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27020 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793245280 TSecr=2291162399 |
| 12 | 17.425746 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27020 → 80 [RST, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793245280 TSecr=2291162399 |
| 13 | 27.426607 | 192.168.1.162 | 192.168.1.231 | TCP | 74 | 27024 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3793255281 TSecr=0 WS=128 |
| 14 | 27.426973 | 192.168.1.231 | 192.168.1.162 | TCP | 74 | 80 → 27024 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2291172400 TSecr=3793255281 WS=128 |
| 15 | 27.426998 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27024 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793255281 TSecr=2291172400 |
| 16 | 27.427114 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27024 → 80 [RST, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793255281 TSecr=2291172400 |
| 17 | 37.438371 | 192.168.1.162 | 192.168.1.231 | TCP | 74 | 27028 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3793265292 TSecr=0 WS=128 |
| 18 | 37.438781 | 192.168.1.231 | 192.168.1.162 | TCP | 74 | 80 → 27028 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2291182412 TSecr=3793265292 WS=128 |
| 19 | 37.438806 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27028 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793265293 TSecr=2291182412 |
| 20 | 37.438915 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27028 → 80 [RST, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793265293 TSecr=2291182412 |
| 21 | 47.440229 | 192.168.1.162 | 192.168.1.231 | TCP | 74 | 27032 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3793275294 TSecr=0 WS=128 |
| 22 | 47.440605 | 192.168.1.231 | 192.168.1.162 | TCP | 74 | 80 → 27032 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2291192414 TSecr=3793275294 WS=128 |
| 23 | 47.440629 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27032 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793275295 TSecr=2291192414 |
| 24 | 47.440730 | 192.168.1.162 | 192.168.1.231 | TCP | 66 | 27032 → 80 [RST, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3793275295 TSecr=2291192414 |

Dakle, vidimo mrežne pakete s kojima se otvara TCP veza:

- **SYN** – primjerice paket broj **1.** (vidljivo u stupcu `No.`)
- **SYN**, **ACK** – primjerice paket broj **2.** (vidljivo u stupcu `No.`)
- **ACK** – primjerice paket broj **3.** (vidljivo u stupcu `No.`)

A odmah potom vidimo da se TCP sesija hitno zatvara s *Reset* porukom (**RST**):

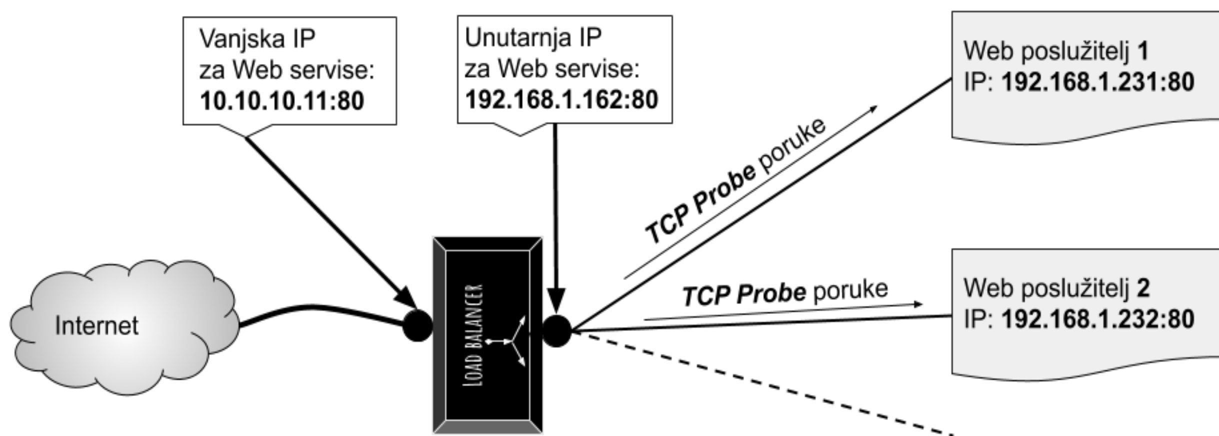
- **RST**, **ACK** – primjerice paket broj **4.** (vidljivo u stupcu `No.`)

To se konkretno stalno ponavlja svakih desetak sekundi jer se svako toliko i šalju *TCP probe* poruke:

- **Prva** *TCP probe* **poruka** – navedeni paketi **1.** – **4.** (vrijeme **0.s.** [vidljivo u stupcu `Time`]).
- **Druga** *TCP probe* **poruka** – paketi **5.** – **8.** (vrijeme **~7.s.** [vidljivo u stupcu `Time`]).
- **Treća** *TCP probe* **poruka** – paketi **9.** – **12.** (vrijeme **~17.s.** [vidljivo u stupcu `Time`]).
- . . .

Pogledajmo ovaj način rada odnosno logičku shemu slanja *TCP probe* poruka.

**Slika 224.2. TCP Probe poruke koje šalje Load Balancer prema web poslužiteljima kako bi periodički provjeravao jesu li dostupni (*živi*).**






Pogledajte i poglavlja:
**23.4.5. Redundancija na OSI sloju tri (OSI 3) i *VRRP* protokol.**
**24.2.9. Nadzor zagušenja (*Congestion control*).**

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju.
Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju zatvaranja TCP veze:

| Naziv brojača | Opis |
|---|---|
| **`TcpEstabResets`** | Broj puta kada su TCP veze izvršile izravan prijelaz u stanje CLOSED iz stanja ESTABLISHED ili stanja CLOSE-WAIT. |
| **`TcpOutRsts`** | Broj poslanih TCP segmenata koji sadrže RST oznaku. |
| **`TcpExtTCPClose`** | Ovaj brojač bilježi broj TCP veza koje su zatvorene. Kada TCP veza završi komunikaciju i zatvori se putem razmjene **FIN** paketa, to će rezultirati povećanjem ovog brojača. |
| **`TcpExtTCPAbortOnTimeout`** | Ovaj brojač bilježi broj TCP veza koje su prekinute zbog isteka vremenskog ograničenja (*timeout*). To može uključivati slučajeve kada je TCP konekcija ostala neaktivna i nije bilo aktivnosti, što može dovesti do automatskog prekida veze. |
| **`TcpExtTCPAbortOnClose`** | Ovaj brojač bilježi broj TCP veza koje su prekinute tijekom zatvaranja veze. To može uključivati slučajeve kada je zatvaranje veze prekinuto ili nije uspjelo iz nekog razloga. |
| **`TcpExtTCPReset`** | Ovaj brojač bilježi broj TCP RST paketa koji su poslani ili primljeni. TCP RST (*reset*) paketi se koriste za trenutno prekidanje TCP veze, bez očekivanja zatvaranja veze putem razmjene FIN paketa. Povećanje ovog brojača ukazuje na prekid veze putem TCP RST paketa. |
| | |
| **`TcpExtTCPFastRetrans`** | Iako nije izravno povezan s zatvaranjem veze, brojač `TcpExtTCPFastRetrans` bilježi broj TCP segmenata koji su brzo ponovno poslani (*retransmitirani*) kao odgovor na brzi *retransmit* događaj, što može biti dio zatvaranja veze u situacijama kada se dogode gubici podataka ili problemi u mreži. |

**Izvor informacija:(1482),(1483),(1651), `man 7 tcp`,`man nstat`, RFC 793, RFC 5681, RFC 7230, RFC 9293.**

## 24.2.13. Mehanizam TCP retransmisije

*TCP* kao transportni protokol brine se o tome kako bi prijenos podataka ostao pouzdan. To se postiže tako što primaoc paketa mora pošiljatelju potvrditi da je primio paket. Pošto svaki paket ima svoj identifikacijski broj (*sequence number*) na osnovu njega se i potvrđuju zaprimljeni paketi. Problem može biti gubitak primljenih paketa, ali i gubitak potvrdnih paketa s kojima pošiljaocu potvrđujemo primitak paketa s porukom `TCP ACK` **XY** (**XY**=broj paketa).
Ovi problemi rješavaju se definiranjem vremenskih okvira (Engl *Timeout*) unutar kojih primatelj mora potvrditi zaprimljeni paket. Ako pošiljatelj ne zaprimi potvrdu o primitku određenog paketa (ili nîza paketa), taj paket (ili taj nîz paketa) se ponovno odašilje.
U ovom mehanizmu važnu ulogu igraju i vremenski okviri i strategija *retransmisije* odnosno ponovnog slanja paketa.

Pri tome postoji nekoliko osnovnih vrsta ponovnog slanja odnosno *retransmisija* paketa:

- **TCP Retransmission** odnosno klasična retransmisija koja se događa kada je istekao vremenski brojač (*timer*) odnosno vrijeme unutar kojeg je paket morao biti potvrđen, pa se paket ponovno šalje na odredište.

- **TCP Fast Retransmission** se događa u nekoliko slučajeva (ako je ova opcija uopće uključena):
    - Standardno kada pošiljatelj pošalje pojedini paket, a vrijeme (funkcija od vremena „*round-trip delay time*“) potrebno da dobije potvrdu o primitku paketa je isteklo, a potvrda nije stigla od primaoca paketa. Pošiljaoc odmah ponovno šalje taj paket, bez čekanja na primaočevu poruku o potvrdi primitka tog paketa.

    - **TCP duplicate ACK** odnosno slučaj primanja dupliciranih `ACK` poruka na strani pošiljaoca. Ovaj slučaj događa se kada primaoc zaprimi pakete pravilnim redoslijedom poput: 1,2,3, a potom zaprimi pakete 6,7,8. Dakle nedostaju paketi 4 i 5. U tom slučaju primatelj u trenutku kada je zaprimio prvi paket koji nije u nizu, a to bi ovdje bio paket broj 6, šalje pošiljatelju `ACK` poruku s kojom potvrđuje sâmo primitak paketa broj 3. Potom kada primi paket broj 7 on ponovno potvrđuje sâmo primitak paketa broj 3 jer je on zadnji koji je primljen ispravnim redoslijedom. Nadalje kada zaprimi i paket broj 8, on će ponovno pošiljatelju potvrditi samo paket broj 3. Pošiljatelj će u jednom trenutku zaprimiti tri (3) potvrde (`TCP ACK`) za paket broj 3. Dakle primit će duplicirane `ACK` poruke nakon kojih će poslati pakete u nîzu, počevši od paketa broj 4. Prema **RFC 2001** standardu, pošto *TCP* sloj na strani pošiljatelja inicijalno kada zaprimi tek dvije duplicirane `ACK` poruke (za isti segment/paket) smatra kako je potencijalno tek došlo do gubitka paketa jer se mogla dogoditi greška u kojoj su poslane dvije `TCP ACK` poruke za isti segment/paket. Ako ipak pošiljatelj zaprimi tri ili više dupliciranih `TCP ACK` poruka, to je jasna indikacija kako je došlo do
    primanja paketa koji nisu u nizu (1,2,3,4,5,…) te se odmah šalju paketi koje je potrebno poslati u nizu, bez čekanja na istjecanje vremenskih okvira (*timera)*.

- **TCP Spurious Retransmission** je posebna vrsta *retransmisije* kod koje, iako je strana koja prima podatke potvrdila konkretan paket (***ACK***), strana koja odašilje ipak taj konkretan paket identificira kao izgubljen, te ga šalje ponovno. To se može dogoditi, ako pošiljatelj nije dobio potvrdnu **ACK** poruku pa kao što i treba (standardna *retransmisija*): nakon isteka ***RTO*** vremena ponovno šalje isti paket, koji može biti detektiran kao *spurious* sa strane primatelja jer je on u tom slučaj prethodno već mogao poslati ***ACK***, koji je možda izgubljen pa niti nije mogao doći do strane koja odašilje. Drugi uzrok može biti sljedeći: pošiljatelj je uredno dobio ***ACK*** paket, ali on u određenom vremenskom okviru (***RTO***) nije stigao biti obrađen od strane TCP/IP stôga (pr. preopterećen CPU). Zbog toga se ponavlja prvi scenarij: ponovno slanje paketa (*retransmisija*), iako ga je primatelj zapravo potvrdio sa ***ACK*** porukom.
Dakle ovdje se radi o ponovno slanju paketa koji je već uredno zaprimljen i potvrđen, što indicira neke greške u mreži ili na strani koja šalje podatke (pakete).

U klasičnoj implementaciji *TCP* protokola kada je paket (segment) izgubljen, mora ga se ponovno poslati (*retransmisija*) kao i sve pakete poslije njega koji su ispravno zaprimljeni. Dakle zaprimaju se sâmo paketi koji dolaze u točnom nizu; primjerice: 1,2,3,4,5,….

Druga naprednija metoda su selektivne potvrde (**SACK** odnosno selektivni `ACK`) za koju **pogledajte poglavlje: 24.2.6.**
Pri tome **SACK** metodu moraju podržavati obje strane u komunikaciji. **SACK** je definiran u standardu **RFC 2018** te nadograđen u **RFC 2883**.

Statistiku *retransmisija* paketa možemo vidjeti s naredbom `netstat` na sljedeći način:

```
netstat -s | grep -i retrans

    941513 segments retransmited
    TCPLostRetransmit: 30
    818977 fast retransmits
    95291 forward retransmits
    14827 retransmits in slow start
    8053 SACK retransmits failed
```

Statistike možemo vidjeti i pomoću naredbe ss na sljedeći način:

```
ss -i
```

Dodatne optimizacije iz ovog područja mogu se postići sa ***sysctl*** varijablom: `net.ipv4.tcp_retrans_collapse`.



Pogledajte i poglavlja:
**24.2.4. Trajanje TCP veze.**
**24.2.15. TCP keepalives.**
**25.3 Statistike, analiza i praćenje mrežnih paketa.**
**25.7.4 *Naredba netstat*.**
**25.7.7 Naredba socket statistics (*ss*).**

Pogledajte i poglavlje:
**24.2.12. TCP reset** odnosno primjer sa slanjem ***TCP probe*** poruka za provjeru dostupnosti druge strane.

Pogledajmo kako pomoću naredbe nstat (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju.

Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju raznih retransmisija TCP veze:

| Naziv brojača | Opis |
|---|---|
| **TcpExtTCPLostRetransmit** | Ovaj brojač bilježi ukupan broj izgubljenih *retransmitiranih* TCP segmenata. Izgubljeni segmenti su oni koji nisu uspjeli stići do odredišta iako su retransmitirani kao odgovor na izostanak potvrde o primitku (ACK) za prethodni segment. |
| **TcpExtTCPFastRetrans** | Ovaj brojač bilježi broj TCP segmenata koji su brzo retransmitirani (ponovno poslani). **Fast retransmit** je mehanizam koji se aktivira kada TCP pošiljatelj primi više duplikata ACK (potvrda o primitku) za isti segment, što može ukazivati da je taj segment izgubljen u mreži |
| **TcpExtTCPSlowStartRetrans** | Ovaj brojač bilježi broj TCP segmenata koji su retransmitirani tijekom faze "*slow start*". ***Slow start*** je jedan od mehanizama za kontrolu zagušenja u TCP-u, gdje se postupno povećava broj poslanih segmenata. Retransmisija tijekom "*slow start*" faze događa se kada su izgubljeni segmenti ili nije stigla potvrda o primitku (ACK) za prethodni segment. |
| **TcpExtTCPRetransFail** | Ovaj brojač bilježi broj neuspjelih retransmisija TCP segmenata. Neuspjele retransmisije nastaju kada TCP pošiljatelj ne uspije ponovno poslati izgubljeni segment ili kada ponovno poslani segmenti ponovno budu izgubljeni u mreži. |
| **TcpExtTCPSynRetrans** | Ovaj brojač bilježi broj TCP SYN segmenata koji su retransmitirani. TCP SYN segmenti su korišteni tijekom uspostavljanja TCP veze. Retransmisija SYN segmenata može se dogoditi ako inicijalni SYN segment nije stigao na odredište ili ako odgovor to jest SYN+ACK segment nije stigao na vrijeme |

**Izvor informacija:(K-6),(1406),(1651),** `man 7 tcp`,`man nstat`, **RFC793, RFC2001, RFC2018.**

## 24.2.14. Primitak paketa izvan redoslijeda slanja (*Out-Of-Order*)

Iako TCP/IP na strani primatelja očekuje pakete u istom redoslijedu kako su i poslani, u određenim slučajevima moguće je zaprimiti pakete izvan tog redoslijeda. U tim problematičnim situacijama (koje će TCP/IP stôg Linuxa uredno riješiti), ako analiziramo mrežni promet (pakete) s programima poput ***Wireshark***, vidjet ćemo pakete koji imaju oznaku: ***Out-Of-Order***.

Koji sve uvjeti moraju biti zadovoljeni kako bi paket bio označen kao paket koji dolazi izvan slijeda slanja odnosno kao takozvani ***Out-Of-Order***:

- Prvi uvjet koji se mora podudarati je da primljeni segment (paket) nema sekvencijski broj (**SEQ**) koji je jednak ili veći od sljedećeg očekivanog sekvencijskog (**SEQ**) broja paketa.
- Zatim se provjerava radi li se o takozvanoj „**Fast retransmission**" metodi (pogledajte prijašnje poglavlje).
- Potom se uspoređuje razlika vremena između ovog segmenta (paketa) i vremena segmenta (paketa) s najvišim zaprimljenim sekvencijskim (**SEQ**) brojem. Ako je ta razlika manja od početnog vremena prolaska paketa tj. **RTT** vremena, to indicira da se ne može raditi o retransmisiji. Ili ako je razlika vremena manja od 3 ms kada uspostava TCP veze u tri koraka nije zabilježena, to se sve smatra paketom koji je zaprimljen izvan redoslijeda slanja, odnosno koji će biti označen kao: ***Out-Of-Order***, ako ti paketi imaju isti **SEQ** broj ili, ako je prvi uvjet zadovoljen.



Prije nego krenemo dalje, dobro proučite kako se uspostavlja TCP veza i primjere **SEQ** i **ACK** brojeva:
**24.2.2. Kako se uspostavlja TCP veza.**

Pogledajmo jedan primjer snimljenih paketa u programu ***Wireshark***, na slici 224.3:

**Slika 224.3. TCP Out-of-Order i TCP Retransmission paketi**

| | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 89 | 12.478644 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 TSval=2587437 TSecr= |
| 90 | 12.478656 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Retransmission] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 |
| 91 | 12.478665 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Retransmission] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 |
| 261 | 12.828605 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Out-Of-Order] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 T |
| 262 | 12.828617 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Out-Of-Order] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 T |
| 263 | 12.828625 | 172.27.9.241 | 192.168.1.11 | TCP | 1424 | [TCP Out-Of-Order] 80 → 52651 [ACK] Seq=1302655196 Ack=4220141780 Win=8009 Len=1358 T |

**Izvor fotografije: (931)**

Na prethodno slici imamo slučaj gdje je paket broj **89** ispravno zaprimljen i ima: **SEQ**=1302655196 i **ACK**=4220141780 oznake. Potom u paketima brojeva: **90** i **91** dolazi do *retransmisije* istog paketa što je legitimno u slučaju gubitka istog, jer se tada radi *retransmisija* odnosno ponovno slanje paketa koji nije potvrđen.
Nakon toga, a tek nakon nekog vremena, odnosno tek u paketu **261** ponovno primamo isti paket s istim **SEQ**=1302655196 i **ACK**=4220141780 oznakama. Što je očito detektirano kao paket izvan redoslijeda slanja, odnosno ***Out-Of-Order*** jer se sada očekuje da **SEQ** brojevi moraju biti veći od: **SEQ**=1302655196 jer je prošlo dovoljno vremena, odnosno više od **RTT** vremena pa su zadovoljeni svi gore navedeni uvjeti.

Pogledajmo i sljedeću komunikaciju odnosno primjer (ispis smo skratili za određena polja i izbacili smo neke mrežne pakete):

```
5514  10.10.10.1 10.10.10.2 TCP 317 16611 → 48337 [ACK] Seq=28376 Ack=1185
5515  10.10.10.1 10.10.10.2 TCP 543 16611 → 48337 [ACK] Seq=28631 Ack=1185
5524  10.10.10.1 10.10.10.2 TCP 317 [TCP Retransmission] 16611 → 48337 [ACK] Seq=28376 Ack=1185
5528  10.10.10.1 10.10.10.2 TCP 74  [TCP Dup ACK 5525#1] 48337 → 16611 [ACK] Seq=1185 Ack=28631
5529  10.10.10.1 10.10.10.2 TCP 543 [TCP Out-Of-Order] 16611 → 48337 [ACK] Seq=28631 Ack=1185
```

Naime ovdje vidimo (radi se o lokalnoj mreži s ***RTT*** ispod 0.1*ms*) da smo zaprimili paket broj `5515` koji ima `Seq=28631` i `Ack=1185`, pa imamo nîz od nekoliko drugih paketa.

Nakon nekog vremena (`42.99911-42.79478=0,20433 s = 204ms`) dolazi nam paket broj: `5529`. On ima iste: `Seq=28631` i `Ack=1185` brojeve kao i paket broj: `5515`.

Pošto je prošlo dovoljno vremena (više od trenutnog **RTT**), ovaj paket je označen da dolazi izvan redoslijeda slana, odnosno označen je kao:

***Out-Of-Order*** jer se očekuje da mu je sekvencijski (**SEQ**) broj definitivno veći od: `28631`.

## Često zaprimanje paketa izvan redoslijeda slanja

Ako se zaprimanje paketa izvan redoslijeda slanja (***Out-Of-Order***) događa često ili stalno, moguće je da su uzroci sljedeći:

- Postoji više putova do odredišta: paketi izvan redoslijeda slanja mogu nastati zbog tokova podataka koji slijede više staza kroz mrežu; poput prometa koji putuje putem Interneta i prolazi kroz razne usmjerivače od kojih se u nekom trenutku promet može prebaciti na neki drugi usmjerivač u nîzu (pr. zbog ispada jednog od njih). Stoga paketi na odredište mogu dolaziti izvan redoslijeda kojim su poslani (barem u trenutku kada se prebacivanje dogodilo).

- **Queuing mehanizmi** odnosno mehanizmi redova čekanja: naime loše konfiguriranim redovima čekanja na tranzitnim uređajima, primjerice usmjerivačima, ili čak asimetričnim konfiguracijama usmjeravanja, kada mrežni uređaj ima konfiguriran red čekanja koji ne prosljeđuje pakete po istom redoslijedu dolaska.

- Odnosno u slučaju kada se koristi prevelika međumemorija te dolazi do takozvanog ***Bufferbloat***-a.



Vezano za ***queuing*** i ***bufferbloat*** pogledajte poglavlja:
**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**
**25.1.4.1. Bufferbloat.**

- **Agregacija veza** i ujednačavanje opterećenja (engl. *Load balancing*) mogu uzrokovati primitak paketa izvan slijeda slanja, pogotovo kod ujednačavanja opterećenja, ako se koristi algoritam koji se temelji na *Round-Robin* mehanizmu.



Vezano za **agregaciju** veza (*bonding/teaming*), pogledajte poglavlje:
**20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).**

Odnosno za ujednačavanje opterećenja (*Load Balancing*), pogledajte poglavlje:
**23.4.5.3. Balansiranje opterećenja (*Load Balancing*) – ručni rad.**

- Uzrok može biti i u tranzitnim uređajima koji rade fragmentiranje i defragmentiranje paketa.



Vezano za fragmentaciju paketa, pogledajte poglavlje:
**23.2. IP fragmentacija.**

- Uzrok mogu biti i preopterećeni uređaji ili računala: preopterećeni tranzitni mrežni uređaji (usmjerivači, vatrozidi, preklopnici) u slučaju većeg preopterećenja mogu odbacivati pakete, što u konačnici može uzrokovati dolazak paketa izvan slijeda slanja. U slučaju virtualizacije, ako je *hipervizor* preopterećen (ili namjerno prepunjen [***overprovisioning***]), on također može uzrokovati ovakve probleme.



Uzroci preopterećenja mogu biti i loša konfiguracija uređaja: počevši od mrežnih opcija i parametara, parametara upravljačkog programa, preko mrežnog stôga, do svih drugih komponenti uključenih u mrežni sustav!



Vezano za moguće uzroke, pogledajte poglavlje:
**25.3. Statistike, analiza i praćenje mrežnih paketa.**

## TCP Reordering

Važno je razumjeti da je posljedica svih navedenih uzroka, dolazak paketa na odredište izvan redoslijeda slanja.
Na strani primatelja, TCP stôg koji ih prima trebao bi ih prvo spremiti u međuspremnik i promijeniti njihov redoslijed prije prosljeđivanja podataka aplikacijskom sloju i u konačnici aplikacijama.
Linux definira maksimalan broj paketa u jednom TCP toku koji se tako mogu presložiti.

Optimiziranje TCP stôga po pitanju preslaganja (engl. *reordering*) paketa može se postići sa ***sysctl*** varijablom: `net.ipv4.tcp_reordering`. Vrijednost ove varijable govori kernelu odnosno mrežnom stôgu koliko se TCP paketa unutar određenog tôka može presložiti bez pretpostavke da je paket izgubljen negdje na putu. Standardno je postavljena vrijednost 3. To znači da ako se zaprimi veći broj paketa izvan redoslijeda slanja, da će se pretpostaviti gubitak paketa i zatražiti (TCP) retransmisija.

Druga ***sysctl*** varijabla je: `net.ipv4.tcp_max_reordering`. Ona označava maksimalnu razinu preslagivanja paketa u određenom TCP toku. Zadana je vrijednost 300 i ona je prilično konzervativna vrijednost, ali možete je povećati ako se TCP tok koristi za uravnoteženje opterećenja (primjerice upotrebom *Round Robin* načina rada).
*TCP Reordering* se može vidjeti u statistikama koje dobivamo s naredbama: `ss --info` ili s naredbom: `netstat -s`.



Za pronalazak uzroka ovakvog ponašanja, potrebno je analizirati rad svih tranzitnih mrežnih uređaja između nas i pošiljatelja, kao i opterećenje našeg računala (ili ***hipervizora***, ako nam je računalo virtualno).

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju. Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju zatvaranja TCP veze:

| Naziv brojača | Opis |
|---|---|
| **`TcpExtTCPOFOQueue`** | Ovaj brojač bilježi broj TCP segmenta koji su stavljani u red čekanja (*queue*) zbog dolaska izvanrednih (*out-of-order*) segmenata. Kada TCP primatelj primi segmente koji nisu u očekivanom redoslijedu, on ih često stavlja u red čekanja dok se ne primi očekivani segment koji slijedi, kako bi se moglo rekonstruirati ispravno slijed podataka. |
| **`TcpExtTCPOFODrop`** | Ovaj brojač bilježi broj TCP segmenta koji su odbačeni zbog dolaska izvanrednih (*out-of-order*) segmenata. Kada red čekanja postane prevelik ili ako izvanredni segmenti postanu zastarjeli, TCP primatelj može odlučiti odbaciti neke od tih segmenata kako bi spriječio prekomjerno opterećenje i osigurao učinkovitost komunikacije. |
| **`TcpExtTCPOFOMerge`** | Ovaj brojač bilježi broj izvanrednih (*out-of-order*) segmenata koji su spojeni (*merged*) s drugim segmentima u red čekanja. Kada se primatelju isporuče izvanredni segmenti, TCP primatelj može pokušati spojiti te segmente s postojećim segmentima u red čekanja kako bi se smanjio broj segmenata koji čekaju na obradu i poboljšala učinkovitost komunikacije. |
| Brojači koji su indirektno povezani s *out-of-order* slanjem | |
| **`TcpExtTCPDSACKRecv`** | Ovaj brojač bilježi broj primljenih TCP DSACK (*Duplicate Selective Acknowledgment*) potvrda. DSACK potvrde se koriste kako bi se označili duplikati podataka koji su primljeni kao izvanredni segmenti. |
| **`TcpExtTCPDSACKOldSent`** | vaj brojač bilježi broj puta kada su stariji (old) podaci ponovno poslani (*resent*) na temelju primljene DSACK potvrde. |
| **`TcpExtTCPDSACKOfoSent`** | Ovaj brojač bilježi broj puta kada su podaci poslani ponovno (*resent*) na temelju primljene DSACK potvrde koja se odnosi na segmente koji su izvan okvira slanja (*out-of-order*). |
| **`TcpExtTCPSACKReorder`** | Ovaj brojač bilježi broj puta kada su podaci poslani ponovno (*resent*) zbog reorganizacije izvanrednih (*out-of-order*) segmenata na temelju TCP SACK potvrda. |

**Izvori informacija: (K-6), (929),(930),(931),(1536),(1651).** **`man ss`, `man netstat`,`man nstat`**.

## 24.2.15. TCP keepalives

*TCP keepalive* poruke, definirane su u standardu **RFC 1122**. Kada su dva računala povezana preko mreže putem *TCP/IP* protokola, odnosno kada je među njima otvoren *TCP* komunikacijski kanal, ***TCP Keepalive*** paketi mogu se koristiti i za utvrđivanje je li veza još uvijek valjana i po potrebi ju prekinuti. Međutim primarna funkcionalnost ovog mehanizma jest održavanje veze (*TCP* komunikacijskog kanala) živim, u slučaju kada na njoj nema prometa duže vremena, jer bi se tada veza zatvorila zbog isteka TCP vremenskih okvira za neaktivnu vezu.

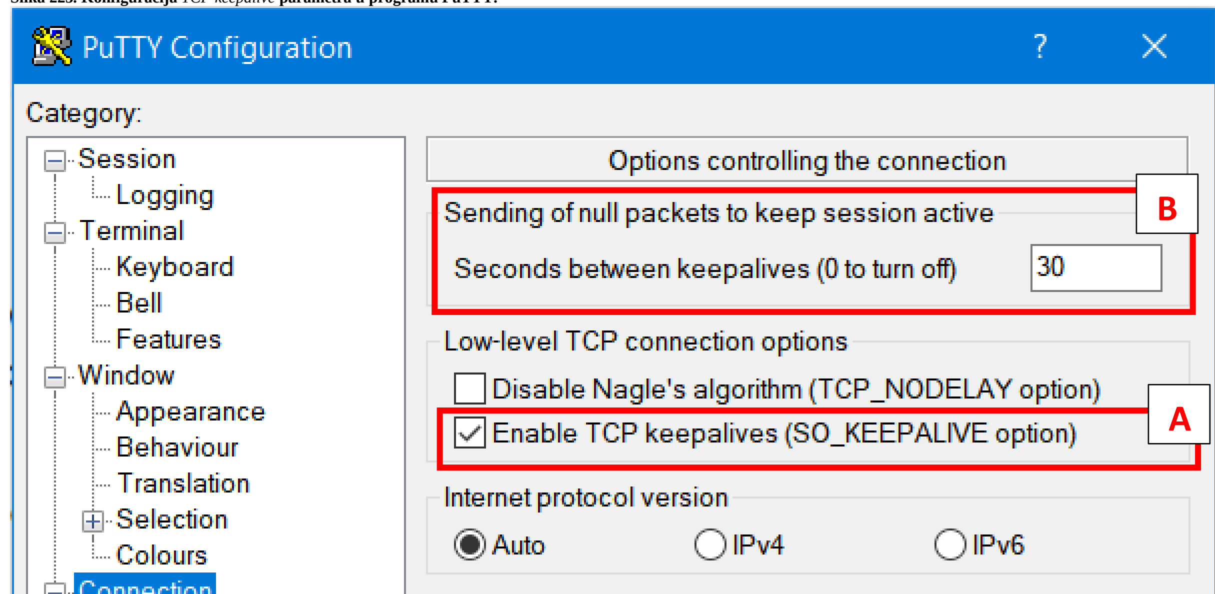


Vezano za potvrđivanje primitka TCP paketa, pogledajte poglavlje:
**24.2.5. Standardne potvrde (*ACK*).**

Većina računala koji podržavaju *TCP* protokol podržavaju i *TCP Keepalive* poruke. *Keepalive* radi tako da klijentska strana povremeno šalje *TCP* paket svom susjednom računalu unutar komunikacijskog kanala, na koji se zahtjeva odgovor odnosno potvrda primitka ove poruke.

Ako se šalje određeni broj *keepalive* poruka i ne primi potvrda za njih (s `ACK`), poslužiteljska strana će prekinuti vezu.
Nadalje, ako je veza prekinuta zbog *TCP Keepalive* isteka vremena *(timeouta)*, a druga strana šalje paket za staru (već zatvorenu) vezu, strana koja je prekinula vezu će poslati paket s postavljenom `RST` zastavicom da signalizira drugom *računalu* kako stara veza više nije aktivna.
To će prisiliti drugu stranu da prekine vezu kako bi se uspostavila nova TCP veza odnosno komunikacijski kanal.
Ovaj mehanizam je jednostavan: nakon što se *TCP* komunikacijski kanal otvori, provjeravaju se *keepalive* vremenski okviri (*timeri*). Kako prolazi vrijeme, *TCP timer* se smanjuje do nule, a tada se šalje *TCP keepalive* poruka na drugu stranu, te se ovaj *timer* ponovno povećava na inicijalnu vrijednost, pa ponovno smanjuje protekom vremena i tako iz početka.

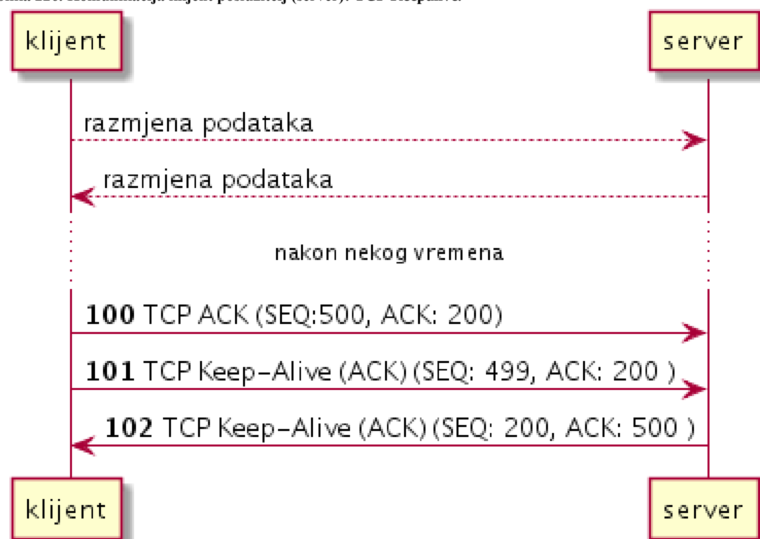
**Slika 225. Konfiguracija *TCP keepalive* parametra u programu PuTTY.**




Pogledajmo konfiguraciju upotrebe *TCP keepalive* mehanizma za *SSH* klijenta u programu **PuTTY** (slika 225).
U njemu *SSH* klijenta možemo konfigurirati tako da šalje *TCP Keepalive* poruke (dio označen sa A) i to u željenim vremenskiom intervalima od konkretno 30 sekundi (dio označen sa B) i to od trenutka uspostave *SSH* veze.

S ovim mehanizmom smo se osigurali da iako kroz ovaj (SSH) program potencijalno duže vremena ne bude prometa odnosno komunikacije, da se komunikacijski TCP kanal neće zatvoriti.
Pogledajmo i kako izgleda ovakva komunikacija, gledajući na mrežne pakete, kako je vidljivo na slici 226. na sljedećoj stranici.
Nakon što klijent pošalje zadnju `ACK` poruku (ovdje označenu kao **100**) o ispravnom primitku zadnjeg bloka podataka (paketa) i istekne 60 sekundi koje smo namjestili u programu, on mu šalje *TCP keepalive ACK* paket (**101**) s kojim govori kako je `SEQ` broj, jedan *bajt* manji od onoga koji se očekuje, s istom `ACK` porukom (`ACK` brojem).
Poslužitelj mu tada potvrđuje primitak *TCP keepalive* poruke, s porukom br.**102** (u našem primjeru).

Slika 226. Komunikacija klijent poslužitelj (server): TCP Keepalive.




Tipična postavka za Linux računala je da, ako je ***keepalive*** opcija uključena (u aplikaciji), šalje se prvi TCP *keepalive* paket nakon `75` sekundi neaktivnosti veze. Ako je konekcija neaktivna i druga strana nije odgovorila, ponovno se šalje nova poruka nakon `75` sekundi. Ako nema odgovora na `9` *keepalive* poruka, veza se zatvara.

Dakle veza se u tom slučaju zatvara ukupno nakon: **9** *pokušaja* **x 75** *sekundi* **= 675** sekundi (cca. 11 minuta). Zatvaranje veze se odrađuje standardnim TCP mehanizmima.
Ove vrijednosti definirane su u *sysctl* varijablama koje ćete ubrzo upoznati.



*Iako je **TCP keepalive** konfiguriran na sustavu, stvar je aplikacije da koristi (ili ne) ove postavke.*
*Naime, Linux sâm od sebe neće slati **keepalive** pakete, ako to konkretna aplikacija ne podržava.*
*Ako se aplikacija pokreće s opcijom **SO_KEEPALIVE** odnosno ako je s njom otvorila mrežni socket, samo i isključivo tada će se i koristiti **TCP keepalive** mehanizam na razini aplikacije, što je potom vidljivo i iz Linuxa.*

Pogledajmo Linux *TCP keepalive* vremenske okvire/brojače (*timere*) i njihove postavljene vrijednosti:

| Lokacija varijable | *sysctl* naziv varijable | Standardna vrijednost varijable |
|---|---|---|
| `/proc/sys/net/ipv4/tcp_keepalive_time` | `net.ipv4.tcp_keepalive_time` | **`7200 sekundi`** |
| `/proc/sys/net/ipv4/tcp_keepalive_intv` | `net.ipv4.tcp_keepalive_intvl` | **`75 sekundi`** |
| `/proc/sys/net/ipv4/tcp_keepalive_probes` | `net.ipv4.tcp_keepalive_probes` | **`9 komada`** |

Kako provjeriti koristi li određena mrežna aplikacija ***TCP keepalive*** mogućnost sustava. Pogledajmo nekoliko mrežnih servisa i njihove *timere* (`-o`) u kojima ćemo vidjeti i status upotrebe *TCP keepalive* mogućnosti (filtrirano je samo nekoliko servisa):

```
netstat -anto | grep ESTABLISHED

Proto Recv-Q Send-Q Local Address    Foreign Address    State       Timer
tcp        0    112  10.10.10.1:22    10.10.10.100:2551 ESTABLISHED on (0.10/0/0)
tcp        0     96  10.10.10.1:80    10.10.10.100:7551 ESTABLISHED keepalive (84.28/0/0)
tcp        0      1  10.10.10.1:8888 10.10.10.100:7551 ESTABLISHED off (0.00/0/0)
```

Ovdje vidimo tri mrežna servisa na koje imamo otvorene TCP veze:

- SSH Servis na portu `22`.
- HTTP (Web) servis na portu `80`.
- Posebni mrežni servis na portu `8888`,.

Za *SSH* servis (port `:22`) u stupcu **`Timer`** (koji ćemo sada promatrati) vidimo stanje `on` što znači kako ovaj servis u ovom trenutku čeka poruku o potvrdi paketa (`TCP ACK`), što nužno (o ovom djeliću sekunde) ne znači kako je *TCP keepalive* onemogućen, već da se trenutno čeka na TCP potvrdni paket, nakon kojega se može i preći u *keepalive* stanje.
Za ***HTTP*** servis (port `:80`) vidimo kako je on (servis) definitivno pokrenut tako da je otvorio *TCP socket*, tako da podržava *TCP keepalive* mogućnost jer se vidi ključna riječ: `keepalive` što znači i kako je *keepalive timer* aktivan.

Pošto je trenutno *keepalive timer* aktivan, uz njega su vidljivi brojevi koji znače sljedeće: prvi broj (`84.28`) je ***keepalive*** brojač, koji se stalno smanjuje (od inicijalnih ***75*** sekundi) te kada dođe na nulu (***0***), ponovno se provjerava je li bilo ikakvog prometa na toj konekciji, a ako nije, šalje se TCP *keepalive (samo ako je aplikacija otvorila socket s opcijom **SO_KEEPALIVE**).*

Drugi broj označava broj retransmisija (`0`), a treći broj *keepalive* poruka (*probes*) koje su poslane.
Zadnji servis (naša posebna mrežna aplikacija `:8888`) nije u mogućnosti koristiti *TCP keepalive*, pa ima stanje: `off`.



Pogledajte i poglavlja:
**24.2.2. Kako se uspostavlja TCP veza** → za uspostavu i održavanje veze.
**24.2.4. Trajanje TCP veze** → za vrijeme trajanja TCP veze.
**24.2.5. Standardne potvrde (*ACK*)**



**Postoji i mogućnost slanja takozvanih** *TCP probe* **poruka drugoj strani, kako bi se provjerilo je li druga strana dostupna, što se koristi kod** ***Load Balancera*** **poput primjerice** `HAproxy` **ili** `keepalived` **servisa.**



Vezano za takozvane **TCP probe** poruke, pogledajte poglavlje:
**24.2.12. TCP reset.**

## Konfiguracija SSH klijenta pod Linuxom, vezana za TCP *keepalive* opciju

Za trenutnog korisnika kreirajmo konfiguracijsku datoteku za SSH klijenta, s uređivačem teksta `vi`:

```
mkdir $HOME/.ssh
vi $HOME/.ssh/config
```

U navedenu datoteku ćemo dodati sljedeće retke:

```
Host *
    ServerAliveInterval 120
```

U prvom retku s `Host *` smo definirali da sve što slijedi, vrijedi za sve udaljene SSH poslužitelje.
Potom smo u drugom retku definirali opciju `ServerAliveInterval` koju smo postavili na 120 sekundi, što znači da će naš SSH klijent prema SSH poslužitelju svakih 120 sekundi slati poruke o održavanju veze živom (*keepalive*).
Zatim konfiguracijskoj datoteci SSH klijenta, promijenimo ovlasti, kako bi joj se moglo pristupiti i da bi sve radilo kako treba:

```
chmod 600 $HOME/.ssh/config
```

**Isto je moguće napraviti i na strani SSH poslužitelja, uređivanjem datoteke:** `/etc/ssh/sshd_config` **i dodavanjem:**

```
ClientAliveInterval 120
```

*S ovom opcijom smo SSH poslužitelju naložili da on svaku ostvarenu SSH vezu održava živom, što nije dobra opcija.*

## Analiza mrežnog prometa po pitanju TCP keepalive poruka

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa.
Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
IpInReceives                    2112345
```

← U ispisu vidimo statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom.

Pogledajmo tablicu s brojačima odnosno statistikama vezanim za TCP sloj po pitanju *TCP keepalive* poruka:

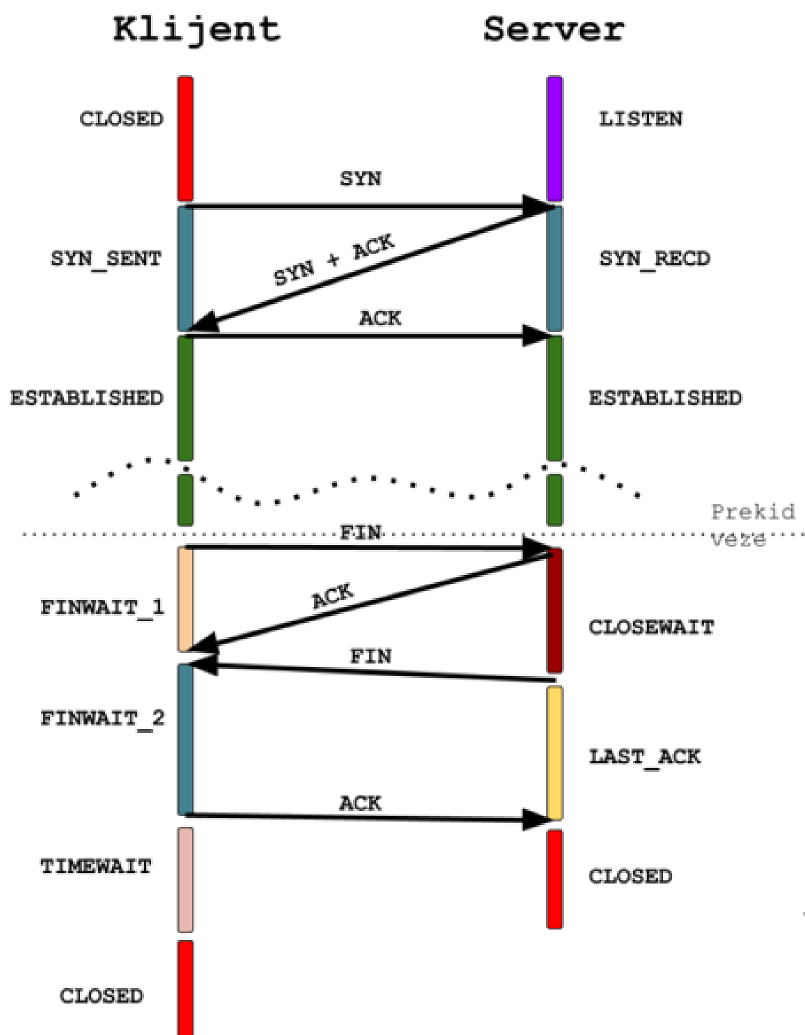
| Naziv brojača | Opis |
|---|---|
| **`TcpExtTCPKeepAlive`** | Ovaj brojač bilježi ukupan broj *TCP keepalive* poruka koje su poslane. Povezana ***sysctl*** varijabla: `net.ipv4.tcp_ext.tcp_keepalive` |
| **`TcpExtTCPKeepAliveGhosts`** | Ovaj brojač bilježi ukupan broj TCP veza koje su zatvorene zbog isteka *TCP keepalive* vremenskog ograničenja, a nisu primile nikakav odgovor. Povezana ***sysctl*** varijabla: `net.ipv4.tcp_ext.tcp_keepalive_probes`. |
| **`TcpExtTCPKeepAliveProbes`** | Ovaj brojač bilježi ukupan broj *TCP keepalive* provjera koje su poslane. Povezana ***sysctl*** varijabla: `net.ipv4.tcp_ext.tcp_keepalive_time`. |
| **`TcpExtTCPKeepAliveProbesFailed`** | Ovaj brojač bilježi ukupan broj neuspjelih *TCP keepalive* provjera. Povezana ***sysctl*** varijabla: `net.ipv4.tcp_ext.tcp_keepalive_intvl`. |

**Izvor informacija: (K-6),(639),(640),(641),(642),(916),(917),(1406),(1650),(1651),** **`man socket`**, **`man 7 tcp`**, **`man 5 ssh_config`**, **`man nstat`**.

## 24.2.16. Stanja TCP veze i njena vremenska ograničenja *(timeri)*

Svaka *TCP* veza u određenom trenutku može biti u više stanja, gledano s razine operativnog sustava, odnosno u više među stanja. Pogledajmo koja su to sve među stanja *TCP* veza kako je vidljivo na slici 227.

**Slika 227. Stanja TCP veze klijenta i poslužitelja.**



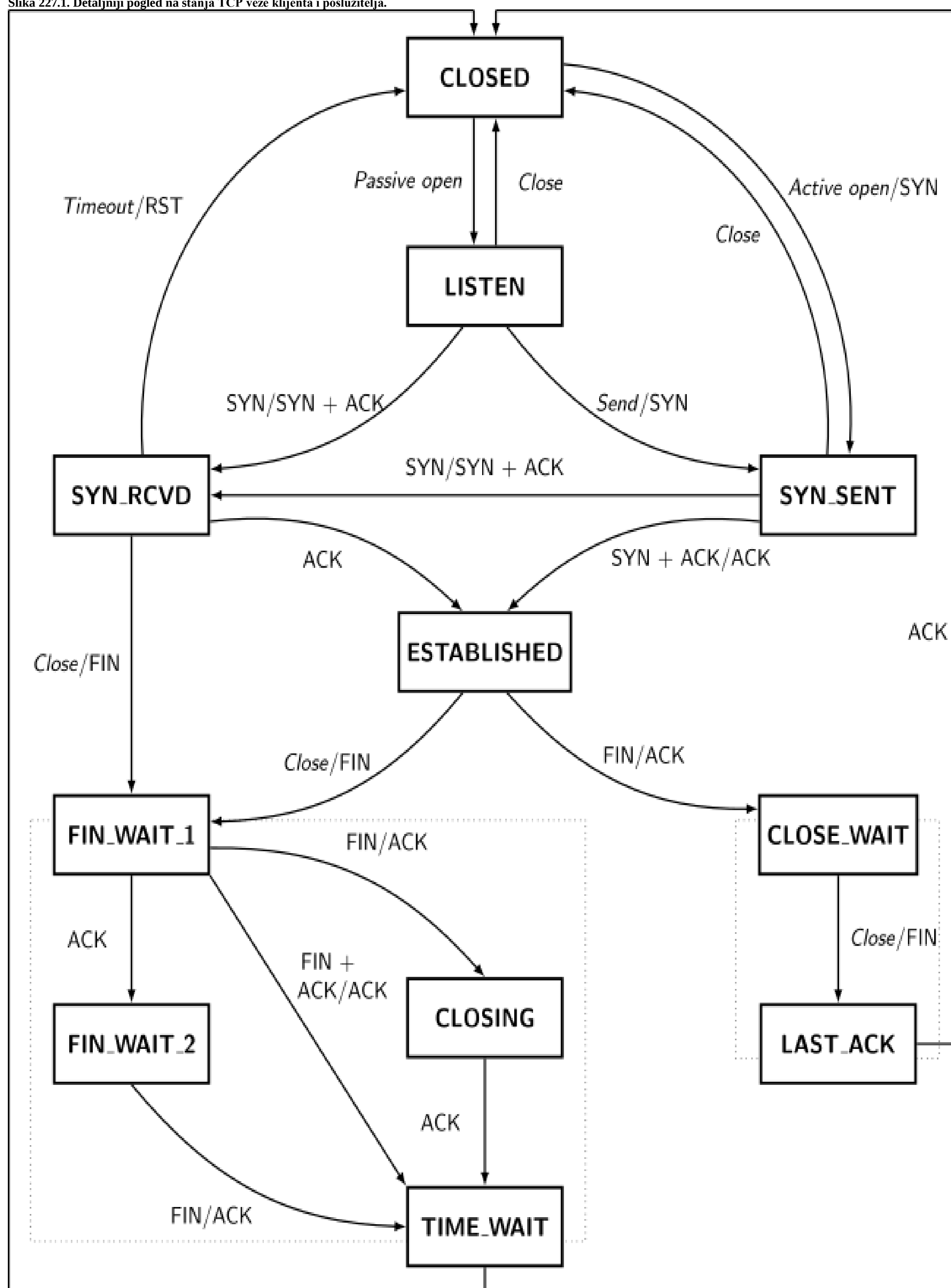

Prije nego počnemo s objašnjenjima, ne zaboravite kako su *TCP* stanja značajna za vezu na uređaju odnosno računalu, zasebna za klijenta i poslužitelj (server). Naime ona nisu stanja sâme veze, tako da jedan uređaj odnosno strana može biti u jednom stanju, a drugi u drugom stanju.
Kao što možete vidjeti u primjeru, klijent počinje sa statusom TCP veze u `CLOSED` (zatvoreno) stanju, ali poslužitelj je već u stanju `LISTEN` jer je poslužiteljska aplikacija (pr. *Web* poslužitelj) na poslužitelju, ona koja mora slušati zahtjeve i kreirati odgovore, već pokrenuta i aktivna.
Klijent šalje `SYN` poruku i pomiče se u stanje `SYN_SENT`. Kada poslužitelj primi taj `SYN` paket, on će se prebaciti na stanje `SYN_RECEIVED` slanjem `SYN+ACK` poruke, natrag klijentu. Klijent će se potom prebaciti u stanje `ESTABLISHED` te poslati `ACK` poruku kao odgovor.
Čim primi `ACK` poruku, poslužitelj se prebacuje u stanje `ESTABLISHED` (ostvareno).
Nadalje je važno znati da poslužitelj prelazi iz stanja `LISTEN` u `SYN_RECEIVED` stanje, za svaki TCP komunikacijski kanal koji ima otvoren, jer poslužitelj može istodobno prihvatiti višestruke veze s različitih klijenata.
To znači kako svaki puta kada poslužitelj zaprimi `SYN` paket, stvorit će zasebnu instancu komunikacijskog kanala za tu vezu, koja će biti zatvorena kada se ta veza završi. Na taj će način na poslužitelju uvijek postojati instanca u stanju `LISTEN` **(**stanje slušanja) spremna prihvatiti nove veze, od novih klijenata. Kada se razmjena podataka dovrši, klijent u ovom primjeru želi prekinuti vezu. To se postiže slanjem `FIN` poruke i prijelazom na stanje `FIN_WAIT_1`.
Kada poslužitelj primi tu `FIN` poruku, on će odgovoriti s `ACK` i prijeći u stanje `CLOSE_WAIT`.
Kada klijent primi ovaj `ACK` on prelazi u stanje `FIN_WAIT_2` čekajući da se poslužiteljska strana veze zatvori.
Nakon što poslužitelj završi s finalnim radnjama, on šalje poruku `FIN`, prelazeći u stanje `LAST_ACK`. Kada je `FIN` primljen na strani klijenta, klijent šalje `ACK` poruku i pomiče se u stanje `TIME_WAIT`, a tek nakon isteka `TIME_WAIT` *timera* prelazi u stanje `CLOSED` odnosno zatvoreno. S poslužiteljske strane, kada se primi `ACK` od klijenta, poslužitelj prelazi u stanje `CLOSED`. Potom je ova veza (*Klijent 1:portX - Server 1:portY*) završila. U zadnjem koraku, nakon slanja zadnje `ACK` poruke (paketa) koju šalje strana koja je inicirala zatvaranja veze, inicijator zatvaranja veze obavezno prije zatvaranja veze prelazi u `TIME_WAIT` i prije nego zatvori vezu, čeka istek ovog *timera (brojača)*. Naime moguće je da i serverska strana inicira zatvaranje veze pa će onda i ona nakon što pošalje zadnji `ACK` paket, preći u `TIME_WAIT` i čekati istek ovog *timera (brojača)*.



Pogledajte i sljedeća poglavlja:
**24.2.2. Kako se uspostavlja TCP veza.**
**24.2.4. Trajanje TCP veze.**
**24.2.11. Kako se zatvara TCP veza.**

Pogledajmo i stanja TCP veze na malo detaljnijem dijagramu, na slici 227.1.

**Slika 227.1. Detaljniji pogled na stanja TCP veze klijenta i poslužitelja.**




Autor dijagrama: ***Ivan Griffin***, licenca: *LaTeX Project Public License 1.3*

**Izvori informacija: (K-6),(708),(709),(1406),** `man 7 tcp`**, RFC793, RFC 1122.**

### 24.2.16.1. Timeri i stanja veze

*TCP* protokol se u radu oslanja na nekoliko brojača vremenskih okvira odnosno takozvanih *tajmera (Engl. Timer)* unutar kojih se određene akcije moraju odraditi ili na koje se mora čekati.
Proći ćemo neke od njih, a koje su lako vidljive u operativnom sustavu Linux:

- ***TCP (Re)transmit Timer** i **Zero Window Probe Timer.***
- ***Delay ACK Timer** i **Keepalive Timer.***
- ***SYN-ACK Timer** i **TIME_WAIT Timer.***

Neke od osnovnih *tajmera* možemo vidjeti s poznatim *Linux* programom `netstat` korištenjem prekidača `-o`.
Međutim gotovo iste statistike možemo dobiti i s naredbom: `ss -tn -o`. Ipak se vratimo naredbi `netstat`:
Zapravo ćemo koristiti još neke od prekidača naredbe `netstat` koje ćemo sada navesti:

- `-t` - želimo vidjeti sve TCP konekcije.
- `-a` - prikaz svih stanja konekcija.
- `-n` - numerički prikaz, bez pretvaranja IP adresa u imena računala, broj porta ili imena korisnika.
- `-o` - prikaz TCP *timera.*
- `--timer` - samo za prikaz svih *tajmera.*

Pogledajmo i kako to napraviti s naredbom `netstat`:

```
netstat -anto
```

Dobit ćemo ovakav ispis (skratili smo ga):

```
Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address       Foreign Address       State       Timer
tcp        0      0 127.0.0.1:25        0.0.0.0:*             LISTEN      off (0.00/0/0)
tcp        0      0 0.0.0.0:22          0.0.0.0:*             LISTEN      off (0.00/0/0)
tcp        0    320 192.168.1.254:22    192.168.1.128:2303    ESTABLISHED on (0.21/0/0)
tcp        0      0 192.168.1.254:445   192.168.1.128:1598    ESTABLISHED keepalive (6934.86/0/0)
tcp        0      0 192.168.1.254:445   192.168.1.129:60194   ESTABLISHED keepalive (20.38/0/0)
```

Dodatno, mogli smo koristiti i prekidač `-c` kako bi se podaci osvježavali u stvarnom vremenu.
Pogledajmo samo zadnja dva (označena) stupca:

- `State` - označava u kojem je stanju pojedina veza: **`LISTEN`**, **`TIME_WAIT`**, **`ESTABLISHED`**, … .
- `Timer` - podijeljen je u dva dijela:
  - Prvi dio može biti u stanjima:
    - `keepalive` - ako je *keepalive timer* (i funkcionalnost) uključen za *socket* (konekciju).
    - `on` - ako je trenutno uključen *retransmisijski* timer za *socket* (konekciju).
    - `off` - ako nije niti prvi niti drugi *tajmer* uključen, u njemu su vidljiva sljedeća stanja:
      - `LISTEN` - stanje čekanja na pristup klijenta.
      - `ESTABLISHED` - stanje uspostavljene veze (dvosmjernog komunikacijskog kanala).
      - `CLOSE_WAIT` je stanje u kojemu nakon prve primljene poruke za zatvaranje veze (`FIN`) te našeg odgovora (`ACK`) odmah (bez *timera*) šaljemo poruku `FIN` drugoj strani, kako bi se veza zatvorila. Stanje `CLOSE_WAIT` u normalnom radu prolazi u djeliću sekunde, tako da ga je gotovo nemoguće primijetiti. Naime ovo je faza u kojoj smo primili (mi kao poslužitelj) zahtjev za zatvaranje veze od strane klijenta, ali naša aplikacija još nije zatvorila ovu vezu odnosno ovaj komunikacijski kanal, jer je zauzeta ili u nekim problemima. Dakle naša aplikacija može raditi ili u najgorem slučaju ostati u zaglavljenom stanju, a kernel u stanju čekanja (za ovu vezu) kako bi mu aplikacija naredila da zatvori ovu vezu odnosno mrežni *socket*.
    - `timewait` - predstavlja stanja čekanja, poput:
      - `FIN_WAIT_1` - je stanje u kojemu se čeka na potvrdu standardne terminacije veze s druge strane s porukom `ACK`, a što je odgovor na prvu `FIN` poruku.
      - `FIN_WAIT_2` - je stanje u kojem nakon što je dobivena poruku o terminaciji veze (`FIN`) od druge strane, odgovara se s porukom `ACK` te se čeka istek *timera*, prije prelaska u sljedeće stanje. U Linuxu je ovo vrijeme obično postavljeno na 60 sekundi, i definirano u *sysctl* TCP varijabli: `net.ipv4.tcp_fin_timeout`.
      - `TIME_WAIT` - kod kojega se čeka dovoljno vremena, kako bi se uvjerilo da je udaljena strana *TCP* konekcije primila potvrdu o zahtjevu za prekid veze. Drugi slučaj može biti kada veza ulazi u stanje `TIME_WAIT` kada se previše klijentskih veza uspostavlja u kratkom vremenskom razdoblju i svaki klijent brzo prekida vezu. Linux postavlja ovu vrijednost na **60** sekundi, standardno. Prema RFC793 ovo vrijeme iznosi 2 x **MSL** (Maximum Segment Lifetime) vrijeme.

- Drugi dio unutar zagrade **( )** se sastoji od tri vrijednosti, odvojene sa znakom **/** prema principu: ***(x/y/z)***
  - Prva vrijednost (**x**) - označava vrijeme *timera*: u sekundama od kada su zaprimljeni zadnji podaci (paketi) do trenutka kada će biti poslan prvi *keepalive probe*, ako se radi o `keepalive` odnosno *retransmission timeru*.
  - Pogledajmo ove opcije:
    - Ako je prvo polje `keepalive` onda je ovo *keepalive timer*. Standardno je ovo vrijeme 7200 sekundi i resetira se svaki puta kada se zaprime novi podaci (paketi).
      Ako je ova vrijednost relativno niska to znači kako se neko vrijeme *keepalive* konekcije nisu osvježavale ili su u najgorem slučaju ostale u zablokiranom stanju.
    - Ako je prvo polje **on**, onda je ovo *retransmission timer*.
  - Druga vrijednost (**y**) - označava broj retransmisija.
  - Treća vrijednost (**z**) - označava broj *keepalive probe* paketa koji su poslani.

Pogledajmo dostupne osnovne brojače (*tajmere*) u Linuxu:

| Stanje | *sysctl* ***naziv varijable*** | Komentar | Standardna vrijednost varijable |
|---|---|---|---|
| **FIN_WAIT_2** | `net.ipv4.tcp_fin_timeout` | | 60 sekundi |
| **TIME_WAIT** | **TIME_WAIT**= | **2 x FIN_WAIT2** | 120 sekundi |
| | `net.ipv4.netfilter.ip_conntrack_tcp_timeout_time_wait` | Radi samo uz upotrebu kernel slijedećih modula: `ip_conntrack` i `xt_conntrack` ili `nf_conntrack` | 120 sekundi |
| | `net.netfilter.nf_conntrack_tcp_timeout_established` | | 432000 sekundi |
| | `net.ipv4.tcp_tw_recycle` | Pogl. opis dolje | 0 |
| | `net.ipv4.tcp_tw_reuse` | Pogl. opis dolje. | 2 |
| | `net.ipv4.tcp_max_tw_buckets` | Maksimalan broj *socketa* u **TIME_WAIT** stanju | 8192 |

Prva navedena TCP varijabla `tcp_fin_timeout` određuje koliko će se sekundi pričekati prije nego se zatvori *socket*, koji je u stanju zatvaranja ***FIN-WAIT-2***. Za poslužitelje s velikim brojem konekcija koje se stalno otvaraju i zatvaraju ovo vrijeme se prema potrebi može dosta smanjiti, kako bi se smanjilo čekanje na zatvaranje *socketa* odnosno brže oslobađanje istih za nove veze. Dodatno svaki otvoreni *socket* troši i RAM memoriju: oko 1.5kB po konekciji. TCP varijabla `tcp_tw_recycle` omogućava brzo recikliranje ***TIME-WAIT*** *socketa*. Postavljena vrijednost je **0** (isključeno). Omogućavanje ove opcije nije preporučljivo jer to uzrokuje probleme prilikom rada s **NAT** (*Network Address Translation*) tehnologijom.

Druga TCP varijabla `tcp_tw_reuse` dopušta da se ponovno koristi ***TIME-WAIT*** *socket* za nove veze kada je to moguće, s točke gledišta protokola. Postavljena vrijednost je **0** (isključeno). Kod zadnje dvije vrijednosti: `tcp_tw_recycle` i `tcp_tw_reuse` problem je potencijalno u tome što omogućavanje ovih opcija neće uzrokovati rušenje računala ili nestabilnosti, ali može praviti probleme u TCP/IP funkcionalnosti, ako je računalo (*host*) povezao s uređajima kao što su *load balanceri* ili vatrozidi (*firewalls*).

Neki od tih uređaja mogu odbiti **SYN** paket za uspostavljanje nove TCP konekcije, ako se ponovno koristi ista veza (koja je brzo prekinuta i pre brzo ponovno iskorištena za novu upotrebu). Dakle kada su *Src/dst* IP i *src/dst* portovi isti, ali se sve dogodi prebrzo. Standard RFC 1122 opisuje kada je prihvatljivo reciklirati vezu kada **SYN** stigne za vezu koja je trenutno u stanju ***TIME_WAIT***. Problem (ako postoji) će se manifestirati kao greške pri uspostavljanju veze ili prestanak rada nekih veza koje ispravno rade, na primjer kada uspostavljanje TCP veze (*TCP three way handshake*) ne uspije ili kada se veza ne zatvori ispravno. Postavljanje `tcp_tw_recycle` na **1** uzrokuje da Linux računalo smanjuje ***TIME_WAIT*** stanje veze, mnogo brže nego inače. Umjesto unaprijed definiranog vremena od: **2 x** ***MSL*** vrijednosti, u vremenu od 60 sekundi, koristit će se *timeout* na temelju ***RTT*** procjene. Za **LAN** mreže (lokalne mreže) obično je to nekoliko milisekundi. Postavljanje `tcp_tw_reuse` na **1** učinit će ponovnu upotrebu iste veze za nove odlazne veze i to vrlo brzo (potencijalno prebrzo za uređaje koje smo već naveli).
Druge vrijednosti *tajmera* koje se mogu optimizirati, samo ako su učitani dodatni kernel moduli: `ip_conntrack` i `xt_conntrack` ili `nf_conntrack` možemo izlistati sa sljedećom naredbom:

```
sysctl net.netfilter
```

## Analiza mrežnog prometa po pitanju TCP veza u stanju čekanja/zatvaranja

Pogledajmo kako pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa.
Primjerice, ako nas zanima brojač imena `TcpInSegs`:

```
nstat -az TcpInSegs
```

Pogledajmo tablicu s brojačima odnosno statistikama direktno i indirektno vezanim za *TCP TIME/FIN WAIT* stanja:

| Naziv brojača | Opis |
|---|---|
| **`TcpExtTW`** | Ovaj brojač prati ukupan broj trenutnih veza u **TIME WAIT** stanju. Kada se TCP veza zatvori, ulazi u **TIME WAIT** stanje kako bi osigurala da svi paketi povezani s tom vezom budu obrađeni prije nego što se veza potpuno ukloni. Broj veza u **TIME WAIT** stanju može biti važan za praćenje opterećenja na mreži i poslužitelju. |

| Naziv brojača | Opis |
| --- | --- |
| `TcpExtTWRecycled` | Ovaj brojač bilježi broj recikliranih TCP veza u **TIME WAIT** stanju. Recikliranje veza je proces ponovnog korištenja resursa veza koje su zatvorene kako bi se izbjeglo stvaranje novih veza i povećala efikasnost resursa. Broj recikliranih veza može biti pokazatelj efikasnosti sustava u upravljanju veza u **TIME WAIT** stanju. |
| `TcpExtTWKilled` | Ovaj brojač bilježi broj ubijenih (*killed*) TCP veza u **TIME WAIT** stanju. TCP veza može biti ubijena ako je otkrivena kao zastarjela ili nepoželjna. To se obično događa ako je veza previše dugo u **TIME WAIT** stanju ili ako postoje problemi s resursima. Broj ubijenih veza može biti koristan pokazatelj problema u sustavu ili potrebe za prilagodbama u konfiguraciji **TIME WAIT** vremenskog ograničenja. |
| `TcpExtTWTimeouts` | Ovaj brojač broji koliko puta se dogodio istek vremenskog ograničenja (*timeout*) za TCP vezu u **TIME WAIT** stanju. **TIME WAIT** stanje je ono kada se jedan od sudionika veze (najčešće klijent) zatvori, ali još uvijek prima potencijalne pakete koji su mogli biti poslani prije zatvaranja. Za kontrolu vremenskog ograničenja **TIME WAIT** stanja možete koristiti ***sysctl*** varijablu `net.ipv4.tcp_fin_timeout`. |
| `TcpExtTCPLossFailures` | Ovaj brojač prati broj neuspjelih TCP veza. To uključuje situacije kada je veza izgubljena tijekom prijenosa ili kada su se dogodile greške tijekom veze, što može dovesti do **TIME WAIT** stanja. Za kontrolu ponašanja TCP veza i reakcije na gubitke možete koristiti ***sysctl*** varijable; `net.ipv4.tcp_retries1` i `net.ipv4.tcp_retries2`. |
| `TcpExtTCPFastOpenActive` | Ovaj brojač bilježi broj aktivnih *TCP Fast Open* veza. **TCP Fast Open** je mehanizam koji omogućava klijentu slanje podataka u prvim paketima za inicijalno uspostavljanje veze (*handshake*) kako bi se ubrzalo uspostavljanje veze. Broj aktivnih veza može utjecati na količinu veza u **TIME WAIT** stanju. |
| `TcpExtTCPFastOpenPassive` | Ovaj brojač bilježi broj pasivnih *TCP Fast Open* veza. To su veze koje su primile podatke u prvim paketima za inicijalno uspostavljanje veze (*handshake*) kako bi ubrzale uspostavljanje veze. I ovdje, broj pasivnih veza može utjecati na **TIME WAIT** stanje. |
| `TcpExtTCPBacklogDrop` | Ovaj brojač bilježi broj odbačenih veza zbog prekoračenja **TCP backlog** međumemorije. TCP *backlog* je ograničenje na broj otvorenih veza koje poslužitelj može prihvatiti. Ako se veze odbacuju, to može dovesti do **TIME WAIT** stanja za veze koje su već uspostavljene. Povezana ***sysctl*** varijabla za upravljanje veličinom *backlog* međumemorije je: `net.core.somaxconn`. |
| `TcpExtTCPRcvCollapsed` | Ovaj brojač broji koliko puta je dolazni TCP segment (paket) "*kolabirao*". Ako dolazni TCP segment ne može biti u potpunosti smješten u međumemoriju za prijem jer nema dovoljno slobodnog prostora, onda se dio tog segmenta može kombinirati s već postojećim segmentom u memoriji. Takvi scenariji mogu izazvati **TIME WAIT** stanje. Uzrok je preopterećen sustav ili poslužiteljska aplikacija. |
| `TcpExtTCPPartialUndo` | Ovaj brojač bilježi broj parcijalnih poništenja TCP veza. To se događa kada TCP segment koji je već potvrđen bude djelomično poništen kasnijim segmentom. Ovi događaji mogu rezultirati dodatnim vremenom zadržavanja veze u **TIME WAIT** stanju. |

**Izvori informacija: (340),(1406),(1650),(1651),(K-6),`man 7 tcp`,`man nstat`,RFC793, RFC 1122.**

## *24.2.16.2.* Maximum Segment Lifetime *(MSL)*

**MSL** (*Maximum Segment Lifetime*) je maksimalno vrijeme u sekundama, unutar kojega TCP paket može biti "živ" na mreži. Ovo je važno vrijeme i mehanizam kod zatvaranja *TCP* konekcije između stanja `TIME_WAIT` i `CLOSED`.
Standardno je prema RFC **RFC 793**, `TIME_WAIT` vrijeme jednako **2 x *MSL*** vremenu, kako bi se osiguralo da ne dođe do dolaska zakašnjelih paketa u trenutku kada je veza već zatvorena. To znači kako ova vremena ne smiju biti previše mala, ali niti previše velika, kako pogotovo na sustavima na kojima postoji veliki broj veza koje se otvaraju i zatvaraju, ne bi ostao preveliki broj veza koje bi trebalo već zatvoriti, a one su trenutno u stanju čekanja (`TIME_WAIT`) s čime se nepotrebno troše resursi.
**Za čuvanje (održavanje) svake TCP veze, potrebno je dodatno procesorsko vrijeme, RAM memorija i *file deskriptori*.**



Za više detalja o optimizaciji mreže, kako na TCP/UDP/IP ali i drugim slojevima mreže pogledajte poglavlje: **25.1.1 Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**

**Izvori informacija:(335),(336),(337),(338),(339),(340),(K-6), `man 7 tcp`, RFC 793.**

### 24.2.17. Uzroci odbacivanja TCP veza

Uzroci odbacivanja TCP veza mogu biti višestruki. U ovoj cjelini ćemo pomoću naredbe `nstat` (koja dolazi u `iproute2` programskom paketu) pratiti statistke mrežnih paketa na TCP sloju koje su vezane za odbacivanje paketa.
Primjerice, ako nas zanima brojač odnosno statistika mrežnog prometa imena `TcpInSegs`, trebamo pokrenuti:

```
nstat -az TcpInSegs
```

← U ispisu ćemo vidjeti statistike za navedeni brojač `TcpInSegs` odnosno koliko mrežnih paketa je zavedeno pod tom statistikom. Pogledajmo tablicu s brojačima odnosno metrikama vezanim za TCP sloj po pitanju odbacivanja TCP veza:

| Naziv brojača | Opis |
|---|---|
| **TcpAttemptFails** | Ovaj brojač bilježi broj neuspjelih pokušaja uspostavljanja TCP veza. To su situacije kada klijent pokušava uspostaviti vezu (putem TCP *3-way handshake* procesa), ali veza ne uspije. Uzroci su nedostupnost servisa ili preopterećenost sustava. |
| **TcpExtTCPMinTTLDrop** | Ovaj brojač bilježi ukupan broj TCP segmenata koji su odbačeni jer im je **IP TTL** (*Time-to-Live*) manji od minimalnog TTL-a koji je postavljen na sustavu. To nije problem samo po sebi. Pogledajte: **23.4.1. Time to live (*TTL*).** |
| **TcpExtTCPOFOQueue** | Ovaj brojač bilježi broj TCP segmenta koji su stavljani u red čekanja (*queue*) zbog dolaska izvanrednih (*out-of-order*) segmenata. Kada TCP primatelj primi segmente koji nisu u očekivanom redoslijedu, on ih često stavlja u red čekanja dok se ne primi očekivani segment koji slijedi, kako bi se moglo rekonstruirati ispravno slijed podataka. Ovo također nije nužno problem osim ako se događa često. Pogl.: **24.2.14. Primitak paketa izvan redoslijeda slanja (*Out-Of-Order*).** |
| **TcpExtTCPOFODrop** | Ovaj brojač bilježi broj TCP segmenta koji su odbačeni zbog dolaska izvanrednih (*out-of-order*) segmenata. Kada red čekanja postane prevelik ili ako izvanredni segmenti postanu zastarjeli, TCP primatelj može odlučiti odbaciti neke od tih segmenata kako bi spriječio prekomjerno opterećenje i osigurao učinkovitost komunikacije. Ovo također nije nužno problem osim ako se događa često. Pogl.: **24.2.14. Primitak paketa izvan redoslijeda slanja (*Out-Of-Order*).** |
| **TcpExtTCPBacklogDrop** | Ovaj brojač prati broj TCP veza koje su odbačene zbog prekoračenja ograničenja (*backlog međumemorije*) reda čekanja na uspostavu veze. Kada je broj zahtjeva za uspostavu veze veći od dostupnih resursa za obradu, sustav može odbaciti nove zahtjeve, a ovaj brojač bilježi te slučajeve. Povećanje ***sysctl*** varijabli: `net.core.somaxconn`, `net.core.netdev_max_backlog`, `net.ipv4.tcp_max_syn_backlog` i `net.ipv4.tcp_syncookies` može pomoći. Pogl.: **10.5.1.3.1. Softverski signali prekida detaljnije.** |
| **TcpExtTCPReqQFullDrop** | Ovaj brojač bilježi ukupan broj TCP segmenata koji su odbačeni jer je red čekanja (*request queue*) pun. To se može dogoditi kada postoji previše aktivnih TCP veza ili kada je postavljeno ograničenje veličine reda čekanja (*queue*). Povećanje ***sysctl*** varijabli: `net.core.somaxconn` i `net.ipv4.tcp_max_syn_backlog` mogu pomoći, kao i indirektno: `net.ipv4.tcp_syncookies` i `net.ipv4.tcp_abort_on_overflow`. |
| **TcpExtTCPRcvQDrop** | Ovaj brojač bilježi ukupan broj TCP segmenata koji su odbačeni jer je red čekanja (*receive queue* međumemorija) pun. Povećanje ***sysctl*** varijabli: `net.core.rmem_max`, `net.core.rmem_default`, `net.core.netdev_max_backlog` može pomoći. Pogl.:**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].** Pogl.: **25.1.1. Raspodjeljivanje mrežnih paketa (network queuing/scheduler).** |
| **TcpExtTCPRcvCollapsed** | Ovaj brojač broji koliko puta je dolazni TCP segment (paket) "*kolabirao*". Naime, ako dolazni TCP segment ne može biti u potpunosti smješten u međumemoriju za prijem jer nema dovoljno slobodnog prostora, onda se dio tog segmenta može (barem djelomično) kombinirati s već postojećim segmentom u memoriji. Uzrok je preopterećen sustav ili poslužiteljska aplikacija. |
| **TcpExtPFMemallocDrop** | Ovaj brojač prati broj TCP paketa koji su odbačeni zbog problema s alociranjem memorije na sustavu. To može značiti da sustav nije mogao dodijeliti dovoljno memorije za obradu dolaznih TCP paketa, što može biti znak preopterećenja ili problema s resursima. Povećanje ***sysctl*** varijabli: `net.core.rmem_max`, `net.core.wmem_max`, `net.ipv4.tcp_mem`, `net.ipv4.tcp_rmem` i `net.ipv4.tcp_wmem` ovdje može pomoći. Pogledajte: **25.1.1. Raspodjeljivanje mrežnih paketa (network queuing/scheduler).** |

| Naziv brojača | Opis |
|---|---|
| **TcpExtListenOverflows** **TcpExtListenDrop** | Kada kernel primi **SYN** od klijenta i ako je red čekanja za TCP pun, kernel će ispustiti SYN paket i dodati **1** u `TcpExtListenOverflows`. U isto vrijeme kernel će također dodati **1** u `TcpExtListenDrops`. Kada je TCP socket u stanju **LISTEN**, a kernel treba ispustiti paket, kernel bi uvijek dodao 1 u `TcpExtListenDrops`. Dakle, povećanje `TcpExtListenOverflows` omogućilo bi povećanje `TcpExtListenDrops` u isto vrijeme, ali bi se `TcpExtListenDrops` također povećalo bez povećanja `TcpExtListenOverflows`, npr. neuspješna dodjela memorije također bi omogućila povećanje `TcpExtListenDrops`. Dakle na `TcpExtListenDrops` utječe nedostatak resursa (međumemorija), koji se može poboljšati s povećanjem vrijednosti *sysctl* varijabli: `net.core.somaxconn`, `net.core.netdev_max_backlog`, `net.ipv4.tcp_max_syn_backlog`. |
| **TcpExtPruneCalled** | Povećava se pri pokušaju smanjenja memorije dodijeljene *socketu* [**tcp_prune_queue()1**] |
| **TcpExtRcvPruned** | Povećava se kada probno smanjenje dodijeljene memorije *socketu* nije uspjelo, podaci se ispuštaju [**tcp_prune_queue()2**] |
| **TcpExtTCPAbortOnData** | Ova statistika prati broj TCP veza koje su prekinute zbog dolaska neočekivanih podataka na vezi. To može značiti da su primljeni podaci koji ne odgovaraju očekivanom obrascu ili protokolu, što može biti znak problema s komunikacijom ili neispravno konfiguriranim aplikacijama. |
| **TcpExtTCPAbortOnMemory** | Ova statistika prati broj TCP veza koje su prekinute zbog problema s memorijom. To se može dogoditi kada sustav nema dovoljno memorije za održavanje aktivne TCP veze ili kada je memorija postala neispravna. Povećanje vrijednosti ***sysctl*** varijabli: `net.core.rmem_max`, `net.core.rmem_default` može pomoći. |
| **TcpExtTCPAbortOnTimeout** | Ova statistika prati broj TCP veza koje su prekinute (abortirane) zbog isteka vremenskog ograničenja. Ovo se događa kada sustav odluči prekinuti vezu nakon što istekne određeno vremensko razdoblje bez odgovora od druge strane. Povećanje vrijednosti ***sysctl*** varijabli: `net.ipv4.tcp_syn_retries`, `net.ipv4.tcp_synack_retries` može pomoći.. |
| **TcpExtSyncookiesFailed** | Broj primljenih TCP paketa gdje ACK redni broj nije važeći *syncookie*. Aktiviranje ***sysctl*** varijable: `net.ipv4.tcp_syncookies` može pomoći |
| **TcpExtDelayedACKLocked** | Ovaj brojač prati broj situacija u kojima je odgoda ACK paketa blokirana (*locked*). To se može dogoditi kada postoji problem s resursima ili kada postoje ograničenja u mrežnoj konfiguraciji koji sprječavaju odgodu ACK paketa. Povećanje vrijednosti ***sysctl*** varijabli: `net.ipv4.tcp_delack_min`, `net.ipv4.tcp_delack_max` može pomoći. |
| **TcpExtTCPSackRecoveryFail** | Ovaj brojač bilježi broj neuspjelih pokušaja TCP-a da se oporavi (recovery) korištenjem SACK-a. To se može dogoditi kada SACK blokovi nisu dostupni ili nisu pouzdani, ili kada postoje problemi s primanjem potvrda od strane druge strane komunikacije. |
| **TcpExtTCPSackFailures** | Ovaj brojač bilježi broj neuspjelih pokušaja korištenja TCP SACK opcije tijekom komunikacije. Neuspjeli pokušaji mogu se dogoditi zbog različitih problema, uključujući nedostupnost SACK blokova ili nedostatak podrške za SACK opciju na drugoj strani komunikacije |
| **TcpExtTCPSACKDiscard** | Ovaj brojač bilježi broj TCP SACK potvrda koje su odbačene ili ignorirane. To se može dogoditi primjerice zbog nedostatak resursa, neispravne SACK potvrde ili problema u procesiranju SACK opcija na strani primatelja. |
| **TcpExtTCPFromZeroWindowAdv** | Ovaj brojač bilježi broj puta kada TCP primatelj oglašava (*advertises*) svoj prozor veličine nula (*zero window*) prema TCP pošiljatelju. Prozor veličine nula signalizira da TCP primatelj trenutno ne može primiti nove podatke jer je prijemna međumemorija ispunjena ili je sustav preopterećen. Povećanje vrijednosti ***sysctl*** varijable: `net.core.rmem_max` može pomoći. |
| **TcpExtTCPZeroWindowDrop** | Ovaj brojač bilježi broj TCP paketa koji su odbačeni zbog toga što je primatelj oglašavao prozor veličine nula (*zero window*). Kada primatelj oglašava prozor veličine nula, to znači da trenutno nije spreman primiti nove podatke od pošiljatelja jer mu je prijemna međumemorija ispunjena ili je sustav preopterećen. Povećanje vrijednosti ***sysctl*** varijable: `net.core.rmem_max` može pomoći. |
| **TcpExtTCPOFODrop** | Ovaj brojač bilježi broj TCP segmenta koji su odbačeni zbog dolaska izvanrednih (*out-of-order*) segmenata. Kada red čekanja postane prevelik ili ako izvanredni segmenti postanu zastarjeli, TCP primatelj može odlučiti odbaciti neke od tih segmenata kako bi spriječio prekomjerno opterećenje. Povećanje vrijednosti ***sysctl*** varijable: `net.core.wmem_max` može pomoći. |

Statistike s problemima vezanim za ***sockete*** možemo vidjeti s: **`netstat -s | egrep -i "lock|socket\ buf"`**.
**Izvori informacija: (340),(1406),(K-6)**,**`man 7 tcp`**,**`man nstat`**,**RFC793**, **RFC 1122.**

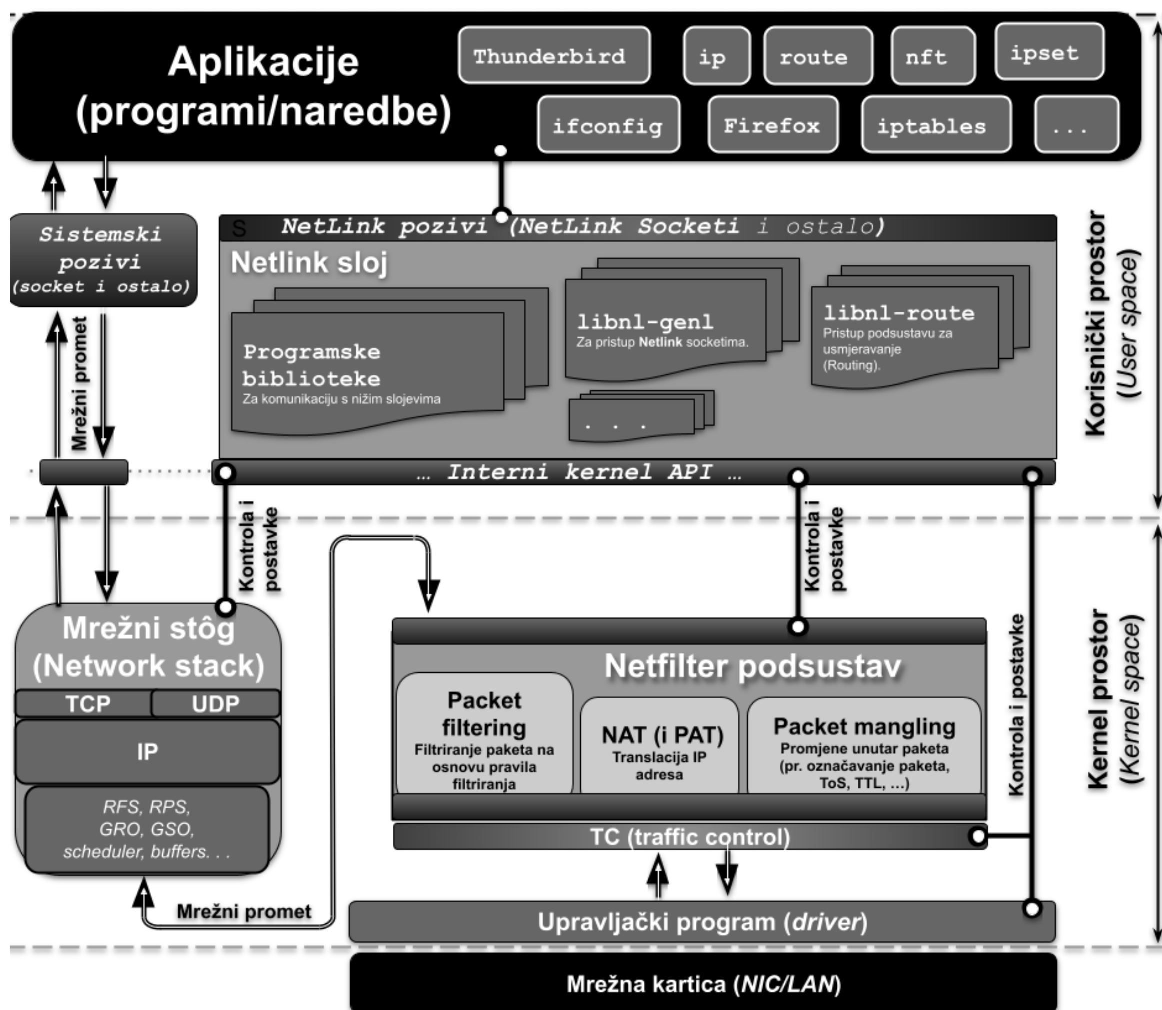
# 25. Mreža u Linuxu

U ovom većem poglavlju, pozabavit ćemo se s Linuxovim mrežnim modelom i načinom komunikacije između aplikacija i mrežnog sustava te u konačnici i s mrežom. Dakle govorit ćemo uglavnom o mrežnom stôgu Linuxa.

## 25.1. Linux mrežni model

Prije nego krenemo s upoznavanjem s mrežom pod Linuxom i detaljnom konfiguracijom mreže, bilo bi dobro razumjeti mrežni model koji koristi Linux. Važna specifičnost mrežnog podsustava u odnosu na ostale podsustave poput diskovnog je i u tome što se mrežni uređaji **ne** nalaze u `/dev/` direktoriju gdje se inače nalaze svi uređaji u Linuxu ili bilo kojem *UNIX-u*.
To je tako jer je mrežna komunikacija puno složenija od primjerice diskovne, koja se svodi na operacije čitanja ili zapisivanje na disk, a dodatno se ne čitaju ili ne zapisuju bajti podataka fiksne veličine, već (mrežni) paketi varijabilne veličine.
Dodatni problem je i primanje paketa različitim brzinama, zbog čega postoji potreba za privremenim pohranjivanjem mrežnih paketa u međuspremnik koji bi u nekim trenucima mogao postati prepunjen te bi se paketi koji ne bi stali u spremnik, tada izgubili. Još jedna od komplikacija su specifični mrežni protokoli i mehanizmi za filtriranje (vatrozid) i nadzor mrežnog prometa, logička i virtualna mrežna sučelja i slično, što sve zajedno uvodi potrebu za znatno kompleksnijim mrežnim modelom.
Stoga se mrežni model u Linuxu sastoji od nekoliko komponenti odnosno slojeva, kao što je vidljivo na slici 228.

**Slika 228. Mrežni model Linuxa.**






Naredbe iz paketa `net-tools` poput naredbe `ifconfig` ne komuniciraju s Linux kernelom preko ***Netlink*** sloja, već pomoću ***sistemskih poziva*** stoga imaju razna ograničenja. Jedno od njih je nemogućnost naredbe `ifconfig` da dodjeli više od jedne IP adrese pojedinom mrežnom sučelju (*interface*); primjerice `eth0`, već samo mrežnim pôd sučeljima (engl. *subinterface*) kao što je primjerice: `eth0:0`.

Istovremeno novija generacija alata iz paketa `iproute2` poput naredbe `ip` nema takvih problema, te uredno može dodijeliti ili ispisati sve IP adrese na svakom mrežnom sučelju jer komunicira preko ***Nelink*** sloja direktno s mrežnim slojevima Linuxa.

Komunikacija između aplikacija i mrežnog podsustava može se odvijati samo i isključivo na dva načina:
- Pomoću ***Netlink poziva (pr. Netlink socketa).***
- Pomoću ***sistemskih poziva***.

Pri tome su ***Netlink*** sloj i/ili ***sistemski pozivi*** sučelje između korisničkih programa/aplikacija i internih funkcija kernela, prema mrežnom stôgu Linuxa ili prema specifičnim *kernel modulima* i drugim dijelovima kernela zaduženim za razne logičke slojeve mrežne komunikacije.



Za više detalja o ***Netlink i Netfilter*** sustavu te ***sistemskim pozivima***, više u poglavlju:
**26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije.**



Važno je razumjeti da ***Netlink*** komunikacija ne može preko mreže komunicirati s vanjskim entitetima, što znači da se koristi i može djelovati samo unutar računala. To istovremeno znači da se komunikacije aplikacija preko mreže može ostvariti samo pomoću sistemskih poziva odnosno preko tzv. ***network socketa***.

## Što je ***mrežni socket*** odnosno takozvana *mrežna utičnica* ?

*Sistemski pozivi* omogućuju aplikacijama interakciju s mrežnim slojevima operativnog sustava kako bi slale i primale podatke preko mreže. *Sistemski pozivi* pružaju apstrakcijski sloj između aplikacija i kernela koji ima sučelje i mehanizme pristupa prema složenih mrežnim protokolima, omogućujući jednostavnu mrežnu komunikaciju bez potrebe za direktnim upravljanjem detaljima nižih mrežnih slojeva. Pri tome kernel ima ključnu ulogu u mrežnoj komunikaciji jer:
- Upravlja mrežnim stôgom Linuxa i protokolima niže razine poput transportnih protokola (TCP i UDP) te još nižeg sloja odnosno IP protokola (u TCP/IP mrežnom modelu), kao i svih još nižih slojeva mrežnog podsustava.
    - Rješava probleme poput gubitka paketa, ponovnog slanja i kontrole zagušenja: primjerice kod TCP protokola.
- Održava međuspremnike za prijenos i primanje podataka preko mreže.
- Osigurava zaštitu kroz mehanizme kao što su vatrozidi ili filtriranje paketa i druge značajke nižih slojeva mrežnog podsustava.

Vratimo se na *mrežne utičnice*. Možemo reći i ovako: mrežne utičnice (*network socketi*) su posebne vrste sistemskih poziva koje operativni sustav koristi za komunikaciju između aplikacija preko mreže. Konkretno *mrežni socketi* pripadaju kategoriji sistemskih poziva za mrežni podsustav. *Mrežni socket* je poput "krajnje točke" u mrežnom komunikacijskom kanalu jer svaka aplikacija koja komunicira preko mreže koristi *mrežni socket* kao ulaznu i izlaznu točku za mrežnu komunikaciju. *Mrežni socket* definira izvorišnu i odredišnu IP adresu i broj porta, koji zajedno predstavljaju jedinstveni entitet za svaku pojedinu mrežnu komunikaciju.

**Koje su prednosti upotrebe mrežnih utičnica (*network socketa*):**
- **Apstrakcija složenosti** – aplikacije koriste jednostavne funkcije za otvaranje *socketa*, slanje i primanje podataka, dok kernel upravlja detaljima protokola (TCP, UDP, IP) te drugih mehanizama unutar mrežnog podsustava.
    - **Fleksibilnost komunikacije** – *socketi* podržavaju različite protokole poput TCP-a za pouzdanu komunikaciju i UDP-a za brzu, nepouzdanu komunikaciju.
- **Platformska neovisnost** – *socket API* je standardiziran (**POSIX**), omogućujući rad aplikacija na različitim platformama ili inačicama operativnih sustava s minimalnim izmjenama.
- **Optimizacija resursa** – kernel osigurava efikasno korištenje memorije i mrežnog prometa putem optimizacija poput **Nagle** algoritma ili kontrola protoka.
- **Asinkroni rad** – *mrežni socketi* omogućuju neblokirajuće operacije, omogućavajući aplikacijama da obrađuju više zahtjeva istovremeno.
- **Skalabilnost** – *socketi* podržavaju veliki broj istovremenih veza, što ih čini idealnim za poslužitelje i aplikacije s visokim opterećenjem. Pri tome operativni sustav pruža mehanizme poput `epoll` ili `select` za učinkovito upravljanje tisućama *socketa*.
- **Asinkroni i višenitni rad** - *socketi* podržavaju neblokirajuće operacije i višenitne modele, omogućujući aplikacijama da procesiraju više zahtjeva istovremeno bez blokiranja resursa.
- **Sigurnost i izolacija** – kernel štiti aplikacije od međusobnog ometanja, omogućujući izolaciju *socketa* između procesa.



Za više detalja o ***network socketima*** pogledajte poglavlja:
**9.4.3.2. Network socketi.**
**24. Transportni protokoli (OSI sloj 4).**

**Izvori informacija: (323),(325),(340),(365),(778),(1311),(1534), `man 7 tcp`, `man 7 udp`, `man 2 socket`, plakat P5.**

## 25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*)

U ovoj ***naprednoj*** cjelini pregledat ćemo već okvirno navedene slojeve komunikacije unutar mrežnog modela Linuxa, ali na malo drugačiji način, prilagođen OSI odnosno TCP/IP mrežnom modelu. Usporedbu ćemo raditi stoga, kako bi vam bile jasnije međuveze između raznih slojeva u mrežnoj komunikaciji unutar sâmog računala, a kako je vidljivo na slici 229.

**Slika 229. Slanje i primanje paketa na mreži, gledano kroz mrežne slojeve Linuxa.**

Slanje podataka
Aplikacija
Socket međumemorija za slanje (SKB)
Mrežni stôg (*network stack*)
*Socket*
*TCP i UDP*
*IP*
Netfilter (*filtriranje paketa*)
Niz za slanje (queue) + Qdisc/scheduler
Upravljački program (driver)
TX prstenasta memorija (*ring buffer*)
Mrežna kartica (*NIC/LAN*)
Interna memorija (fiksna)
Ethernet flow control (slanje na "žicu")

Primanje podataka
Aplikacija
Socket međumemorija za primanje (SKB)
Mrežni stôg (*network stack*)
*Socket*
*TCP i UDP*
*IP*
NAPI (poll) (No queue)
Međumemorija (buffer)
ILI
Netfilter (*filtriranje paketa*)
Upravljački program (driver)
RX prstenasta memorija (*ring buffer*)
Mrežna kartica (*NIC/LAN*)
Interna memorija (fiksna)
Ethernet flow control (primanje sa "žice")

U mrežnoj komunikaciji, u prvom koraku, aplikacija koja želi podatke slati na mrežu (gornji lijevi dio slike) priprema podatke i prosljeđuje ih u međuspremnik za slanje (*Send socket buffer [**SKB**]*). Ako kojim slučajem nema slobodnog prostora u tom međuspremniku odnosno međumemoriji, poziv sustava kojim se odrađuje ova operacija nije uspio te dolazi do blokiranja zbog čekanja na pražnjenje ovog međuspremnika. Operacija slanja podataka na mrežu u prvom koraku se obično odrađuje od strane sâme aplikacije. Mehanizme ove kontrole sada nećemo spominjati



Brzina slanja aplikacijskih podataka koji se šalju na ovaj međuspremnik (***Send socket buffer***) se mora kontrolirati.

Zatim kada su podaci prošli *socket* međuspremnik (**SKB**), prvo dolaze do mrežnog stôga Linuxa, a u njemu i do transportnog sloja, primjerice do **TCP**-a ili **UDP-a**. On ih priprema i šalje nižim slojevima: prvo na **IP** sloj, koji odrađuje svoj dio, pa sve šalje prema nižem sloju odnosno prema **Netfilter-u**.

***Netfilter*** je zadužen za filtriranje mrežnih paketa odnosno implementaciju vatrozida na najnižoj razini, koji ovisno o trenutnoj konfiguraciji, odlučuje što s paketom: proslijediti ga, odbaciti i/ili snimiti poruku o njemu. Tada sve dolazi do nîza za slanje (*queue*), iz kojega mehanizam za raspodjelu mrežnih paketa (***QDISC/Scheduler***) uzima pakete i šalje ih prema upravljačkom programu mrežne kartice koji sve šalje na mrežu.

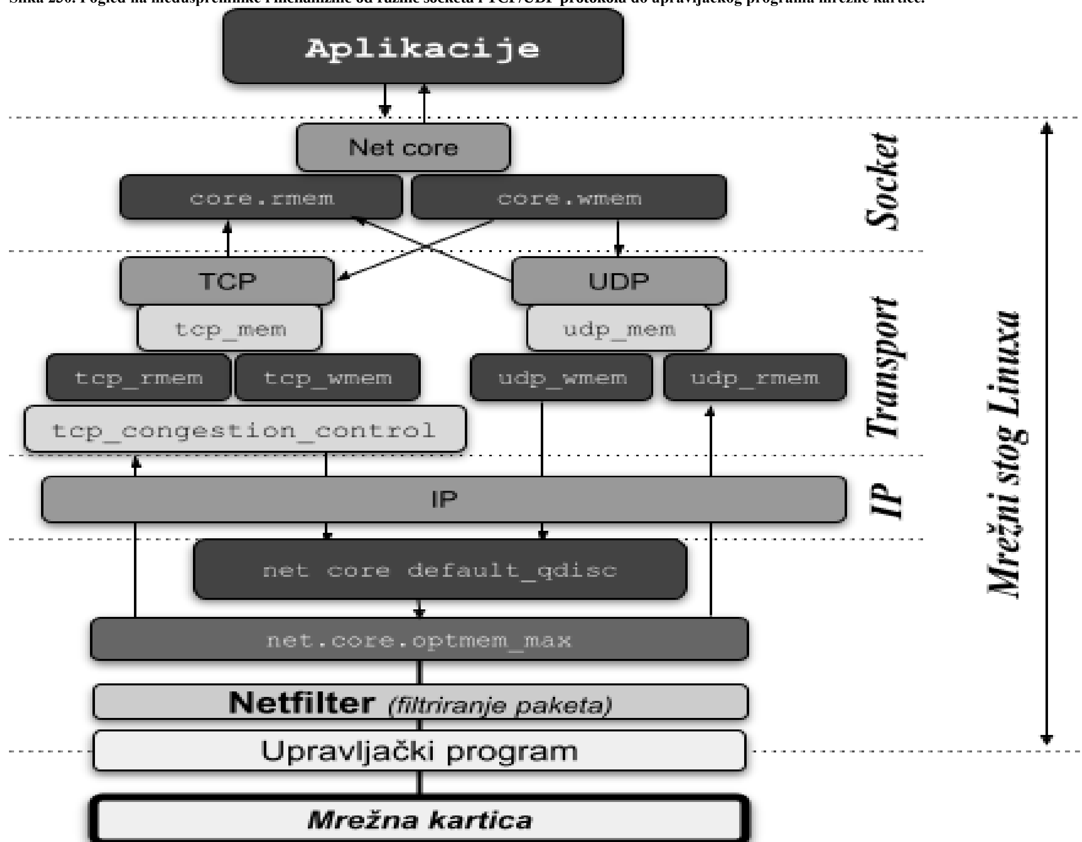


Vezano za detalje oko rada mehanizma za raspodjelu mrežnih paketa, pogledajte poglavlje:
**25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa.**

Za detalje o međuslojevima i mehanizmima koje zbog lakšeg razumijevanja nismo opisali, pogledajte poglavlja:
**25.2. Dodatne mogućnosti linuxa.**
**25.5. Hardverski ubrzane mrežne funkcionalnosti.**

Pogledajmo pojednostavljeno koje međuspremnike i mehanizme (te optimizacije) imamo dostupne u slojevima između aplikacije i upravljačkog programa mrežne kartice (engl. *drivera*).

**Slika 230. Pogled na međuspremnike i mehanizme od razine socketa i TCP/UDP protokola do upravljačkog programa mrežne kartice.**




Pogledajmo i što sve možemo optimizirati pomoću ***sysctl*** varijabli, na ovoj razini (uz opise):

| ***Sysctl*** **naziv varijable** | **Komentar** | **Zadana vrijednost varijable (bajta)** |
|---|---|---|
| `net.core.rmem_max` | Definira maksimalnu moguću veličinu *socket buffera* kod **primanja** (*receive*) paketa s mreže, za sve mrežne protokole za svaki socket (dakle po ***socketu***), koju aplikacija može zatražiti.<br>Postavlja gornju granicu za **SO_RCVBUF** socket opciju. | 16777216 |
| `net.core.rmem_default` | Definira standardnu veličinu *socket buffera* kod **primanja** (*receive*) paketa s mreže za sve mrežne protokole, koja će biti u upotrebi za svaki socket (dakle po ***socketu***). Ako aplikacija eksplicitno ne zatraži drugačiju veličinu, koristit će se ova vrijednost. Može se eventualno povećavati (do `rmem_max`), ovisno o mreži i namjeni računala (poslužitelja)(*) | 1048576 |
| `net.core.wmem_max` | Definira maksimalnu moguću veličinu *socket buffera* kod **slanja** (*write*) paketa na mrežu za sve protokole za svaki socket (dakle po ***socketu***). Postavlja gornju granicu za **SO_SNDBUF** socket opciju. | 16777216 |

| *Sysctl* naziv varijable | Komentar | Zadana vrijednost varijable (bajta) |
|---|---|---|
| `net.core.wmem_default` | Opcija kernela koja definira standardnu veličinu *socket buffera* kod **slanja** (*write*) paketa na mrežu, koja će biti u upotrebi za svaki socket (dakle po ***socketu***). Ako aplikacija eksplicitno ne zatraži drugačiju veličinu, koristit će se ova vrijednost. Može se eventualno povećavati, ovisno o mreži i namjeni računala (poslužitelja)[(*)] | `1048576` |
| `net.core.optmem_max` | Ovdje se definira memorija dodijeljena za dio mrežnih paketa koji sadrži dodatne informacije i kontrolne podatke poput mrežnog sučelja, dodatnih zaglavlja i slično, za svaki socket (dakle po ***socketu***). Ona ne predstavlja ukupnu međumemoriju za pakete, kao što to možda izgleda. U praksi se može povećavati, a neki preporučaju kao optimum, povećanje do 65535 bajta (64kB)[(*)] | `20480` |

Za **TCP** transportni protokol imamo sljedeće opcije odnosno ***sysctl*** varijable koje možemo optimizirati:

| *Sysctl* naziv varijable | Komentar | Zadana vrijednost varijable |
|---|---|---|
| `net.ipv4.tcp_mem` | Ovdje se definira ukupna količina memorije koju TCP stôg može koristiti na razini cijelog sustava, dok ostale varijable kontroliraju memoriju po pojedinoj konekciji. Ova vrijednost uključuje sve TCP *sockete* (konekcije) koji se trenutno koriste. [(*)]<br>Ovdje imamo tri vrijednosti (broja):<br>○ **Min** – minimalni broj stranica memorije koji sustav garantira TCP *stôgu*. Sustav neće pokušavati osloboditi memoriju ispod ove razine zauzeća. *TCP stôg* uvijek ima na raspolaganju barem ovu količinu memorije.<br>○ **Pressure** – kada potrošnja memorije dosegne ovaj prag, kernel aktivira mehanizme za oslobađanje memorije poput:<br>  ○ Smanjivanja veličina *buffera* za nove konekcije.<br>  ○ Agresivnije oslobađanje memorije te smanjivanje veličine TCP prozora (**TCP Window**) gdje je to moguće.<br>○ **Max** – apsolutni maksimum stranica memorije koje TCP stôg može koristiti. Nakon ovog praga događa se sljedeće:<br>  ○ Nove TCP konekcije će biti odbijene a aplikacije mogu dobiti **ENOBUFS** greške.<br>  ○ Pokušaji povećanja *buffer*-a mogu biti odbijeni.<br>  ○ Postojeće konekcije mogu dobiti manje *buffere* nego što su zatražile te će samim time dobiti smanjivanje veličine TCP prozora (**TCP Window**).<br>  ○ Moguće smanjenje propusnosti. | **Prvi** broj: `94389` → **370MB**<br>**Drugi** broj: `125852` → **490MB**<br>**Treći** broj: `188778` → **740MB**<br><br>***Ove vrijednosti su u stranicama memorije od 4096 bajta (4kB)a ne u bajtima*** |
| `net.ipv4.tcp_rmem` | Ovo su vrijednosti međumemorije za **primanje** paketa. Prva vrijednost govori kernelu koliki je minimalni međuspremnik za svaku TCP vezu, a taj je međuspremnik uvijek dodijeljen svakom TCP *socketu*, čak i pod velikim opterećenjem sustava. Druga navedena vrijednost je standardno kernelu zadana veličina međumemorije koja se dodjeljuje za svaki *socket*. Treća i zadnja vrijednost navedena u ovoj varijabli određuje maksimalni prijemni međuspremnik koji se može dodijeliti za TCP socket. [(*)] | **Prvi** broj: 4096 → **4kB**<br>**Drugi** broj: 11796480 (11MB)<br>**Treći** broj: 11796480 (11MB)<br><br>*Ove vrijednosti su u bajtima* |
| `net.ipv4.tcp_wmem` | Ovo su vrijednosti međumemorije za **slanje** paketa. Prva vrijednost govori kernelu koliki je minimalni međuspremnik za svaku TCP vezu, a taj je među spremnik uvijek dodijeljen svakom TCP *socketu*, čak i pod velikim opterećenjem sustava. Druga navedena vrijednost je standardno kernelu zadana veličina među memorije dodijeljena za svaki *socket*. Ova vrijednost nadjačava vrijednost iz: `/proc/sys/net/core/wmem_default` koja se koristi i u drugim protokolima. Treća odnosno zadnja vrijednost navedena u ovoj varijabli određuje maksimalni međuspremnik koji se može dodijeliti za TCP *socket* za slanje mrežnih paketa. [(*)] | **Prvi** broj: 4096 (4kB)<br>**Drugi** broj: 11796480 (11MB)<br>**Treći** broj: 11796480 (11MB)<br>*Ove vrijednosti su u bajtima* |

Za **UDP** transportni protokol imamo sljedeće opcije odnosno ***sysctl*** varijable koje možemo optimizirati:

| *Sysctl* naziv varijable | Komentar | Zadana vrijednost |
|---|---|---|
| `net.ipv4.udp_mem` | Ova opcija određuje ukupnu količinu memorije koju kernel može koristiti za sve UDP *sokete* na sustavu. Definira se pomoću tri vrijednosti (identično kao i `net.ipv4.tcp_mem`)(*). | **Prvi** broj: 12384288<br>**Drugi** broj: 16512384<br>**Treći** broj: 24768576<br><br>*Ove vrijednosti su u stranicama memorije od 4096 bajta (4kB)* |
| `net.ipv4.udp_rmem_min` | Ova opcija postavlja minimalnu veličinu memorijskog međuspremnika za prijem (***receive buffer***) za svaki pojedinačni *UDP soket*. *UDP soketi* koriste ovaj međuspremnik za pohranu podataka koji stižu putem mreže dok aplikacija ne preuzme te podatke. Povećanje ove vrijednosti može pomoći aplikacijama koje primaju velike količine podataka putem UDP-a. | 4096<br>(4kB)<br><br>*Ova vrijednost je u bajtima* |
| `net.ipv4.udp_wmem_min` | Služi kao minimalni memorijski međuspremnik za slanje (***send buffer***) za svaki UDP *socket*. Ovaj međuspremnik koristi se za privremeno pohranjivanje podataka koje aplikacija šalje putem mreže dok ih mrežni stog ne prenese. Povećanje ove postavke korisno je za aplikacije koje šalju velike količine podataka putem UDP-a | 4096<br>(4kB)<br><br>*Ova vrijednost je u bajtima* |
| `net.core.default_qdisc` | Ovdje se definira ***network scheduler*** (***qdics***) koji će se koristiti. | Postoji ih cijeli niz |

Već smo naučili kako pomoću naredbe `sysctl` ili unosa u datoteci `/etc/sysctl.conf` možemo mijenjati navedene vrijednosti. Više detalja o navedenim i svim onim nenavedenim slojevima mreže u Linuxu, vidjet ćete u nastavku.



Vezano za upotrebu naredbe **`sysctl`**, pogledajte poglavlje:
**4.5.8. Naredba *sysctl*.**

Preporuke i dodatne informacije:

- `net.core.rmem_max` – ova vrijednost se inicijalno po
- `net.ipv4.tcp_mem` – ova vrijednost se inicijalno podešava prilikom svakog pokretanja sustava ovisno o količini dostupne memorije. Naime, prilikom podizanja sustava, kernel će provjeriti dostupnu memoriju na sustavu i na temelju dostupne memorije će dodijeliti pripadajuće vrijednosti u: `/proc/sys/net/ipv4/tcp_mem` **(1769).**
  - ***PRESSURE*** vrijednost treba biti ~20-30% veća od MIN.
  - ***MAX*** vrijednost treba biti 2x (200%) veća od PRESSURE.
- Vrijednosti treba općenito prilagoditi prema:
  - Dostupnoj sistemskoj memoriji.
  - Broju očekivanih konekcija.
  - Tipičnim veličinama *buffera*.
  - Prirodi aplikacijskog prometa.

Važno je razumjeti ovu hijerarhiju:
**1. NET CORE** – u varijablama: `net.core.rmem_max` i `net.core.wmem_max` se definiraju maksimalne vrijednosti *buffera* (međumemorija) za svaki pojedini *socket*. Dakle ovdje se definira **globalni maksimum** po POJEDINOM *socketu* (bilo koje vrste, ne samo TCP ili UDP. Dakle ovo je vršno ograničenje koje se ne može preći niti za jednu vrstu *socketa* (**TCP**, **UDP**, **SCTP**, …).

**2. NET IPV4: TCP** – u varijabli: `net.ipv4.tcp_mem` se definira **globalno ograničenje** ukupne memorije za SVE **TCP** konekcije. **UDP** – u varijabli: `net.ipv4.udp_mem` se definira **globalno ograničenje** ukupne memorije za SVE **UDP** konekcije. Ove vrijednosti mogu ograničiti veličinu TCP prozora (*TCP Window size*) ukoliko se dosegne MAX vrijednost.

**3. NET IPV4 TCP** RMEM ili WMEM– u varijablama: `net.ipv4.tcp_rmem` i `net.ipv4.tcp_wmem` se definiraju TCP specifična ograničenja po POJEDINOM TCP *socketu*. Ove vrijednosti moraju biti manje ili jednake: `net.core.rmem_max` i `net.core.wmem_max` (ovisno radi li se o slanju ili primanju). Ove vrijednosti direktno ograničavaju maksimalnu veličinu TCP prozora (*TCP Window size*).



Vezano za veličinu ***TCP prozora*** pogledajte poglavlje:
**24.2.8. Kontrola protoka (*TCP Window* i *Window scaling*).**

**Izvori informacija: (323),(325),(340),(365),(1311),(1406),(1590),(1769),(1772), `man 7 tcp`, `man 7 udp`, plakat P5.**

## 25.1.2. Optimizacija TX međumemorije (`txqueuelen`) mrežnog sučelja

Vratimo se na niže slojeve mrežnog sustava. Nakon ***TCP, IP*** i ***Ethernet*** sloja svi podaci se šalju na još niže slojeve, prema nîzu za slanje (*transmit queue*) ili nîzu za primanje (*receive queue*), ovisno šaljemo li ili primamo mrežne pakete.
Međutim, ispod razine transportnog (***TCP/UDP)*** sloja postoji mehanizam koji se brine o tome kako će se mrežni paketi prosljeđivati niže prema upravljačkom programu odnosno niže prema mreži.
Ovaj mehanizam ima i nekoliko podvarijanti, od kojih sâmo jedna može biti aktivna.
Navedeni mehanizam se zove ***queuing scheduler*** ili ***qdisc*** odnosno mehanizam za raspodjelu mrežnih paketa.



Za više detalja o ***queuing scheduler*** (***qdisc)*** mehanizmu pogledajte poglavlje:
**25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa.**

Na razini ispod navedenog mehanizma za raspodjelu mrežnih paketa nalazi se sloj u kojemu je nîz odnosno međumemorija za slanje (*TX*), a koja je fiksno definirane veličine za sve pakete, a u Linuxu se naziva: `txqueuelen`.
Vezano za sâm međuspremnik na ovom sloju Linuxa (`txqueuelen`), on se definira za svako mrežno sučelje zasebno te je vidljiv pomoću naredbi `ifconfig` ili `ip`.
Pogledajmo veličinu `txqueuelen` koja se definira za ***broj paketa*** koji mogu stati u taj nîz (ispis je skraćen).

```
ifconfig eth0

eth0: flags=6211<UP,BROADCAST,RUNNING,SLAVE,MULTICAST>  mtu 1500
        ether d8:9d:67:62:1d:e0  txqueuelen 1000  (Ethernet)
```

← Vrijednost `txqueuelen` je obično postavljena na `1000` odnosno za pohranu do 1.000 mrežnih paketa:

Odnosno ovaj nîz je vidljiv i s naredbom: `ip link show` kao `DEFAULT qlen`:

```
ip link show eth0

2: eth0:<BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond0 state
UP mode   DEFAULT qlen 1000 link/ether d8:9d:67:62:1d:e0 brd ff:ff:ff:ff:ff:ff
```

Naravno ovu vrijednost možemo privremeno promijeniti s istim naredbama; primjerice povećati na 2.000.



Povećanje ne znači nužno dobar potez jer je potrebno dobro razumijevanje i analiza sustava prije njegove promjene. Većim povećanjem ove međumemorije možemo uvesti i kašnjenje odnosno latenciju u obradi paketa!.

U ovom primjeru ga postavimo odnosno povećajmo na `2000`:

```
ip link set eth0 txqueuelen 2000
```

Odnosno trajno, kreiranjem ***udev rules*** datoteke, poput nove datoteke: `/etc/rules.d/80-net-txqueuelen.rules` s kojom bi se trajno mogla promijeniti ova veličina za sva `eth`*XY* mrežna sučelja, dodavanjem slijedećeg retka i to za:

**RedHat/CentOS 6.x** (za mrežne kartice imena: `eth*`):

```
SUBSYSTEM==”net”, ACTION==”add”, KERNEL==”eth*”, ATTR{tx_queue_len}=”2000”
```

Ili za ***RedHat/CentOS* 7.x+**, nešto poput (za mrežne kartice: `em0`, `em1` i `em2`) *[varijabla* **`%k`** *predstavlja uređaj/karticu]*:

```
KERNEL==“em[0,1,2]“, RUN+=“/sbin/ip link set %k txqueuelen 2000“
```



Za neke od primjera mrežnih sučelja kojima se može promijeniti ***txqueuelen***, pogledajte poglavlja:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**
**20.6.3. TUN i TAP - posebna mrežna sučelja.**
**20.6.4. VETH - posebno mrežno sučelje.**
**20.6.6.2. Konfiguracija *bondinga* u Linuxu.**
**26.8.2. Druga mrežna sučelja (*MACVLAN,IPVLAN,VXLAN,MACVTAP/IPVTAP*).**

Pogledajte i optimizacije vezane uz virtualizaciju i **TAP** mrežno sučelje:
**27.1.2.2. Praćenje i analiza rada *TAP* sučelja.**

**Izvori informacija: (321), `man ip link`, plakat: P5.**

### 25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*]

**Slijedi napredna cjelina!**
Kako bismo sve još dublje razumjeli, spustimo se na još niže slojeve mreže.
Vratimo se nazad na komunikaciju i sliku **229** s početka poglavlja.
Sada kada su podaci u `txqueuelen` nîzu za slanje, upravljački program (*driver*) ih potom dohvaća i zapisuje u posebnu takozvanu prstenastu memoriju odnosno tzv. *TX ring*, koji mrežna kartica vrlo brzo može dohvatiti i pripremiti, a potom pohraniti u svoju posebnu internu ekstremno brzu međumemoriju (*NIC internal buffer*) koja se nalazi na sâmoj mrežnoj kartici. Kako se ova vrlo brza interna međumemorija mrežne kartice popuni s većim brojem paketa, mrežna kartica ih može, sve od jednom u nîzu i poslati vrlo brzo na mrežu.
Sâmu proceduru obrade ćemo objasniti u proceduri primanja paketa.

Naime na ovom sloju mrežnog sustava, tijekom primanja mrežnih paketa (desni dio slike **229**) koji prvo dolaze do interne međumemorije mrežne kartice (*NIC internal buffer*) upravljački program pokreće zahtjev za prijenos paketa u trenutku dok CPU izvršava neki drugi zadatak, bez čekanja odgovora od strane CPU-a. Potom mrežna kartica šalje pakete i obavještava CPU o tome. S druge strane, procedura je ista i kod slanja paketa. Dakle paketi se šalju i primaju asinkrono. Prvo upravljački program zahtjeva primanje paketa dok CPU obavlja neki drugi zadatak. Zatim mrežna kartica prima pakete i obavještava CPU o tome, a upravljački program obrađuje primljene pakete. Kako bi sve to bilo moguće, potreban je određeni memorijski prostor u koji se ti paketi mogu zapisivati, stoga mrežna kartica koristi strukturu znanu kao prstenasta memorija odnosno „*ring*“ (točnije *RX ring za* primanje paketa). Ovaj prsten je sličan „*queuing*“ nîzu odnosno strukturi reda čekanja na dohvaćanje.

Kod ove prstenaste strukture međuspremnika, uz fiksni broj unosa, jedan unos sprema jedan zahtjev ili jedan odgovor. Zapisi se redom koriste jedan za drugim. Važno je znati kako postoje dva prstena memorije: jedan za primanje (*RX ring*) i jedan za slanje (*TX ring*). Uobičajeno se koristi naziv „prsten“ budući da se fiksno definirana polja za unose ponovno koriste u krug: kako se redom prazne, tako ponovno postaju dostupna za popunjavanje.
Tijekom primanja paketa sve se događa obrnuto od slanja.



Upravljački program (*driver*) mrežne kartice i sâma mrežna kartica (***LAN***/***NIC***) komuniciraju ***asinkrono***.

Na Linuxu je moguće vidjeti i mijenjati (unutar granica hardvera i upravljačkog programa), veličinu *RX* i *TX ring* nîzova odnosno međumemorija. Pogledajmo kako su oni postavljeni, pomoću naredbe `ethtool` na sljedeći način:

```
ethtool -g eth0

Ring parameters for eth0:
Pre-set maximums:
RX:             4078
RX Mini:        0
RX Jumbo:       0
TX:             4078
Current hardware settings:
RX:             1333
RX Mini:        0
RX Jumbo:       0
TX:             4078
```

Ovdje imamo vidljive dvije važne stvari:

- Definirano je koliki je maksimum koji uopće možemo konfigurirati, a ovisno o hardveru mrežne kartice i njenom upravljačkom programu. To je vidljivo u opciji: `Pre-set maximums`.
- Vrijednost koja je trenutno konfigurirana, na razini hardvera mrežne kartice: `Current hardware settings`.

Ove vrijednosti su brojevi deskriptora paketa odnosno indirektno broja paketa koji stanu u nîz odnosno *ring* (*TX* ili *RX*).
Naravno ove *prstenaste memorije* mrežne kartice možemo i mijenjati, sve do granice odnosno maksimalnih vrijednosti definiranih u dijelu `Pre-set maximums`, što je i hardversko ograničenje kartice.

U našem slučaju je maksimalna vrijednost i za RX i TX: 4.078.
Mi ćemo ih sada povećati upravo do te vrijednosti, uz istu napomenu kao i prije:



Povećanje ne znači nužno dobar potez te je potrebno dobro razumijevanje i analiza prije njegove promjene.

Sada povećajmo obje prstenaste međumemorije (i TX i RX *ring buffera*) na 4.078, korištenjem prekidača (veliko slovo `G`):

```
ethtool -G eth0 rx 4078 tx 4078
```

I sada provjerimo njihovo novo stanje (malo slovo `G`):

```
ethtool -g eth0
```

```
Ring parameters for eno4:
Pre-set maximums:
RX:             4078
RX Mini:        0
RX Jumbo:       0
TX:             4078
Current hardware settings:
RX:             4078
RX Mini:        0
RX Jumbo:       0
TX:             4078
```

Vidimo kako su se sada oba *prstenasta spremnika* (*prstenaste memorije*) povećala na 4.078, što smo i htjeli.



***Daleko najčešći razlog gubitka mrežnih paketa/okvira je prepunjavanje međumemorije.***

Kako smo vidjeli, kernel postavlja ograničenja na veličinu ovih međumemorija, a u nekim slučajevima međumemorija se puni brže nego što se stigne prazniti. Kada se to počne događati, nakon nekog vremena ili odmah, mrežni paketi odnosno mrežni okviri se počinju odbacivati i u konačnici gubiti.

Kod dolaznog prometa koji prebrzo dolazi, možemo ga ili usporiti ili povećati veličinu međumemorije u koju se privremeno sprema. Međutim prevelikim povećanjem ove međumemorije unosimo dodatno kašnjenje u obradi primljenih paketa, jer je potrebno više vremena da paket koji je ušao u međumemoriju i izađe iz nje i obradi se. Stoga se vrlo često radi o dobrom balansiranju između zadovoljavajućeg kašnjenja i gubitka paketa te njihove retransmisije.
U takvim slučajevima, dodatno je moguće i ubrzati pražnjenje međumemorije.
Ova mogućnost se optimizira na međumemoriji mrežne kartice (*ring buffera*).

Na razini mrežne kartice postoji varijabla: `/proc/sys/net/core/dev_weight` u kojoj se definira broj mrežnih okvira (paketa) koje mrežna kartica može prihvatiti u svoju međumemoriju, prije nego li ih se sve odjednom može dohvatiti preko ***softirq*** signala u slučaju *POOLING* načina rada mrežne kartice. Ova opcija je također dostupna preko `sysctl` varijable: `net.core.dev_weight` pa ju s njome možemo i mijenjati. Njena standardna vrijednost je 64 (*po CPU jezgri*) što znači kako će se do maksimalno 64 paketa odjednom moći dohvatiti pomoću *softIRQ* + *NAPI pooling* mehanizma u jednom *NAPI* ciklusu dohvaćanja (ako imamo NAPI kompatibilan upravljački program mrežne kartice).

## LRO i GRO

U slučajevima kada upravljački program i mrežna kartica podržavaju i koriste, odnosno kada su na njima uključene opcije: ***LRO*** ili ***GRO*** to znači da broj 64 označava 64 agregirana odnosno povezana nîza mrežnih paketa.



Pogledajte i poglavlje o **LRO** i **GRO:**
**25.5.1. Large receive offload (*LRO*) i Generic receive offload (*GRO*).**

Pogledajmo i kako primjerice povećati ovu vrijednost na recimo 128. Malo povećanje može čak i smanjiti latenciju (kašnjenje), ali i povećati opterećenje CPUa, kod nekih sustava. (***Oprez kod promjene ove vrijednosti***):

```
sysctl net.core.dev_weight=128
```



Pogledajte i poglavlje o optimizaciji signala prekida**:**
**10.5.2.2 Interrupt moderation.**

Na razini od upravljačkog programa prema redovima čekanja (*queuing)*, moguće je definirati i maksimalni broj paketa koji se mogu spremiti u posebnu međumemoriju prema ovom redu čekanja. Problemi se mogu dogoditi kada određena mrežna sučelja (pr. mrežna kartica `eth0`) primaju pakete brže nego što ih kernel (računalo) može obraditi. Standardno postavljena vrijednost ove međumemorije je 1.000. Ona označava broj paketa koji se mogu spremiti u ovu međumemoriju u slučajevima kada je računalo (CPU) preopterećeno s obradom te kernel ne stiže obraditi mrežne pakete.
Ova međumemorija je zajednička za sva mrežna sučelja, tijekom dohvaćanja mrežnih paketa (***RX***). Ona je vidljiva kao *sysctl* opcija: `net.core.netdev_max_backlog`.

Dakle možemo ju eventualno povećavati sa standardno postavljenih 1.000 na 3.000:

```
sysctl -w net.core.netdev_max_backlog=3000
```



Kod vrlo opterećenih sustava, povećanjem ove vrijednosti, čak do nekoliko tisuća, desetaka tisuća ili više, može se izbjeći gubitak paketa, ali će se zbog toga malo povećati njihovo kašnjenje odnosno latencija.
Na nekim sustavim se to primjerice povećava čak do 30.000 ili čak u ekstremnim slučajevima do 300.000.

## O čemu se ovdje radi?

Zapravo postoje tri reda čekanja povezana s prijemnom stranom uspostavljene TCP veze, na razini *mrežne utičnice* (*socketa*).
Naime, ako je aplikacija blokirana prilikom čitanja kada paket stigne, on će općenito biti poslan u pred red za obradu u korisničkom prostoru rada (engl. *Linux user space*) to jest unutar samog procesa aplikacije.

Dalje postoje dva scenarija:

- Ako se ne može staviti u red čekanja, a mrežna utičnica (*socket*) nije zaključan, bit će postavljen u red čekanja. To je vidljivo s naredbom `netstat` (gledamo stupac `Recv-Q`); primjerice ovako:

```
netstat -tunap
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address        Foreign Address        State       PID/Program
name
tcp        0      0 127.0.0.1:2616       0.0.0.0:*              LISTEN      767/staticd
tcp        0      0 127.0.0.1:2602       0.0.0.0:*              LISTEN      759/ripd
tcp        0      0 127.0.0.1:2601       0.0.0.0:*              LISTEN      743/zebra
tcp        0      0 0.0.0.0:22           0.0.0.0:*              LISTEN      794/sshd
tcp        0      0 192.168.1.129:22     192.168.1.112:13572    ESTABLISHED 2788/sshd:
tcp        0      0 192.168.1.129:22     192.168.1.112:13567    ESTABLISHED 2750/sshd:
tcp        0      0 192.168.1.129:37904  52.166.78.97:443       ESTABLISHED 2828/wget
tcp        0      0 192.168.1.129:56248  52.166.78.97:80        ESTABLISHED 2828/wget
tcp        0      0 192.168.1.129:48764  52.166.78.97:80        TIME_WAIT   -
```

- Ako je mrežna utičnica (tzv. *socket*) zaključana, bit će smještena u red čekanja za zaostalu obradu, kako bi se naknadno obradila. Veličina ovog reda čekanja je definirana u *sysctl* varijabli: `net.core.netdev_max_backlog`.

Odbačeni paketi zbog problema s navedenim redovima čekanja se mogu vidjeti sa naredbom `nstat`, na sljedeći način:

```
nstat -az TcpExtTCPBacklogDrop TcpExtTCPRcvQDrop
```



Za više detalja o ovoj temi pogledajte sljedeća poglavlja:
**25.7.4. *Naredba netstat* → *posebno primjer broj 2. i napredne detalje oko njega*.**
**25.7.5. Naredba *nstat*.**
**25.7.7. Naredba *socket statistics* (*ss*).**

## Kako biti siguran dolazi li do prepunjavanja *backlog* međumemorije?

U datoteci `/proc/net/softnet_stat` se u stvarnom vremenu zapisuju statistike rada (dijela) mrežnog podsustava.
U drugom (**2**) stupcu ove statistike se, svaki puta kada se *backlog* međumemorija prepuni, brojač poveća za jedan.
Svaki redak u ovoj datoteci sa statistikom predstavlja jednu strukturu mrežne statistike, za jednu CPU jezgru.
Sa sljedećom naredbom možemo vidjeti ove statistike, za sve CPU jezgre:

```
cat /proc/net/softnet_stat | awk '{print $2}'
```

Ako su u drugom stupcu sve nule (`00000000`), sve je u redu.

Pogledajmo i kompletnu statistiku za računalo s dvije CPU jezgre:

```
cat /proc/net/softnet_stat

00000c08 00000000 00000002 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000027e7 00000000 00000009 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  . . .
```

**(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) …**

Objasnit ćemo sve stupce u ovoj sveobuhvatnoj statistici (***broj stupaca ovisi o inačici kernela, a mi dajemo opis za v.6.6.28.***):

- **(1)** – Prvi stupac (heksadecimalni broj) [***processed***] označava broj mrežnih okvira („paketa“) koji su procesirani. Optimizira se putem *sysctl* varijable `net.core.netdev_budget`
- **(2)** – Drugi stupac [***dropped***] označava broj mrežnih okvira/paketa koji su odbačeni jer više nema mjesta u ***backlog*** međumemoriji da se pohrane jer je CPU preopterećen i ne stigne ih dohvatiti. Stanje možemo poboljšati, ako povećamo: `net.core.netdev_max_backlog` vrijednost (pogledajte opis gore). Osim ovog ručnog načina pregleda ove statistike, moguće ju je dobiti pokretanjem sljedeće naredbe `nstat`, na sljedeći način:

```
nstat -az TcpExtTCPBacklogDrop
```

- **(3)** – Treći stupac [***time_squeeze***] broj je koliko puta je radnja: (***net_rx_action***) prekinuta jer je njen ili *budget* ili vremenski okvir za obradu istekao. Može se poboljšati povećanjem: `net.core.netdev_budget` i/ili i/ili `net.core.netdev_budget_usecs` i/ili `net.core.dev_weight`.
- **(4-8)** – Ovih pet stupaca je uvijek u stanju nula (`00000000`) odnosno pripremljeno za buduću upotrebu.
- **(9)** – deveti stupac [***received_rps***] je brojač, koliko puta su paket spremljeni za **RPS** funkcionalnost.
- **(10)** – deseti stupac [***flow_limit_count***] je brojač, koliko puta je aktiviran mehanizam ograničenja toka, koji se koristi u ***Receive Packet Steering*** funkcionalnosti.



Vezano za ***Receive Packet Steering*** pogledajte poglavlje:
**25.2.3. Receive packet steering (RPS), Receive flow steering (RFS) i Transmit Packet Steering (XPS).**

- **(12)** - [***input_qlen + process_qlen***] – zbroj duljine ulaznog reda i reda za obradu. Predstavlja ukupan broj paketa koji čekaju obradu.
- **(13)** - [***seq→index***] - prati redoslijed obrade paketa. Važan za TCP protokol koji zahtijeva isporuku paketa u točnom redoslijedu.
- **(14)** - [***input_qlen***] - duljina ulaznog reda. Označava broj paketa koji čekaju na početnu obradu. Optimizira se putem sysctl varijable `net.core.netdev_max_backlog`.
- **(15)** - [***process_qlen***] - duljina reda za obradu. Označava broj paketa koji čekaju detaljniju obradu nakon početnog primitka. Optimizira se putem *sysctl* varijable `net.core.dev_weight`.

Na još višem sloju mreže Linuxa, moguće je ograničiti ili povećati broj dolaznih konekcija na sustav odnosno definirati koliki broj istovremenih konekcija u stanju otvaranja (nakon *TCP SYN* poruke/paketa i slanja *TCP SYN ACK*, a prije zadnjeg *TCP ACK*), dozvoljavamo na sustav. Ova opcija se definira u `net.core.somaxconn`. Njena standardna vrijednost je obično `128`. To se odnosi sâmo na nove paralelne konekcije u kojima sustav iz stanja ***LISTEN*** prelazi u stanje prije ***ESTABLISHED***. Dakle odnosi se na konekcije u trenutku odnosno stanju otvaranja, a ne na ukupan broj svih ostvarenih/otvorenih konekcija na sustav. Navedena opcija je u međuvezi i s TCP opcijom: `net.ipv4.tcp_max_syn_backlog` koju smo spomenuli u poglavlju o TCP protokolu.



Pogledajte i poglavlja:
**10.5.1.3.1. Softverski signali prekida detaljnije.**
**24.2.3. Sigurnosni aspekt uspostavljanja TCP veze (SYN).**

Od dodatnih korisnih opcija, tu je još i opcija definirana u `net.core.optmem_max`. Standardna vrijednost ove varijable je 10.240 bajta. Ona se odnosi na dodatne informacije vezane za mrežne pakete, poput mrežnog sučelja s kojeg su došli, razna zaglavlja, opise grešaka ili slično. Na primjer, kod slanja ili primanja određenih kontrolnih poruka u komunikaciji, a koje sadrže ovakva polja, one se spremaju u ovaj memorijski međuspremnik. Ovaj memorijski međuspremnik je namijenjen za dodatna polja ili zaglavlja mrežnih paketa, kao što su recimo opcije unutar zaglavlja IP paketa (pr. ***IP TTL***). Ova veličina se definira za razinu svakog pojedinog *socketa*. U nekim ekstremnim slučajevima potrebno ju je proširiti. Sljedeće opcije su vezane za kontrolne poruke upozorenja (*warning messages*) koje sustav može slati *kernelu* u *kernel log*. Tako da, ako primjerice povećavamo vrijednost varijable: `net.core.message_cost` s inicijalno postavljene vrijednosti **5**, smanjujemo broj ovakvih poruka koje će se logirati (snimiti) u *kernel log*.

S druge strane varijabla `net.core.message_burst` definira razinu iznad koje će ovakve poruke biti odbačene odnosno neće se zapisivati u *kernel log*. Standardna vrijednost je također **5**, što znači kako je sustav ograničio primanje ovih poruka na maksimalno jednu (1) poruku svakih pet (5) sekundi.



Proučite i poglavlja:
**24.2.4. Trajanje TCP veze.**
**24.2.16. Stanja TCP veze i njena vremenska ograničenja *(timeri)*.**

Pogledajmo koje smo sve *sysctl* varijable odnosno značajke mrežnog stôga ovdje koristili:

| *sysctl* naziv | Kratki opis | Zadana vrijednost |
|---|---|---|
| `net.core.dev_weight` | Broj mrežnih okvira (paketa) koje mrežna kartica može prihvatiti u svoju među memoriju. | `64` |
| `net.core.message_burst` | Razina odbacivanja kontrolnih poruka (`message_cost`). | `10` |
| `net.core.message_cost` | Maks. broj kontrolnih poruka prema kernelu. | `5` |
| `net.core.netdev_max_backlog` | Veličina *backlog* međumemorije. | `1000` |
| `net.core.optmem_max` | Veličina memorije za opcije uz mrežne pakete (*per socket*). | `10240` |
| `net.core.somaxconn` | Maksimalan broj istovremenih konekcija u stanju otvaranja (*TCP SYN*) dozvoljavamo na sustav. | `4096` |
| `net.ipv4.tcp_max_syn_backlog` | Maksimalan broj konekcija za koje još nije zaprimljen *TCP SYN ACK*. | `128` |

**Izvori informacija: (325),(342),(343),(344),(345),(346),(1403),(1404),(1590),(1648),(1649), `man 5 proc`,`man 7 tcp`.**
**Pogledajte i shemu mrežnog stôga Linuxa na plakatu/posteru: P5.**

## 25.1.3.1. NAPI i Pooling

Sada se u analizi (slika 229) postavimo na poziciju između upravljačkog programa i prstenaste međumemorije (*ring buffer*), u slučajevima kada mrežna kartica i njen upravljački program podržavaju takozvanu ***NAPI*** metodu za dohvaćanje paketa.
Tada je moguće ograničiti koliko će maksimalno paketa sustav moći dohvatiti u jednom *NAPI* ciklusu povlačenja (engl. *polling*) paketa, sa svih mrežnih sučelja na sustavu, koje podržavaju *NAPI* metodu dohvaćanja.

Naime s ***NAPI polling*** metodom je moguće ubrzati (ili ponekad usporiti) efektivnu brzinu dohvaćanja te kasnijeg procesiranja paketa. Ipak s *NAPIi polling* načinom rada za mrežna sučelja većih brzina (10+Gbps) i vršnog opterećenja mreže postižu se bolji rezultati u odnosu na klasično dohvaćanje paketa kod kojeg se za primljene pakete generira hardverski signal prekida i pokreće mehanizam dohvaćanja paketa. Naime kada mrežno sučelje zaprima veliki broj paketa, često generiranje signala prekida (IRQ) za svaki dolazni paket može preopteretiti CPU. Ako se koristi *NAPI* kompatibilan upravljački program za mrežno sučelje (karticu), on u tom slučaju automatski prelazi s režima korištenja signala prekida na ***polling*** režim rada, gdje CPU periodično dohvaća pakete u *polling* načinu rada. To smanjuje učestalost signala prekida i omogućuje efikasniju obradu paketa.

Osnovna *sysctl* opcija za NAPI jest opcija: `net.core.netdev_budget` i standardno je postavljena na vrijednost od `300` paketa. Naravno u nekim slučajevima je moguće i potrebno malo povećati ovu vrijednost (uz praćenje statistike gubitka paketa ili njihovog kašnjenja), kako bi se u jednom dohvaćanju moglo dohvatiti više paketa odjednom.
To eventualno uključuje i optimizaciju (povećanje) drugih međumemorija.

Nije loše znati i to da svaki ***NAPI*** ciklus povlačenje paketa (*pooling*) ne smije trajati duže od vremena koje je predviđeno za to. To vrijeme je definirano u mikro sekundama (µs). Ova dodatna ekspertna opcija odnosno njena dostupnost ovisi o tome kako je kernel kompiliran. Navedena *sysctl* opcija se zove: `net.core.netdev_budget_usecs`.



Za više detalja o ***NAPI*** metodi rada mrežne kartice, pogledajte poglavlje:
**10.5.2.1.1. Kako to (NAPI) radi, još malo detaljnije.**

## Busy polling

*Busy polling* je tehnika optimizacije koja omogućava procesoru aktivno čekanje na nove pakete koji dolaze kroz mrežno sučelje, umjesto da se oslanja na signale prekida (IRQ). Klasično, mrežni uređaji obavještavaju CPU o dolasku novih paketa putem signala prekida (IRQ), no to može uvesti latenciju ako CPU ne reagira dovoljno brzo (jer vjerojatno u tom trenutku obrađuje neki drugi zahtjev/program). S *busy pollingom* CPU aktivno provjerava je li paket dostupan na mrežnom sučelju, što smanjuje latenciju jer procesor ne mora čekati signale prekida za zaprimljene pakete. Ovakav način rada opterećuje CPU, jer svako malo (u definiranim intervalima) provjerava jesu li se na mrežnom sučelju pojavili novi paketi.

**U virtualiziranim okruženjima** (npr. s **VirtIO** mrežnim sučeljima), omogućavanje *busy polling*-a može poboljšati brzinu dohvaćanja mrežnih paketa, smanjujući kašnjenje u mrežnoj komunikaciji. *VirtIO*, koji je optimiziran za virtualizaciju, koristi ovu tehniku za ubrzanje prijenosa podataka između VM-ova i fizičkog računala, naravno uz nešto veće opterećenje CPU-a.
Kod *busy pollinga* imamo dvije slične *sysctl* opcije: `net.core.busy_poll` se izričito koristi za implementaciju *busy polling-a*, gdje CPU konstantno provjerava ima li novih paketa bez čekanja na signal prekida (IRQ). S druge strane postoji `net.core.busy_read` koji nije tehnički oblik *busy polling* jer nije vezan za čekanje na pakete, već za aktivnu provjeru i učestalost čitanja paketa s mrežnog sučelja.

Pogledajmo koje smo sve *sysctl* varijable odnosno značajke vezane za ***NAPI*** i ***polling*** ovdje koristili:

| ***sysctl* naziv** | **Kratki opis** | **Zadana vrijednost** |
|---|---|---|
| `net.core.netdev_budget` | Maksimalan broj paketa koji se ***NAPI*** metodom mogu odjednom dohvatiti. | `300` |
| `net.core.netdev_budget_usecs` | Vrijeme dohvaćanja (*pooling*) unutar ***NAPI*** ciklusa.<br>Definira se u mikro sekundama (μs). | `2000` |
| `net.core.busy_poll` | Kada je aktivirana (>0) *busy polling* znači da kernel aktivno provjerava mrežno sučelje u petlji (svakih ***X*** mikro sekundi definiranih ovdje) bez čekanja na signale prekida (IRQ) od mrežnog sučelja.<br>Dakle ovdje se definira interval između dva *polling* pokušaja.<br>Preporučena vrijednost ovisi o broju socketa:<br>→ Za nekolicinu ~**50.**<br>→ Za nekoliko stotina ~**100**. | `0` |
| `net.core.busy_read` | Ova varijabla regulira koliko često procesor aktivno pokušava dohvatiti pakete s mrežnog sučelja (u mikro sekundama).<br>Ona (>0) povećava učestalost čitanja paketa (ponovno aktivnim čitanjem podataka), bez izravnog čekanja na "*polling*". To omogućuje bržu obradu paketa, ali ne nužno u obliku *busy polling*-a. | `0` |

Da bi `net.core.busy_poll` i `net.core.busy_` radili na Linux kernelu, potrebna je podrška za opciju naziva: `CONFIG_NET_RX_BUSY_POLL` koja se omogućava prilikom kompiliranja kernela. To možemo provjeriti sa:

```
grep CONFIG_NET_RX_BUSY_POLL /boot/config-$(uname -r)
```

Ako mrežni upravljački program (*kernel modul*) nije ***NAPI*** kompatibilan, korištenje *busy polling-a* kroz postavke poput `net.core.busy_poll` i `net.core.busy_read` može biti ograničeno.
Srećom, postoje određene kernel postavke koje mogu poboljšati performanse mreže bez NAPI-a.
Nekoliko važnih postavki koje možete razmotriti su:

- Postavke za ***interrupt coalescing*** – ako mrežna kartica podržava *interrupt coalescing* (grupiranje signala prekida), možete podesiti parametre za smanjenje broja signala prekida.



Za više detalja o ***interrupt coalescing*** (***interrupt moderation***) pogledajte:
**10.5.2.2. *Interrupt moderation*.**

- ***Interrupt Affinity*** – raspodjela signala prekida na više CPU jezgri (tzv. ***IRQ affinity***) može pomoći da se CPU opterećenje bolje balansira.



Za više detalja o ***interrupt affinity*** pogledajte:
**10.5.1.1. IRQ i Linux.**

- Optimizacija ***TX/RX prstenast međumemorije*** – moguće je povećati veličinu niza za prijenos (TX) i prijem (RX).



Za više detalja o ***TX/RX prstenastoj međumemoriji*** pogledajte:
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

- Povećanje drugih međuspremnika u mrežnom stôgu Linuxa.



Za više detalja pogledajte:
**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**

**Izvori informacija: (325),(342),(343),(344),(345),(346),(1403),(1404),(1590),(1648),(1649),(1766), `man 5 proc`,`man 7 tcp`. Pogledajte i shemu mrežnog stôga Linuxa na plakatu/posteru: P5.**

## 25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa

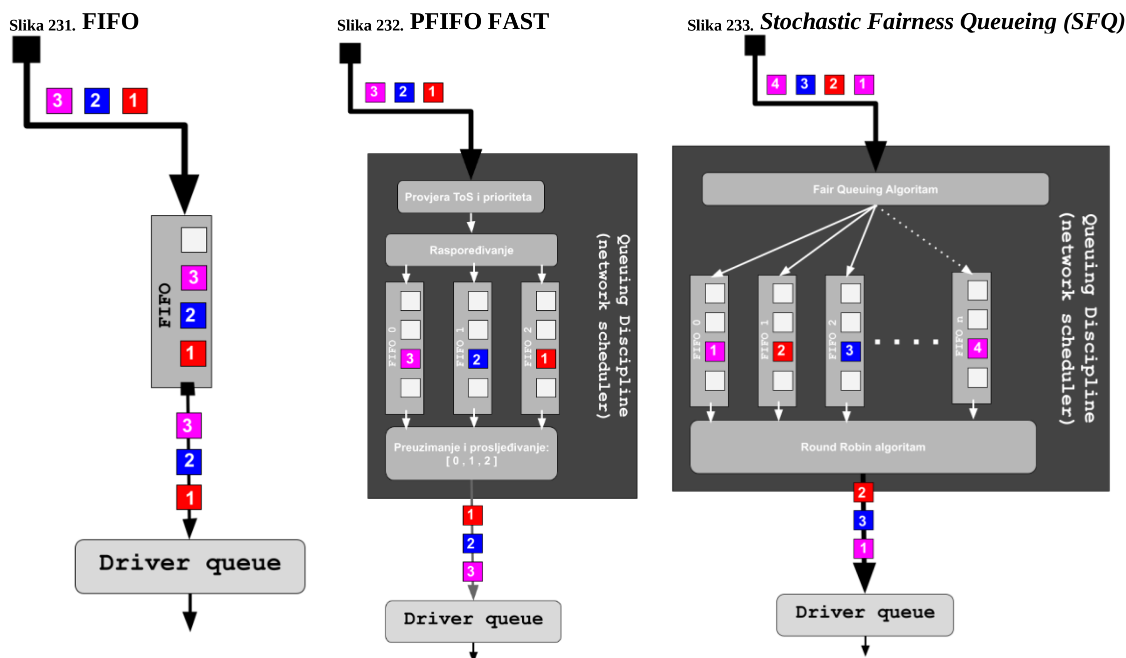
Sada ćemo se vratiti na sloj na kojem se nalaze mehanizmi za prihvaćanje, raspodjelu i prosljeđivanje mrežnih paketa.
Ovaj mehanizam znan je i kao ***network scheduler***, ***packet scheduler***, ***queueing discipline***, ***qdisc*** ili ***queueing*** algoritam.
Iako ova komponenta Linux sustava nije direktno vezana za dio priče o transportnim protokolima i njihovim mehanizmima, važno je razumjeti cijeli put, koji prolazi svaki mrežni paket (odnosno podaci) prije nego li dođe do fizičke mrežne kartice.
Dakle *network scheduler* upravlja nizom mrežnih paketa za slanje (***TX***), stavljajući ih u redove odnosno nîzove, obrađujući ih te prosljeđujući dalje prema nižim slojevima mreže te međumemoriji koja se nalazi u upravljačkom programu mrežnog sučelja (tzv. *driver queue*). Ovu komponentu možemo nazvati i *mrežnim raspoređivačem* paketa na najnižoj razini, čijih varijanti postoji poveći broj. Naime svaka od njegovih varijanti implementira drugačiji algoritam za raspodjelu mrežnog prometa (paketa).
Sama logika ovog *mrežnog raspoređivača* odlučuje koji mrežni paket proslijediti sljedeći u nîzu. Ova logika je povezana sa sustavom čekanja, privremeno spremajući mrežne pakete dok ih se ne prenese prema mrežnoj kartici (sučelju). Dodatno, neki sustavi mogu imati jedan ili više nîzova od kojih je moguće da svaki može pohraniti pakete jednog toka, određene klasifikacije ili prioriteta. Pri svemu tome sâmo jedan od ***qdisc*** mehanizama (algoritama) u pravilu može biti aktivan za pojedino mrežno sučelje odnosno mrežnu karticu.

*Važno je razumjeti da promjenom algoritma za raspodjelu mrežnih paketa (**qdisc**) možemo poprilično utjecati na propusnost ili kašnjenje odnosno latenciju[738], a pogotovo u kombinaciji s TCP algoritmima za nadzor zagušenja.*

Kako smo već rekli, ovih algoritama dostupan je veći broj, a mi ćemo spomenuti sâmo neke koje možemo rasporediti u dvije vršne kategorije:

- *Bez klasne (engl. Classless)*:
  - ***Pfifo_fast*** (engl. *Packet First In First Out (PFIFO)*) – uvodi najmanje kašnjenje jer je najjednostavniji.
  - ***Token Bucket Filter (TBF)*** – ima mogućnost ograničenja propusnosti i brzine rafalnog slanja paketa u nîzu.
  - ***Stochastic Fairness Queueing (SFQ) –*** ima relativno malo opterećenje CPU-a, iako pokušava ravnomjerno raspoređivati prijenos paketa između više nîzova (*queues*). To postiže korištenjem *hash* funkcije za odvajanje prometa u zasebne *FIFO* nîzove, koji se spajaju na izlazu, pomoću *round-robin* algoritma. ***(INFO)***
  - ***CoDel*** i ***Fair Queueing CoDel*** – imaju mogućnost uvođenja kašnjenja isporuke paketa.
- *Klasne (engl. Classful)* koji se koriste, ako imamo različite vrste prometa koje traže različiti tretman odnosno obradu ili klasifikaciju (njih nećemo objašnjavati zbog njihove kompleksnosti). Neki od njih su:
  - ***Hierarchical Token Bucket (HTB).***
  - ***Class Based Queueing (CBQ).***

Pogledajmo kako rade neki od navedenih algoritama znanih kao *network scheduleri* odnosno *network queuing* mehanizmi.



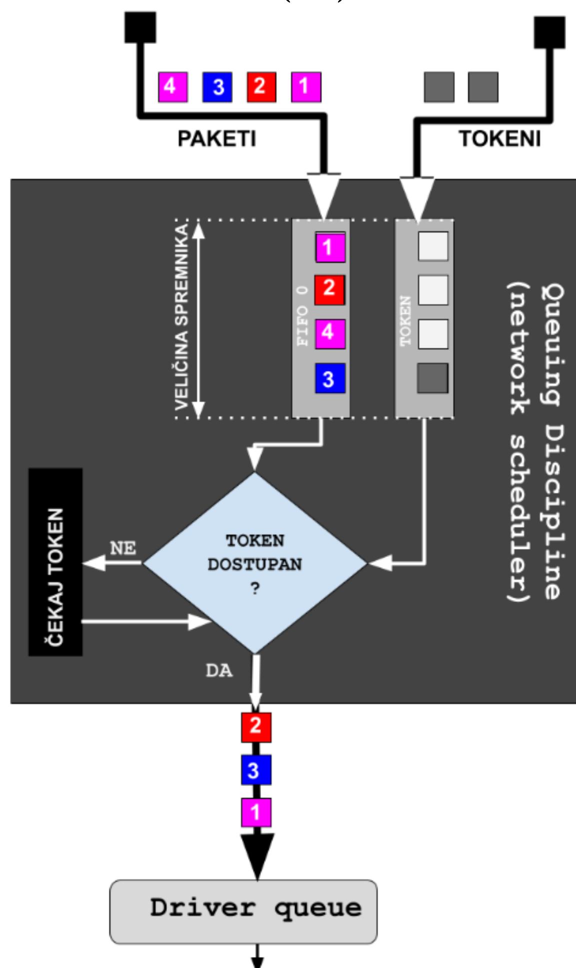

Slika 231. FIFO
Slika 232. PFIFO FAST
Slika 233. *Stochastic Fairness Queueing (SFQ)*

Prvo ćemo se upoznati s najosnovnijim mehanizmom koji se zove **FIFO** (*First-In First-Out*) (slika 231.).
On ne radi nikakve provjere paketa, prema prioritetu, niti radi bilo kakva preslagivanja paketa po bilo kojoj dodatnoj logici. Njegov rad se svodi na primanje paketa, upravo istim redoslijedom kojim dolaze, te popunjavanjem svog memorijskog spremnika (*buffera*), istim tim redoslijedom. Na kraju paketi izlaze istim redoslijedom kojim su i došli.

**Pfifo_fast** algoritam (slika 232.) za pakete koji su u dolazu, provjerava prioritet paketa (iz IP zaglavlja [***ToS*** odnosno ***DSCP*** polje]) te se ovisno o njemu oni prosljeđuju u pojasne nîzove (*FIFO 0, 1 ili 2*). Oni s najvećim prioritetom se prosljeđuju u *FIFO 0*, oni s malo manjim u *FIFO 1*, a oni s najmanjim prioritetom u *FIFO 2*. Na izlazu se prvo preuzimaju **svi** iz *FIFO 0* i prosljeđuju, potom **svi** iz *FIFO 1*, a tek na kraju oni iz *FIFO 2*. Ovime se postiglo prioritetno slanje onih paketa koji su označeni s većim prioritetom (ako uopće imaju ovu oznaku) u odnosu na one s manjim prioritetom.

***SFQ*** algoritam (slika 233.) zaprima sve pakete te ih provlači kroz svoj algoritam za "*pravednu*" raspodjelu, pa ih ravnomjerno raspodjeljuje u veći broj nîzova (*FIFO*). Separacija paketa u isti logički nîz (tzv. pojas) se postiže s *hash* funkcijom svakog paketa, koja se prvo izračunava (odrađuje), prije nego se paket ubaci u određeni nîz.
Na kraju se pomoću klasičnog *round robin* algoritma uzimaju paketi iz svih nîzova, redom: *FIFO 0,1,2,3,4,… n*, pa cijeli proces kreće iz početka. Zbog mogućnosti da *hash* funkcija koja je osnovni mehanizam za raspodjelu paketa u određeni nîz, nije konstantno najbolji odnosno najravnomjerniji odabir, ona se periodički, stalno mijenja, pomoću metode *perturbacije*.

Slika 234. **Token Bucket Filter (TBF)**




**Token Bucket Filter (TBF)**
***TBF*** metoda radi pomoću *tokena (oznake/znaka)* i spremnika (*bucket*). Ovdje se radi ograničenje prometa za određeno mrežno sučelje, čekanjem na takozvane *tokene*, s kojima se potvrđuje jedan blok paketa, koji je veličine koja je definirana trenutnom veličinom spremnika.

Dakle ako postoji dovoljan broj *tokena* u spremniku, svi paketi u *FIFO* nîzu će biti poslani. Kašnjenje se simulira s nedostatkom *tokena* u cijelom spremniku te čekanjem da se napune, kako bi se onda paketi u *FIFO* nîzu mogli poslati dalje.
Pogledajmo i logičku shemu ovakvog rada na slici **234.**



Vezano za klasifikaciju prometa te **QoS**, odnosno **ToS** i **DSCP** zaglavlja u **IP** protokolu, pogledajte poglavlje:

**23.1.1. Osiguranje kvalitete (*QoS*) i klasifikacija prometa (*ToS i DSCP*).**

Ako pogledamo takozvani ***Driver queue*** odnosno *red čekanja* unutar upravljačkog programa, koji je na svim shemama (slike 231-234) vidljiv kao najdonja komponenta, on je jednostavan i radi kao **FIFO** (*First In First Out*). Ovaj memorijski spremnik odnosno red čekanja paketa jednako tretira sve pakete i nema sposobnost razlikovanja paketa različitih tokova.

Ovaj dizajn čuva upravljački program mrežne kartice jednostavnim i brzim.
Imajte na umu kako naprednije *ethernet* mrežne kartice i većina bežičnih mreža podržavaju više neovisnih tokova prijenosa, ali je ipak svaki od tih redova obično *FIFO* jer su sâmo viši slojevi odgovorni za odabir reda za prijenos koji će se koristiti, u kojem se nalaze dodatna logika i algoritmi njihovog rada.



Za više informacija o *network schedulerima*, pogledajte poveznicu: https://lartc.org/lartc.html#LARTC.QDISC

**Sumirajmo:**
Komponenta koja se nalazi između IP sloja (*IP stack*) unutar mrežnog stoga Linuxa i reda čekanja upravljačkog programa (*driver queue*) odnosno njegove međumemorije, sloj je u kojem je mehanizam za raspodjelu mrežnih paketa znan i kao navedeni *Network queuing/qdics* (vidi sliku 235 dolje). Ovaj sloj uvodi mogućnost upravljanja mrežnim prometom Linux kernela, a koji uključuje:

- **Klasifikaciju i prioritizaciju prometa (**takozvani **QoS** to jest *Quality of Service* odnosno osiguranje kvalitete**).**
- **Reguliranje brzine protoka** (takozvani *rate shaping*).

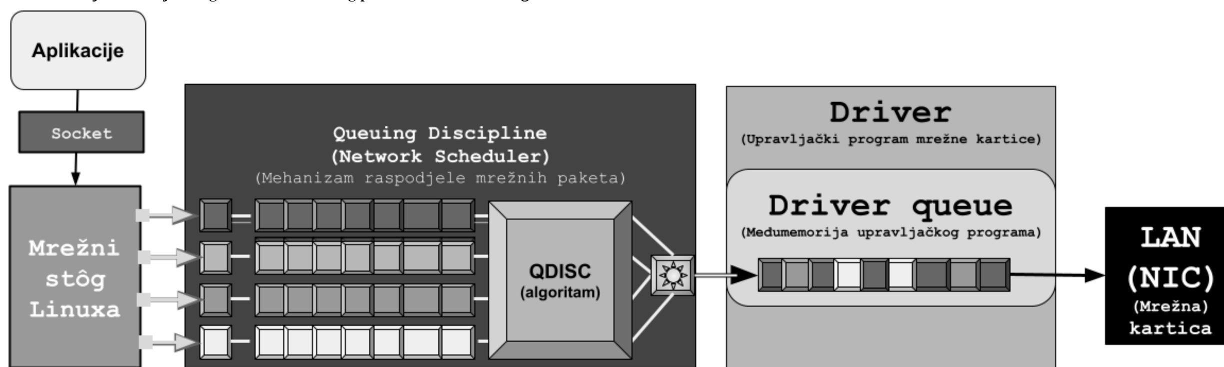
Pri tome, postoje tri ključna pojma odnosno tri dodatna sloja na ovoj razini (dakle na razini *network schedulera*):

- ***Qdisc*** - odnosno sâm algoritam za raspodjelu mrežnih paketa.
- ***Klase*** - pojedini ***qdisc*** algoritmi (*classfull*) mogu implementirati klase pomoću kojih se pojedini nizovi paketa mogu drugačije obrađivati i u konačnici prosljeđivati nižim slojevima.
- ***Filteri*** - odnosno *klasifikatori* (odnose se na ***klase***) su mehanizmi koji se koriste za klasifikaciju paketa unutar pojedine ***klase.***

***Za više detalja, pogledajte sljedeće poglavlje!***

Pogledajmo pojednostavljenu logičku shemu mrežnog podsustava u Linuxu, od IP sloja na niže, s pogledom na mehanizme za raspodjelu mrežnih paketa, vidljivog na slici 235.

**Slika 235. Pojednostavljena logička shema mrežnog podsustava Linuxa s naglaskom na *network scheduler*.**






U poglavlju: **24.2.9. Nadzor zagušenja (*Congestion control*)**, objasnili smo i testirali utjecaj promjene algoritama koji se odnose na **TCP** protokol, i to dio za nadzor zagušenja, koji na TCP razini utječu na konačnu propusnost mreže.

Vratimo se na ovu (nižu) razinu na kojoj imamo mogućnost promjene algoritama odnosno mehanizama za prihvaćanje, klasifikaciju (**QoS**) i prosljeđivanje mrežnih paketa, koja se naziva: *network scheduler*, *packet scheduler*, *queueing discipline* ili ***qdisc*** vidljivo na slici 230 i 235 kao „*Network scheduler*“.
U Linuxu se odabrani globalni ***qdisc*** mehanizam definira u *sysctl* varijabli: `net.core.default_qdisc` odnosno vidljiv je i u */proc* sustavu kao: `/proc/sys/net/core/default_qdisc`.

Prvo pogledajmo koji od njih imamo postavljen kao zadani, odnosno koji je trenutno u upotrebi za sva mrežna sučelja:

```
sysctl net.core.default_qdisc
net.core.default_qdisc = pfifo_fast
```

Vidimo kako imamo postavljen `pfifo_fast` algoritam, koji je u većini slučajeva zadovoljavajući.

Sada moramo provjeriti koje od *qdisc schedulera* uopće imamo kompilirano u našem kernelu (nîz naredbi koji slijedi je u jednom retku):

```
grep '^CONFIG_NET_SCH_' /boot/config-$(uname -r)

CONFIG_NET_SCH_CBQ=m
CONFIG_NET_SCH_HTB=m
CONFIG_NET_SCH_HFSC=m
CONFIG_NET_SCH_PRIO=m
CONFIG_NET_SCH_MULTIQ=m
CONFIG_NET_SCH_RED=m
CONFIG_NET_SCH_SFB=m
CONFIG_NET_SCH_SFQ=m
CONFIG_NET_SCH_TBF=m
CONFIG_NET_SCH_NETEM=m
CONFIG_NET_SCH_MQPRIO=m
CONFIG_NET_SCH_CHOKE=m
CONFIG_NET_SCH_CODEL=m
CONFIG_NET_SCH_FQ_CODEL=m
CONFIG_NET_SCH_FQ=m
CONFIG_NET_SCH_FIFO=y
```

Vidimo kako je ***FIFO*** jedini ugrađen u kernel (`=y`) te vidimo i cijeli nîz drugih dostupnih algoritama, kao kernel modula (`=m`).

**Promjenu ovog algoritma možemo postići na dva načina:**

**1.** Mi ćemo ovdje za primjer promijeniti *qdisc scheduler* u **FQ** (fair *queuing*) koji je dobar izbor za visoko performantne mreže (10Gbps+) i to za sva mrežna sučelja odjednom. Dakle pokrenimo sljedeću naredbu:

```
sysctl -w net.core.default_qdisc=fq
```

Ili ako želimo trajnu promjenu odnosno aktivaciju ovog algoritma (***fq***), možemo u datoteku `/etc/sysctl.conf` dodati:

```
net.core.default_qdisc = fq
```

Nakon toga i nakon restarta računala sva mrežna sučelja će imati postavljen ovaj novi (**FQ** (fair *queuing*)) algoritam.

***Scheduleri (QDISC) imaju i svoju međumemoriju čija veličina se prema potrebi može mijenjati.***
***(Oprez – povećanje ove međumemorije u pravilu nije potrebno)!***

Primjerice možemo ju povećati sa zadanih 32MB na 64MB:

```
tc qdisc change dev eno1 root fq memory_limit 67108864
```

**2.** Nastavno na prvi primjer, možemo napraviti i ručne promjene na željenoj mrežnoj kartici bez trajnih promjena. Primjerice za `eno1` u našem slučaju.
Za ovu potrebu koristiti ćemo naredbu `tc` (*traffic control*), ali prethodno učitajmo i potrebni kernel modul za ovaj rad.

```
modprobe sch_fq
tc qdisc add dev eno1 root fq
```

Bez obzira koristili prvu (**1.**) ili drugu (**2.**) metodu, vidjet ćemo da se *kernel modul* za ***FQ*** (modul: `sch_fq`) učitao.
To možemo provjeriti sa sljedećom naredbom:

```
lsmod | grep -i sch
sch_fq                 17156  9
```

Kernel moduli za *qdisc* (*scheduling)* imaju u nazivu ***sch_****imeschedulera*, a možemo ih sve (kao datoteke) izlistati s naredbom:

```
ls -a /lib/modules/$(uname -r)/kernel/net/sched/sch_*
```

I tek sada možemo provjeriti je li se promijenio *qdisc* (*scheduler*) s naredbom `tc` (skratili smo ispis):

```
tc -s qdisc show dev eno1
qdisc fq 8001: dev eno1 root refcnt 92 limit 10000p flow_limit 100p buckets 1024
```

Vidimo kako je sada odabran `fq`.
Isto smo mogli vidjeti i s naredbom `ip`, na sljedeći način:

```
ip addr show eno1

2: eno1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc fq state UP group
default qlen 1000 link/ether d8:9d:67:62:1d:e0 brd ff:ff:ff:ff:ff:ff
```

Dakle vidimo kako je odabran `fq` *scheduler*.



Postoji i poseban slučaj u kojem se koristi „*Multi Queue*" mehanizam, kod sustava koji imaju mrežne kartice koje hardverski podržavaju primjerice više **RX** nîzova [**25.2.1.**]. Tada se standardno ispred normalnog *schedulera* postavlja **`mq`** pseudo *scheduler* koji distribuira pakete u više nîzova ispod njega, gdje se normalno koriste standardno dostupni *scheduleri*, poput **`pfifo_fast`** ili nekog drugog, a koji se primjenjuju na svako ***RSS*** hardversko sučelje odnosno *RSS* kanal (*RX* nîz).
**Isti mehanizam se može koristiti i kod mrežnih kartica koje imaju hardversku podršku za TX nîzove (TX queue).**

Za `eno1` mrežnu karticu koja podržava hardverski do 24 **RX** nîza (ispis smo skratili), će to onda izgledati ovako:

```
tc qdisc show dev eno1
qdisc mq 0: root
qdisc pfifo_fast 0: parent :18 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :17 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

**Izvori informacija: (320),(321),(321.1),(321.2),(322),(323),(324),(325),(341),(734),(735),(736),(737),(922),(923), `man tc`, `man modprobe`, `man lsmod`, `man sysctl`, `man 5 proc`. Za mrežni stôg Linuxa pogledajte plakat P5.**

### 25.1.4.1. Bufferbloat

*Bufferbloat* je uzrok velike latencije u mrežama uzrokovane prevelikim međuspremnikom paketa. *Bufferbloat* također može uzrokovati varijacije u kašnjenju paketa (poznate kao ***jitter***), ali i smanjiti ukupnu propusnost mreže. Naime kada su usmjerivač ili preklopnik konfigurirani za upotrebu pretjerano velikih međuspremnika, čak i vrlo velike brzine mreže mogu postati praktički neupotrebljive za mnoge interaktivne protokole poput ***VoIP*** kao i vrlo malih poruka *poput TCP: SYN ili ACK* te protokola poput ***DNS***, ***SSH***, ***ping*** kao i mrežnih igara, čak i za uobičajeno surfanje webom. Sve zbog prevelike latencije odnosno kašnjenja. Naime neki proizvođači mrežne opreme koriste nepotrebno velike međuspremnike u nekim svojim uređajima, kao i mehanizme koji navedeni efekt dodatno povećavaju. U takvoj opremi, ***bufferbloat*** se događa u trenutku kada mrežna veza postane zagušena, što dovodi do toga da se paketi na duže vrijeme stavljaju u red čekanja (*queue*) u (tim) prevelikim međuspremnicima.



Pogledajte i druga povezana poglavlja:
**24.2.9. Nadzor zagušenja (*Congestion control*).**
**24.2.14. Primitak paketa izvan redoslijeda slanja (*Out-Of-Order*).**

Primjerice u ***FIFO*** redu čekanja, pretjerano veliki međuspremnici rezultiraju duljim redovima čekanja i većim kašnjenjem te ne poboljšavaju propusnost mreže. Stoga je za uklanjanje ovog problema, često potrebno i/ili promijeniti mehanizam (algoritam) reda čekanja i/ili smanjiti veličinu međumemorije. Naime čak i primjerice povećanjem veličine ***txqueuelen*** međumemorije (**poglavlje: 25.1.2.**) i drugih međumemorija mrežnog sučelja uvodimo određeno kašnjenje zbog nakupljanja paketa, te posljedičnog kašnjenja njihove dalje obrade. No vratimo se na najčešći uzrok, a to je algoritam koji u najvećoj mjeri uzrokuje ovakve probleme s uvođenjem dodatnog kašnjenja. Preporuka je koristiti moderne algoritme poput: **`fq_codel`**, **`cake`** i drugih koji pripadaju u takozvanu: *Smart queue management* (***SQM***) kategoriju. Ovakvi algoritmi stavljaju promet svakog tôka u svoj red čekanja. Tôk se odnosi na promet s jedne IP adrese/porta na drugu IP adresu/port. Tada ***qdisc*** mehanizam osigurava da niti jedan red čekanja ne bude predug. Kako bi to učinio, on pregledava sve redove čekanja i preferira odabir slanja paketa iz tokova koji nemaju mali red čekanja. Ako se red čekanja za određeni tôk poveća, ovaj mehanizam može promet unutar jednog tôka označiti ***ECN*** porukom (poglavlje: **24.2.9.3.**) ili ispustiti određeni postotak tih paketa kako bi se izbjeglo zagušenje (za taj tôk odnosno nîz podataka).
Test svog usmjerivača na ***bufferbloat*** možete napraviti primjerice na stranici: **http://www.dslreports.com/speedtest**.

**Izvori informacija: (322),(324),(325),(341),(734),(737),(922),(923), Za mrežni stôg Linuxa pogledajte plakat P5.**

### 25.1.4.2. Oblikovanje mrežnog prometa i naredba `tc`

**Slijedi napredno poglavlje!**
Svi moderni Linux kerneli imaju ugrađenu kontrolu prometa (engl. *traffic control*) i značajke oblikovanja prometa (engl. *traffic shaping*). Unutar softverskog paketa `iproute` **(**ili `iproute-tc`**)** nalazi se i naredba `tc` za kontrolu ovih značajki iz naredbenog retka. Stoga instalirajmo ove softverske pakete (ako ih slučajno nemamo):

```
yum -y install iproute-tc iproute
```

Dolazni i odlazni mrežni paketi stavljaju se u red čekanja (engl. *queue*) prije nego što budu primljeni odnosno odaslani.
Red čekanja za dolazne pakete poznat je kao ulazni red (engl. *incoming/ingress queue*). Slično tome, red čekanja za odlazne pakete naziva se izlazni red (engl. *outgoing/egress queue*). Mehanizam za prihvaćanje, klasifikaciju i prosljeđivanje (raspodjelu) mrežnih paketa, naziva se i *network scheduler*, *packet scheduler*, *queueing discipline* ili ***qdisc***. Međutim ***qdiscs*** se može koristiti i za oblikovanje prometa korištenjem različitih algoritama čekanja i raspodjele mrežnog prometa. Jasno je da imamo veću kontrolu nad izlaznim redom čekanja (nîzom/***qlen***) jer on sadrži pakete koje je generiralo naše računalo (poslužitelj).
Moguće je promijeniti redoslijed ovih paketa u redu čekanja, efektivno favorizirajući neke pakete u odnosu na ostale.
Primjerice naredba `ip -s link` prikazuje nam kapacitet reda (***qlen***) iskazan u ukupnom broju paketa. Ako je red čekanja pun, a do mrežnog sučelja i samim time reda čekanja dođe više paketa; oni se odbacuju i ne obrađuju i ne prenose dalje.
Ulazni red čekanja sadrži pakete koje su nam poslala druga računala na mreži. Stoga ih ne možemo mijenjati; jedino što možemo učiniti jest eventualno ispustiti neke pakete, ukazujući na zagušenje mreže tako što tada ne šaljemo TCP ACK računalu koje ih šalje. Ovo računalo (ili uređaj) koji nam šalje pakete dobiva našu *poruku* i usporava prijenos paketa prema nama. Međutim za UDP pakete ovakav mehanizam nije moguć jer, ako se UDP paketi odbace, jednostavno se izgube jer nema potvrdne poruke (ACK) i ponovnog prijenosa. S obzirom na to da se može oblikovati samo izlazni promet (jer ne možete kontrolirati kada će vam druga strana poslati pakete), većina funkcionalnosti i dokumentacije *qdiscs*-a koncentrirana je oko izlaznih *qdisc*-ova.
Izlazni *qdisc* može se konfigurirati u hijerarhijskom stablu počevši od korijenskog *qdisc*, dok postoji samo jedan ulazni *qdisc*. Ovo spominjemo samo zato što je, kao što ćete vidjeti, sintaksa za stvaranje ulaznog i izlaznog *qdisc*-a malo drugačija.

Sada ćemo se detaljnije upoznati s nekoliko osnovnih elemenata koji se nalaze u sustavu oblikovanja mrežnih paketa:

- ***Shaping*** odnosno oblikovanje uključuje odgodu prijenosa paketa kako bi se zadovoljila određena brzina prijenosa podataka. Ovo je način na koji osiguravamo da brzina izlaznih podataka ne premaši željenu vrijednost. Mehanizmi za oblikovanje također mogu ublažiti zagušenje prometa. Oblikovanje se vrši na izlazu (***egress*** smjer).
- ***Scheduling*** odnosno raspodjeljivanjem se odlučuje koji će se paket sljedeći poslati. To se postiže preuređivanjem paketa u redu čekanja. Ciljevi su pružiti brzi odgovor za interaktivne aplikacije i također osigurati odgovarajuću propusnost za skupne prijenose poput preuzimanja pokrenutih od strane udaljenih računala. Raspodjeljivanje se vrši na izlazu (***ingress*** smjer).
- ***Policing*** odnosno nadzor je zadužen za mjerenje paketa primljenih na sučelju i njihovo ograničavanje na određenu vrijednost. Paketi bi mogli biti ponovno klasificirani ili odbačeni. Nadzor se obavlja na ulazu (***ingress*** smjer).
- ***Dropping*** odnosno odbacivanje se može dogoditi nakon što promet premaši unaprijed definiranu vrijednost, a tada se paketi jednostavno odbacuju. To se radi i na ulazu i na izlazu.

Komponente ovog sustava su slijedeće:

- **Qdiscs** - je skraćenica za *Queue Discipline*. *Qdisc* je mehanizam za raspoređivanje paketa koji je dodijeljen pojedinom mrežnom sučelju. *Qdiscs* nije jedan već ih je dostupno više vrsta, a oni su implementirani kao kernel moduli, *Qdisc* može ispustiti, proslijediti, staviti u red čekanja, odgoditi ili promijeniti redoslijed paketa na mrežnom sučelju. Navedena naredba `tc` koristi se za upravljanje s ***qdiscs*** na razini mrežnih sučelja. Ostali izrazi koji se koriste za ***qdisc*** su ***network scheduler***, ***packet scheduler*** ili ***queueing discipline***. U konačnici kernel šalje pakete primljene na mrežnom sučelju na ***qdisc***. Slično, kernel dohvaća (vadi iz reda čekanja) pakete s ***qdisc***, za prijenos prema mrežnom sučelju. ***Qdisc-a*** postoje dvije vrste, klasni koji sadrže klase, i *qdiscs* bez klasa.
- ***Klase*** – one čine pod mehanizam ***qdisca***. Klasa može sadržavati drugu klasu. Pomoću klasa možemo detaljnije konfigurirati ***QoS*** sustav (sustav osiguranja kvalitete usluge). Kada ***qdisc*** primi pakete, oni mogu biti stavljeni u red čekanja u unutarnjim *qdisc-ovima* unutar klasa. Kada kernel želi pakete za prijenos, paketi određenih klasa mogu se stavljati ispred drugih, dajući tako prioritet određenim vrstama prometa.
- ***Filteri*** - kada ***qdisc*** s definiranim klasama primi paket, treba odlučiti u kojoj klasi mora biti stavljen u red čekanja. Dakle treba ga klasificirati. Filteri se koriste za klasifikaciju paketa. Filter mora sadržavati naziv klasifikatora. Najčešći klasifikator koji koriste filtri je ***u32*** klasifikator. On se najčešće koristi za odabir paketa na temelju atributa paketa.

U svakom slučaju moramo koristiti klasifikator (**filter**) na odgovarajući *qdisc* mehanizam. Jednostavno rečeno, ovaj *klasifikator* (**`filter`** kako ga `tc` naziva) dohvaća pakete prema željenim kriterijima i, ako podudaranje uspije, izvodi određene radnje na njima. Što se klasifikacija pakete tiče, moguće su razne opcije. Pogledajmo neke od njih:

- `fw` – opcija temelji odluku na tome kako je vatrozid označio paket.
- `u32` – ovo je već viđena opcija, koja temelji odluke na osnovu polja unutar paketa (pr. izvorišna IP adresa).
- `route` – opcija koja temelji odluku prema ruti prema kojoj će paket biti preusmjeren.
- `rsvp` i `rsvp6` – opcija koja radi na mehanizmu za rezervaciju propusnosti mreže (***RSVP***), kao i mnoge druge.

Pri tome ***klasifikatori*** prihvaćaju nekoliko argumenata:

- `protocol` – definicija protokola za klasifikator. Ovo je u pravilu IP protokol (`ip`). Ovaj argument je obvezan.
- `parent` – definicija mrežnog sučelja na koje se aplicira. Ovaj argument je obvezan.
- `prio` – definira prioritet klasifikatora; niža vrijednost se prva obrađuje.
- `handle` – identifikator unutar strukture klasifikatora (koristi se i kao identifikator odredišta).

Upotrebom naredbe `tc` moguće je raditi promjene na odabiru *qdiscs* mehanizma i njegovim opcijama, *klasama* i *filterima*. Pogledajmo jedno mrežno sučelje (**`eth0`** mrežnu karticu) na Linuxu:

```
tc -s qdisc show dev eth0
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 3728071 bytes 59911 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Vidimo da je zadani ***qdisc*** `pfifo_fast`, koji je standardno postavljen na većini sustava. On je zapravo FIFO *qdisc* s prioritizacijom. On ima tri pojasa, FIFO 0, FIFO 1 i FIFO 2. Veličina spremnika svakog pojasa odgovara veličini zadane ***txqueuelen*** međumemorije mrežnog sučelja. Pojas 0 je za promet iz interaktivnih aplikacija, gdje želimo minimizirati kašnjenje. Pojas 1 je za "normalnu" komunikaciju. Dok je pojas 2 za preostale prijenose podataka. Paketi se stavljaju u jedan od tri pojasa na temelju vrijednosti ***ToS (DSCP)*** polja unutar **IP** zaglavlja u paketu. Prvo se šalju svi paketi u FIFO 0. Kada nema preostalih paketa u FIFO 0, prenose se paketi u FIFO 1. Na kraju se prenose paketi u FIFO 2.

Ipak, na novijim kernelima češće se koristi `fq_codel`. On je specifičan po tome što održava pošteniji red čekanja tako što ima niz FIFO redova čekanja te koristi *hash* funkciju za slanje dolaznih paketa u jedan od redova čekanja. Svaki od ovih redova nadzire ***CoDel*** mehanizam čekanja. *CoDel* pokušava kontrolirati kašnjenje paketa na određenu vrijednost, recimo na 5 mili sekundi (ms) prema zadanim postavkama. Ispituje vrh svakog reda čekanja i ispušta pakete koji su tamo već dugo. Zbog toga, s odbačenim paketima i kontroliranim kašnjenjem, mehanizmi kontrole zagušenja TCP protokola mogu uredno obaviti svoj dio posla. Pogledajmo kako promijeniti *qdisc* algoritam u `fq_codel`, za mrežno sučelje **`eth0`**:

```
tc qdisc add dev eth0 root fq_codel
```

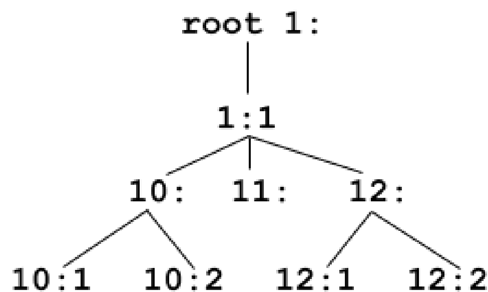
***Handle***

Svi *qdisc*-ovi i klase imaju individualni identifikator (ID) koji ima format **m:n**, gdje je **m** glavni broj, a **n** pomoćni broj. I **m** i **n** su ograničeni na 16 bita. Ovaj ID se koristi kao identifikator odredišta u naredbi `tc`. Za ***qdisc***, sporedni broj je 0. Za klasu, glavni broj je onaj *qdisc-a* kojem klasa pripada. Dakle, ako je pomoćni broj oznake 0, to je ID *qdisc-a*. Inače, to je ID klase čiji je *qdisc* identificiran svojim glavnim brojem. Pri tome vršni odnosno korijenski *qdisc* ima oznaku **`1:0`**. Nadalje, oznaka **`ffff:0`** rezervirana je za ulazni *qdisc*.

**Root qdisc**

Nadalje, svako mrežno sučelje ima izlazni korijenski *qdisc* s oznakom **`1:0`**. Kao što ime sugerira, to je korijen stabla *qdiscs*-a. Kako smo već naučili pôd *qdisc-ovi* su poznati kao klase. Dakle, na vrhu stabla imamo korijen *qdisc-a*. Ostali čvorovi su klase. Kernel je u direktnoj interakciji samo s korijenom stabla (**`1:`**) i stavlja pakete u red čekanja na vrh stabla. Slično, kernel i uklanja pakete iz reda čekanja iz korijena stabla. Paketi se mogu svrstati u jednu od klasa. Razvrstavanje se vrši pomoću filtera koji su priključeni na ***classful qdisc***. Kontrola prometa na izlazu iz mrežnog sučelja zapravo se svodi na izgradnju grana ovog stabla.

**Slika 235.1. Hijerarhijsko stablo *qdisc-a*.**




U radu, paketi mogu primjerice biti klasificirani u lanac, poput:
**1: → 1:1 → 12: → 12:2**
To za konkretan paket (koji je tako klasificiran) znači da će biti dodijeljen klasi **12:2**.

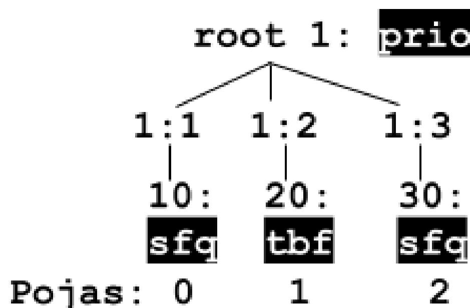
Pogledajmo jedan ovakav primjer pa će vam biti jasnije.
Zamislimo da želimo ograničiti propusnost mrežnog sučelja naziva: **`eth0`**, ali samo za određene kategorije prometa. Za tu potrebo promijenit ćemo standardno postavljeni *qdisc* u **PRIO,** a niže klase u **SFQ** i na **TBF. TBF** smo odabrali jer s njim lako možemo definirati propusnosti mrežnih sučelja.

## Ograničavanje propusnosti mrežnih sučelja ovisno o kategoriji (ToS) mrežnog prometa

Za navedeni primjer primjenjujemo sljedeću konfiguraciju:

```
tc qdisc add dev eth0 root handle 1: prio
tc qdisc add dev eth0 parent 1:1 handle 10: sfq
tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate 20kbit buffer 1600 limit 3000
tc qdisc add dev eth0 parent 1:3 handle 30: sfq
```

**Slika 235.2. Hijerarhijsko stablo *qdisc-a* uz primjer konfiguracije.**




Pogledajmo opis ove konfiguracije:

- U prvom retku definiramo **`prio`** kao korijenski *qdisc* **`1:`** za mrežno sučelje (bežičnu karticu) **`eth0`**. Nakon toga, sustav automatski kreira i klase: **`1:1`**,**`1:2`** i **`1:3`**.
- U drugom retku definiramo kao vršni (**`parent`**) **`1:1`** s naznakom identifikatora klase (**`handle 10:`**) za koju kao zadani *qdisc* definiramo **`sfq`**.
- U trećem retku definiramo kao vršni (**`parent`**) **`1:2`** s naznakom identifikatora klase (**`handle 20:`**) za koju kao zadani *qdisc* definiramo **`tbf`**, kod kojega stavljamo ograničenje propusnosti mreže.
- U četvrtom retku definiramo kao vršni (**`parent`**) **`1:2`** s naznakom identifikatora klase (**`handle 30:`**) za koju kao zadani *qdisc* definiramo **`sfq`**.

Pri tome će interaktivni mrežni promet (s najvećim prioritetom) završiti u pojasu **0** (*handle 10*) ili u pojasu **1** (*handle 20*), a ostali promet s najnižim prioritetom u pojasu **2** (*handle 30*). Ovo ponašanje (klasifikacija prema prioritetu) ovisi konkretnom *qdisc-u*.
Pogledajmo i kako postaviti jednostavniji slučaj ograničavanja propusnosti mrežnog sučelja `eth1` za odlazni promet:

```
tc qdisc add dev eth1 root tbf rate 500kbit burst 16kbit latency 50ms
```

Ovdje smo definirali nekoliko stvari:

- Kao vršni ***qdisc*** smo postavili ***TBF*** (`root tbf`) jer s njim jednostavno možemo ograničavati mrežnu propusnost.
- Zatim smo ograničili propusnost (`rate 500kbit`) i definirati veličinu međumemorije (`burst 16kbit`).
- I na kraju smo dodali i kašnjenje od 50ms (`latency 50ms`).

### *Simuliranje gubitka, kašnjenja i drugih problema u prijenosu paketa*

Određeni *qdisc-ovi*, poput ***netem***, imaju mogućnost simuliranja gubitka mrežnih paketa, što se često koristi kod testiranja aplikacija. Na mrežno sučelje `eth3` ćemo primijeniti **netem** *qdisc* s kojim ćemo simulirati gubitak paketa od 0.1%

```
tc qdisc add dev eth3 root netem loss 0.1%
```

Upotrebom ***netem-a***, moguće je i simulirati i druge stvari, poput:

- Kašnjenja (u *ms*), upotrebom opcije `delay`.
- Kompleksnijeg modela nasumičnih gubitaka paketa, upotrebom opcije `loss genmodel`.
- **ECN** označavanja paketa, upotrebom opcije `ecn` te za dupliciranja paketa, upotrebom opcije `duplicate`.
- Slanja paketa krivim redoslijedom, upotrebom opcije `reorder` te još nekih drugih opcija.



Vezano za klasifikaciju prometa te **QoS** odnosno **ToS** i **DSCP** zaglavlja u **IP** protokolu, pogledajte poglavlje: **23.1.1. Osiguranje kvalitete (*QoS*) i klasifikacija prometa (*ToS i DSCP*).**

## Zrcaljenje mrežnog prometa s jednog mrežnog sučelja na drugo

**Zrcaljenje dolaznog prometa.** Zamislimo poslužitelj na kojem imamo dvije mrežne kartice: dolaznu (`eth0`) i onu [odlaznu] (`eth1`) na koju želimo kopirati i poslati sav dolazni mrežni promet, s dolazne mrežne kartice.
Na dolaznoj mrežnoj kartici (`eth0`) kreirat ćemo dolazni [*ingress*] *qdisc*: Dakle ovo je samo za dolazi mrežni promet.

```
tc qdisc add dev eth0 handle ffff: ingress
```

Osim gore odabranog dodavanja (`add`), moguće je i:

- Brisanje (`delete`).
- Modifikacija u radu (`change`).
- Izmjena (`replace`).
- Ispis (`show`) i drugo.

Sada pogledajmo što smo napravili na ovoj mrežnoj kartici `eth0` (zatamnili smo dio koji smo prethodno dodali):

```
tc -s qdisc ls dev eth0
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5ms interval
100ms memory_limit 32Mb ecn drop_batch 64
 Sent 523885 bytes 6145 pkt (dropped 0, overlimits 0 requeues 0)
qdisc ingress ffff: parent ffff:fff1 ----------------
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

U prvom retku vidimo *qdisc* mehanizam (`qdisc fq_codel`), dakle `fq_codel` je u upotrebi na ovoj mrežnoj kartici.
Potom vidimo da smo dodali dolazni (*ingress*) mehanizam (`qdisc ingress ffff`).

**4.** Sada kreirajmo zrcaljenje svôg prometa sa sučelja `eth0` na odlazno (`egress mirror`) sučelje `eth1`.

```
sudo tc filter add dev eth0 parent ffff: protocol all u32 match u32 0 0 action mirred
egress mirror dev eth1
```

Pogledajmo što smo ovdje napravili:

- Kao dolazno mrežno sučelje smo odabrali `eth0` s oznakom (`parent ffff`).
- Kao mrežne protokole koje želimo kopirati smo naznačili da ćemo koristiti takozvani **u32** filter za sve protokole (`protocol all u32`) te da se unutar njega (`match u32`) filtrira sav promet (`match u32 0 0`). Pri tome (`0 0`) označava sav mrežni promet. ***U32*** je najnapredniji filter odnosno tzv. *klasifikator* prometa (postoji ih više).
- I na kraju govorimo da želimo zrcaljenje (`action mirred`) prema odlaznom (`egress mirror`) sučelju `eth1`.

Sada provjerimo stanje na dolaznom mrežnom sučelju:

```
tc -s -p filter ls dev eth0 parent ffff:
filter protocol all pref 49152 u32 chain 0
. . .
terminal flowid not_in_hw (rule hit 729 success 729)
  match 00000000/00000000 at 0 (success 729 )
  action order 1: mirred (Egress Mirror to device eth1) pipe
  Action statistics: Sent 74560 bytes 729 pkt (dropped 0,overlimits,0 requeues 0)
  backlog 0b 0p requeues 0
```

Vidimo da se broj paketa povećava te da nema grešaka (`rule hit 729 success 729`).
Umjesto zrcaljenja (kopiranja) svog dolaznog mrežnog prometa (svih protokola), kao što smo definirali u točki **1.** moguće je definirati i samo određene mrežne protokole, pa pogledajmo nekoliko takvih primjera:

**5.** Ovdje kreirajmo zrcaljenje samo ICMP paketa (**`match ip protocol 1 0xff`**) sa sučelja: **`eth0`** na odlazno (**`egress mirror`**) sučelje: **`eth1`**.

```
tc filter add dev eth0 parent ffff: protocol ip u32 match ip protocol 1 0xff action mirred egress mirror dev eth1
```

**6.** Sada kreirajmo zrcaljenje samo SSH (port 22) paketa s dva filtera (**`protocol ip u32 match ip protocol 6 0xff ip dport 22 0xffff`**) sa sučelja: **`eth0`** na odlazno (**`egress mirror`**) sučelje: **`eth1`**.

```
tc filter add dev eth0 parent ffff: protocol ip u32 match ip protocol 6 0xff ip dport 22 0xffff action mirred egress mirror dev eth1
```

**Zrcaljenje odlaznog prometa**

Ako također želimo zrcaliti naše izlazne pakete (izlazne s **`eth0`** sučelja), moramo kreirati dodatni *qdisc* i filtere,.
Podsjetimo se da može biti mnogo izlaznih *qdisc-ova*. Mi ćemo jednostavno zamijeniti zadani ***root*** (korijenski) izlazni ***qdisc*** na našem ulaznom sučelju **`eth0`**. Uobičajeno je da se korijenski *qdisc* imenuje s '`1:`'.

Međutim moramo biti svjesni toga da i odlazni (*egress*) *qdiscs* koristi algoritme čekanja za oblikovanje prometa. Stoga ćemo mi koristit ***PRIO*** algoritam jer on ni na koji način ne oblikuje promet, jer svakako ne želimo ometati promet našeg računala/poslužitelja kada ga zrcalimo ili dodatno usporavati sustav. Primijenimo ovaj algoritam:

```
tc qdisc add dev eth0 handle 1: root prio
```

Sada provjerimo dolazno mrežno sučelje (**`eth0`**) i novi algoritam:

```
tc -s qdisc ls dev eth0
qdisc prio 1: root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 930 bytes 7 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Vidimo novi algoritam ***PRIO*** (`prio`) kako smo i htjeli.
I tek sada možemo kreirati odlazni (*egress*) mehanizam (**`action mirred egress`**) i pripadajuće filtere:

```
tc filter add dev eth0 parent 1: protocol all u32 match u32 0 0 action mirred egress mirror dev eth1
```

Što se tiče filtera, odabrali smo sve pakete to jest sve mrežne protokole (**`protocol all u32 match u32 0 0`**).
Provjerimo novo stanje:

```
tc -s -p filter ls dev eth0 parent 1:
filter protocol all pref 49152 u32 chain 0
filter protocol all pref 49152 u32 chain 0 fh 800: ht divisor 1
filter protocol all pref 49152 u32 chain 0 fh 800::800 order 2048 key ht 800 bkt 0
terminal flowid not_in_hw (rule hit 15 success 15)
  match 00000000/00000000 at 0 (success 15 )
        action order 1: mirred (Egress Mirror to device eth1) pipe
        index 2 ref 1 bind 1 installed 6 sec firstused 6 sec
        Action statistics:
        Sent 3770 bytes 15 pkt (dropped 0, overlimits 0 requeues 0)
        backlog 0b 0p requeues 0
```

Za brisanje filtera možemo koristiti sljedeću sintaksu:

```
tc filter del dev eth0 parent ffff:
```

Moguće je nîzati i više filtera, kao u primjeru dolje, gdje smo definirali i drugi tôk (**`flowid 1:3`** → **[pročitajte (1350)]**):

```
tc filter add dev eth0 parent 1:0 protocol ip prio 10 u32 \
match ip protocol 6 0xff \
match u8 0x05 0x0f at 0 \
flowid 1:3
```



Metoda zrcaljenja (kopiranja) kompletnog mrežnog prometa s jedne mrežne kartice na drugu se zove i *Port mirror*, *Switched Port Analyzer* (***SPAN***) ili *Remote Switched Port Analyzer* (***RSPAN***), ovisno o uređaju/OS-u ili proizvođaču.

***I Open vSwitch ima metodu i mehanizme zrcaljenja mrežnog prometa s jednog mrežnog sučelja na drugo!***
***Zrcaljenje prometa može se konfigurirati i pomoću naredbe*** `nmcli`.

**Za postizanje ekstremnih performansi s ubrzanjem na razini hardvera (jer može koristiti hardverske nizove [*multiqueue*]), u određenim scenarijima se preporuča upotreba:** `mqprio` **mehanizma.**

**Izvori informacija: (1344),(1345),(1346),(1347),(1348),(1349),(1350),(1351),(1352),(1353),(1354),(1355),(1356),** **`man tc`,`man tc-u32`,`man tc-fw`,`man tc-route`,`man tc-pfifo_fast`,`man tc-hfsc`,`man tc-tbf`, `man tc-netem`,`man tc-mqprio`,`man nmcli`.**

# 25.2. Dodatne mogućnosti linuxa

U ovoj cjelini govorit ćemo o mehanizmima i tehnikama pomoću kojih je moguće, na računalima odnosno poslužiteljima koji imaju naprednije mrežne kartice, koristiti komplementarne tehnike za povećanje paralelizma obrade mrežnih paketa. Sâmim time poboljšat će se performanse za sustave koji imaju višeprocesorski sustav odnosno za ***SMP*** i ***NUMA*** arhitekturu računala.



Za analizu opterećenja mrežnog sustava na razini hardverskih signala prekida (***IRQ***) kao i softverskih signala prekida, pogledajte sljedeća poglavlja:
**10.5.1.1 IRQ i Linux.**
**10.5.1.3 Servis *ksoftirqd i* softverski *signali prekida*.**

Za osnovne informacije o mrežnoj kartici (za `eth0`) poput upravljačkog programa, inačice, **PCI** utora i slično, pokrenite:

```
ethtool -i eth0

driver: e1000
version: 7.3.21-k8-NAPI
firmware-version:
expansion-rom-version:
bus-info: 0000:00:11.0
```

Nadalje na ovoj razini su moguće i mnoge druge optimizacije koje još nećemo spominjati. Pogledajte detaljne mogućnosti i opcije za neke od često korištenih mrežnih kartica, prema njihovim upravljačkim programima (takozvanim *kernel modulima*):

- `e1000` - za 1Gbps mrežne kartice tvrtke *Intel*: https://www.kernel.org/doc/html/latest/networking/e1000.html
- `ixgbe` - za 10Gbps mrežne kartice tvrtke *Intel*: https://www.kernel.org/doc/Documentation/networking/ixgbe.txt
- `bnx2x` - za 10Gbps mrežne kartice tvrtke *Broadcom*: https://www.supermicro.com/wftp/ISO_Extracted/CDR-X9_1.23_for_Intel_X9_platform/Broadcom/Linux/Driver/README.bnx2x.TXT

**Detaljna dokumentacija za druge mrežne kartice odnosno njihove upravljačke programe (tzv. *kernel module*) i njihove značajke je dostupna na adresi: https://www.kernel.org/doc/Documentation/networking/.**

## 25.2.1. *Receive Side Scaling* (*RSS*)

***Receive Side Scaling (RSS)*** tehnologija je koja distribuira dolazne pakete na više CPU jezgri i to tako da svi paketi koji dolaze s jedne logičke konekcije (pr. pojedina TCP konekcija: klijent-poslužitelj) uvijek završe na obradi na istoj CPU jezgri.
Pri tome svaki prijemni (dolazni) tôk paketa u konačnici završava u jednom prijemnom nîzu za obradu (*receive queue*), a svakom od ovih nîzova se dodjeljuje zaseban hardverski signal prekida (**IRQ** tj. *interrupt*) s kojim upravlja točno određena CPU jezgra, povećavajući performanse cjelokupne obrade. Ova funkcionalnost se odrađuje izračunavanjem *hash vrijednosti* izvorišne i odredišne IP adrese, a na snažnijim mrežnim karticama i ovu zadaću može odrađivati mrežna kartica.



Ako **postoji** hardverska podrška za ***Multiqueue*** (opisan u sljedećem poglavlju), tada **RSS** odrađuje fizička mrežna kartica, koristeći više hardverskih prijemnih (**RX**) nizova/redova. To znači da mrežna kartica sama distribuira promet među RX redovima. Ako mrežna kartica **nema** hardversku podršku za *Multiqueue*, tada se ne može koristiti više hardverskih **RX** redova, ali se ***RSS*** može implementirati u softveru. U tom slučaju, mrežna kartica koristi uobičajeno samo jedan RX red, ali eventualno upravljački program koji podržava **NAPI** (*New API*) može preuzeti posao raspodjele prometa na više CPU jezgri.
**Naravno, hardverskom podrškom za *Multiqueue* se dobivaju višestruko bolje performanse!**

Tvrtka *Intel* je ovu tehnologiju patentirala i nazvala: ***Hashing packet contents to determine a processor***.
I neki drugi proizvođači ju također imaju, ali ju nazivaju drugačije. Naime ovdje se odrađuje i redistribucija signala prekida i raspodjela poslova obrade, ovisno o svakoj konekciji. **RSS** također poznat kao sustav s više prijemnih redova/nîzova (tzv. *receive multi queuing*) distribuira primanje i obradu mrežnih paketa preko nekoliko hardverski baziranih redova za primanje paketa (*receive queues*) omogućujući da se ulazni mrežni promet obrađuje na više procesorskih jezgri. ***RSS*** se može koristiti za ublažavanje efekta uskog grla tijekom primanja mrežnih paketa, uslijed zagušenja sustava s hardverskim signalima prekida (***IRQ***) koji kod velikih opterećenja uzrokuju preopterećenje pojedine jezgre procesora.

Ovaj mehanizam smanjuje i kašnjenje (*delay*) obrade mrežnih paketa. Na nižoj razini to možemo promatrati i na sljedeći način: mrežne kartice koje podržavaju *Multiqueue* i *RSS* imaju mogućnost zapisivanja više dolaznih paketa istovremeno u nekoliko različitih područja RAM memorije; a svaka takva regija memorije pri tome čini zaseban nîz. To omogućuje operativnom sustavu (Linuxu) da koristi više procesorskih jezgri za obradu dolaznih podataka (paketa) paralelno, počevši od hardverske razine.

Pri tome sâma mrežna kartica distribuira pakete primjenjujući određeni filter po kojemu se paketi grupiraju u ove zasebne nîzove, koje obrađuju različite jezgre CPU-a, s čime se dobiva na velikom ubrzanju obrade paketa. Filter koji se ovdje primjenjuje koristi *hash* funkciju na razini IP ili transportnog sloja (TCP/UDP) dakle ili IP adrese ili TCP/UDP porta paketa.

Najčešća implementacija u hardveru mrežnih kartica je upotreba posebne tablice (tzv. *Indirection table*) u kojoj se spremaju identifikatori nîza (*queue*), a za svaki paket se izračunava *hash* funkcija. Potom se od njenog rezultata gleda najnižih sedam bitova koji se koriste kao ključ za raspoređivanje prema identifikatoru nîza i konačnom odabiru onog nîza u koji se taj paket šalje. Neke napredne mrežne kartice imaju i mogućnost korištenja programibilnih filtera, takozvanih ***n-tuple*** filtera, koji se mogu konfigurirati pomoću prekidača `--config-ntuple` unutar naredbe `ethtool`.

Samu *indirekcijsku* tablicu možemo ispisati s naredbom: `ethtool -x` *DEV* (malo slovo **x**), pri tome je *DEV* mrežna kartica.
Odnosno možemo mijenjati ovu tablicu s: `ethtool -X` *DEV* (veliko slovo **x**), pri čemu je *DEV* mrežna kartica.
Naime modificiranjem *indirekcijske* tablice svakom nîzu, mijenjamo „težinu" odnosno mehanizam prema kojem će se određeni mrežni paketi slati u točno određeni **RSS** kanal (tzv. *RX ring*).
Međutim *indirekcijsku* tablicu možemo mijenjati sâmo ako:

- Mrežna kartica hardverski to podržava.
- Ako upravljački program mrežne kartice to podržava te, ako su preko *ethtool*-a podržane dvije funkcije: ***get_rxfh_indir_size*** i ***get_rxfh_indir***.

Pogledajmo trenutnu *indirekcijsku* tablicu za našu mrežnu karticu `eth0` koja ima 8 **RSS** kanala, pomoću naredbe `ethtool`:

```
ethtool -x eth0

RX flow hash indirection table for eth0 with 8 RX ring(s):
    0:      0     1     2     3     4     5     6     7
    8:      0     1     2     3     4     5     6     7
   16:      0     1     2     3     4     5     6     7
   24:      0     1     2     3     4     5     6     7
   32:      0     1     2     3     4     5     6     7
   40:      0     1     2     3     4     5     6     7
   48:      0     1     2     3     4     5     6     7
   56:      0     1     2     3     4     5     6     7
   64:      0     1     2     3     4     5     6     7
   72:      0     1     2     3     4     5     6     7
   80:      0     1     2     3     4     5     6     7
   88:      0     1     2     3     4     5     6     7
   96:      0     1     2     3     4     5     6     7
  104:      0     1     2     3     4     5     6     7
  112:      0     1     2     3     4     5     6     7
  120:      0     1     2     3     4     5     6     7
RSS hash key: Operation not supported
RSS hash function:
    toeplitz: on
    xor: off
    crc32: off
```

Ovu tablicu možemo promatrati i ovako:

| Opseg po osam bitova | Indeks (RX kanal) | Indeks (RX kanal) | Indeks (RX kanal) | Indeks (RX kanal) | Indeks (RX kanal) | Indeks (RX kanal) | Indeks (RX kanal) | Indeks (RX kanal) |
|---|---|---|---|---|---|---|---|---|
| Od 0 do 8 | Bit 0 | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 | Bit 6 | Bit 7 |
| Od 8 do 16 | Bit 8 | Bit 9 | Bit 10 | Bit 11 | Bit 12 | Bit 13 | Bit 14 | Bit 15 |
| Od 16 do 24 | Bit 16 | Bit 17 | Bit 18 | Bit 19 | Bit 20 | Bit 21 | Bit 22 | Bit 23 |
| Od 24 do 32 | Bit 24 | Bit 25 | Bit 26 | Bit 27 | Bit 28 | Bit 29 | Bit 30 | Bit 31 |
| Od 32 do 40 | ... | ... | ... | ... | ... | ... | ... | ... |

U navedenoj tablici prvi stupac s lijeve strane označava bajt (osam bitova) moguće *hash* vrijednost za pakete, a sljedeći stupci predstavljaju bitove čije vrijednost odnosno indeks predstavlja **RX/RSS** kanal. U našem slučaju su to vrijednosti do 128.
Što znači kako je za adresiranje potrebno 7 bitova ($2^7$=128). Tih sedam (7) bitova se prema tome u konačnici koristi za vezanje na pojedini **RSS** RX kanal i u konačnici CPU jezgru.

Vratimo na našu izvornu tablicu. Ako je za naš prvi mrežni paket odabranim *hash* algoritmom dobivena heksadecimalna vrijednost: `0x12345` koja pretvorena u binarno izgleda ovako: `10010001101000101`, pa ako iz toga izvadimo sedam (7) bitova od desno(gore), tada dobivamo binarno: `1000101` što je dekadski vrijednost: `69`.

To znači kako ćemo u našoj *indirekcijskoj* tablici gledati red kojemu počinju indeksi od broja **64**:

| 64: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Indeks broj: | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |

Sada pogledajmo indeks broj `69`. On ima vrijednost `5` što znači kako će ovaj mrežni paket biti ubačen u **RSS**/RX kanal (*queue*) broj `5`.

Ovdje imamo više mogućnosti a najčešće u upotrebi su dvije:

- `equal` **N** s kojom označavamo da želimo podjednako raspodjeljivanje na svih **n** ***RSS (RX)*** kanala.
- `weight` `W0 W1` s kojima dajemo raspon unutar kojeg će se raspoređivati paketi prema ***RSS (RX)*** kanalima.

## Promjena broja RSS kanala

U slučaju kada naša mrežna kartica podržava hardverski *Multiqueue* i samim time *RSS*, moguće je definirati broj kanala za *RSS*.



Za detalje o *Multiqueue* značajki, pogledajte poglavlje:
**25.2.2. Multiqueue funkcionalnost.**

U našem slučaju mrežna kartica hardverski podržava *Multiqueue* i konfigurirali smo 16 *combined Multiqueue* kanala.
Donja optimizacija može primjerice biti korisna ako imamo aplikacije koje ne zahtijevaju korištenje svih 16 redova za RX promet, ili u scenarijima asimetričnog opterećenja gdje imamo mnogo dolaznog prometa (RX), ali manje odlaznog prometa (TX).
Probajmo našu mrežnu karticu `eth0` konfigurirati za navedeni asimetrični promet za raspoređivanje na 12 ***RSS (RX)*** kanala.

```
ethtool -X eth0 equal 12
```

← To je moguće samo i isključivo ako naša mrežna kartica fizički podržava ***Multiqueue*** s minimalno 12 RX ili kombiniranih kanala.

Kako bismo vidjeli statistiku paketa (poslani, primljeni, s greškama i slično) možemo pokrenuti naredbu (za `eth0` sučelje):

```
ethtool -S eth0
```

U ovom vrlo detaljnom ispisu vidjet ćemo svaki ***RSS*** odnosno *rx queue* (0-7), unutar uglatih zagrada **[0], [1], [2],** … posebne statistike, kao i na kraju statistiku za cijelu `eth0` fizičku mrežnu karticu. Naravno pod Linuxom za one mrežne kartice koje to podržavaju, možemo mijenjati i ***RSS*** algoritam po kojem će se izračunavati *hash* vrijednost paketa, a koja će utjecati na to koji mrežni paketi će završiti u kojemu ***RSS*** kanalu (*RSS receive queue*).
To možemo mijenjati u naredbi `ethtool` s prekidačem: `-N rx-flow-hash`.



Za primjere i vrste izračuna ***RSS*** *hash* funkcija, pogledajte sljedeću poveznicu:
https://docs.microsoft.com/en-us/windows-hardware/drivers/network/rss-hashing-types

*Hash* algoritmi koji se koriste, koriste se za mnoge protokole. Kako bismo vidjeli što je postavljeno kao mehanizam za primjerice TCP (Ipv4) *hashing* za ***RSS***, pokrenimo (prekidač je `-n` je za ispis, a `-N` za promjenu vrijednosti):

```
ethtool -n eth0 rx-flow-hash tcp4

TCP over IPV4 flows use these fields for computing Hash flow key:
IP SA
IP DA
L4 bytes 0 & 1 [TCP/UDP src port]
L4 bytes 2 & 3 [TCP/UDP dst port]
```

To znači da se *hash* računa redom na osnovi: *IP Source IP* te *IP Destination IP*, pa TCP ili UDP *source* porta i TCP ili UDP *destination* porta. Sve se može vidjeti/mijenjati za: `tcp4|udp4|ah4|esp4|sctp4|tcp6|udp6|ah6|esp6|sctp6m`



Za više primjera upotrebe naredbe **`ethtool`**, pogledajte poglavlje:
**25.7.10. Naredba ethtool.**

**Izvor informacija za ovu cjelinu: (365.1),(368),(369),(370),(371),(1036), `man ethtool`.**
**Pogledajte i plakat P5.**

## 25.2.2. Multiqueue funkcionalnost

Vezano za broj *Receive Side Scaling* (***RSS)*** kanala odnosno nîzova koje smo spomenuli u prethodnom poglavlju: upravljački program koji upravlja mrežnom karticom koja hardverski podržava ovu značajku preko sâmog kernel modula, daje nam parametar s kojim možemo definirati koliki broj hardverskih prijemnih nîzova (*receive queue*) želimo koristiti. Isto je i s odlaznim nizovima (*transmit queue*). Sve ovisno o maksimalnom broju koliko ih sâma mrežna kartica hardverski podržava te broju CPU jezgri koje imamo. U idealnom slučaju bilo bi ih dobro koristiti ih maksimalno onoliko koliko imamo dostupnih CPU jezgri na računalu. Pri tome svaki prijemni nîz (*receive queue*) ili odlazni niz (*transmit queue*) mogu koristi svoj hardverski signal prekida (*IRQ*), ili TX i RX nizovi mogu biti kombinirani, a na kraju se svaki signal prekida veže za točno određenu CPU jezgru. Tako primjerice mrežna kartica za mrežni paket koji je pristigao u pojedini nîz okida signal prekida (*IRQ*) za taj nîz, odnosno aktivira se određena CPU jezgra da obradi taj paket. *PCI ekspres* (***PCIe***) mrežne kartice za ovu funkcionalnost odnosno ovakav rad koriste ***MSI-X*** signale prekida. Gledajući u datoteku s listom poveznica signala prekida i CPU jezgri: `/proc/interrupts` praktično za svaki *RX/i/ili TX queue* nîz preko ***MSI-X*** signala prekida, uz broj signala prekida koji koristi, vidimo i koju CPU jezgru koristi. Kako bismo utvrdili podržava li uopće naša mrežna kartica (mrežno sučelje) ovakav rad koji se naziva ***Multiqueue***, provjerimo jesu li u datoteci `/proc/interrupts` pridruženi višestruki signali prekida (*IRQ*) na više jezgri procesora za pojedinu mrežnu karticu. Na primjer ako gledamo mrežnu karticu `eth0` koja ima podršku za ***Multiqueue*** (filtrirat ćemo sâmo dio vezan za mrežnu karticu `eth0`), na računalu s osam CPU jezgri (jezgre: CPU0 – CPU7), vidjet ćemo sljedeće:

**`cat /proc/interrupts`**

| IRQ | CPU0 | CPU1 | CPU2 | CPU3 | CPU4 | CPU5 | CPU6 | CPU7 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 82: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 454533548 | IR-PCI-MSI-edge | eth0 |
| 84: | **455438** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | IR-PCI-MSI-edge | eth0-fp-0 |
| 85: | 0 | **3608702** | 0 | 0 | 0 | 0 | 0 | 0 | IR-PCI-MSI-edge | eth0-fp-1 |
| 86: | 0 | 0 | **3950485** | 0 | 0 | 0 | 0 | 0 | IR-PCI-MSI-edge | eth0-fp-2 |
| 87: | 3 | 0 | 0 | **3232320** | 0 | 0 | 0 | 0 | IR-PCI-MSI-edge | eth0-fp-3 |
| 88: | 0 | 0 | 0 | 0 | **345262** | 0 | 0 | 0 | IR-PCI-MSI-edge | eth0-fp-4 |
| 89: | 0 | 0 | 0 | 0 | 0 | **0756229** | 0 | 0 | IR-PCI-MSI-edge | eth0-fp-5 |
| 90: | 0 | 0 | 0 | 0 | 0 | 0 | **909557** | 0 | IR-PCI-MSI-edge | eth0-fp-6 |
| 91: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **14769027** | IR-PCI-MSI-edge | eth0-fp-7 |

U ispisu je vidljivo sljedeće, a što je očekivano kod upotrebe *Multiqueue* (i ***RSS-a)***:

- Mrežna kartica `eth0` koristi signal prekida (*IRQ*) broj `82` koji vidimo prema broju signala prekida koji su obrađeni, jer je njihov broj veći od nula (0) na svim procesorskim jezgrama: CPU0 - CPU7.
- Potom vidimo kako imamo više mrežnih sučelja: `eth0-fp-0`**,** `eth0-fp-1`**,** … `eth0-fp-7` dakle njih **8** (**0-7**). Od kojih svaki koristi svoj signal prekida (*IRQ*). Pa tako za `eth0-fp-0` imamo IRQ `84`, za `eth0-fp-1` imamo IRQ `85` i tako dalje. U ovom primjeru imamo slučaj upotrebe takozvanih višenamjenskih *Multiqueue* kanala (`-fp-` u nazivu), a kojih imamo osam (8) [*TX i RX kombinirano*] za svaku fizičku mrežnu karticu. Nekada su kod ovakvih kartica, koje podržavaju sâmo ***kombinirane*** RX/TX kanale, kartice imena: `eth0-rxtx-0`, `eth0-rxtx-1`, `eth0-rxtx-2`,… ili primjerice: `enp1s0f0-TxRx-0`, `enp1s0f0-TxRx-1` i tako dalje. Dakle u nazivu im je `-rxtx-` ili `-TxRx-`.
- Dodatno svaki od pripadajućih signala prekida koji se koristi za svako pojedino mrežno sučelje/kanal: (`eth0-fp-`**`X`**) se veže za određene CPU jezgre, jer vidimo da pojedina CPU jezgra ima znatno veći broj signala prekida (brojčane vrijednosti), a one koje se na koriste imaju vrijednost nula (`0`).

Sve to znači kako su se mrežna sučelja ili ih možemo nazvati *podsučeljima*, fizičke mrežne kartice, odnosno kanalima, povezale svaka preko svog hardverskog signala prekida (*IRQ*) na točno određenu CPU jezgru, a s time smo dobili željenu mogućnost koju nam nudi ***Multiqueue***. Dakle raspodjeljivanje paketa u više nîzova međumemorije (*queues*), a da je pri tome po jedna CPU jezgra zadužena za rad s pojedinom *međumemorijom* i svim mrežnim paketima u njemu.

Vezano za signale prekida (*IRQ*), afinitet svakog od njih prema CPU jezgrama je vidljiv u datoteci: `/proc/irq/`***`IRQ`***`/smp_affinity_list`. Pri tome je ***IRQ*** broj signala prekida. Tako bi za konkretno naš IRQ `82`, to bilo:

```
cat /proc/irq/82/smp_affinity_list
1
```

Dakle samo CPU jezgra 0 (CPU0) [*dobivena vrijednost* `1`] se smije vezati sâmo za ovaj signal prekida. U slučaju da smo imali pokrenut servis `irqbalance` on bi stalno preraspodjeljivao signale prekida po raznim CPU jezgrama što za vrlo brze mreže (10Gbps+) često i nije najbolje rješenje. Ipak ne mora biti nužno rješenje isključiti ovaj servis već ga se može konfigurirati na način u kojem određene signale prekida (*IRQ*) ne dajemo ovom servisu da ih koristi.



Pogledajte i poglavlje: **10.5.1.2 Servis irqbalance.**

Naša konkretna mrežna kartica je ***Broadcom NetXtreme II BCM57810 10*** Gigabit *Ethernet* koja koristi upravljački program `bnx2x` (*kernel modul*). Kod ovog *kernel modula* možemo vidjeti je li uključena opcija za upotrebu ***Multiqueue*** funkcionalnosti. Filtrirali smo odnosno skratili smo ispis, kako bismo vidjeli samo parametre koje podržava ova mrežna kartica:

```
modinfo bnx2x | grep parm:

parm:        num_queues: Set number of queues (default is as a number of CPUs) (int)
```

Dakle za ovaj *kernel modul* ***Multiqueue*** *(i RSS)* je dostupan (naravno uz podršku na razini hardvera mrežne kartice), pošto je dostupan parametar: `num_queues`. Dakle mi imamo ***Multiqueue*** mogućnost podržanu u hardveru naše mrežne kartice!

Za druge mrežne kartice ovaj parametar se može zvati i drugačije, pa prvo pogledajte detaljnije upute za željeni upravljački program (*kernel modul*) vaše mrežne kartice.

Sada na jednostavan način provjerimo broj podržanih ***Multiqueue*** nîzova, s naredbom `ethtool` na sljedeći način:

```
ethtool -l eth0

Channel parameters for eno1:
Pre-set maximums:
RX:             0
TX:             0
Other:          0
Combined:       30
Current hardware settings:
RX:             0
TX:             0
Other:          0
Combined:       8
```

Ovdje je vidljivo kako naš hardver (`Pre-set maximums`) mrežne kartice podržava do 30 kombiniranih kanala (*Multiqueue*), ali da mi koristimo njih samo 8 (`Current hardware settings`) što je u redu jer imamo samo 8 CPU jezgri. Naime, pošto vidimo da naš hardver mrežne kartice podržava do 30 kombiniranih (TX+RX) kanala, zbog toga i naziv „`Combined`". Vidljivo je i da naš hardver ne podržava sâmo RX ili sâmo TX vrstu kanala zasebno, što je vidljivo pod (`Pre-set maximums`) kao:

```
RX:             0
TX:             0
```

Sličnu informaciju o broju RX *mrežnih kanala* ***(RSS)*** možemo vidjeti i sa sljedećom naredbom (filtrirali smo ispis):

```
ethtool -n eth0

8 RX rings available
```

Ovdje cijelo vrijeme govorimo o takozvanim *ring bufferima* odnosno prstenastim međumemorijama. Dakle svako novo ***Multiqueue*** logičko/fizičko mrežno sučelje (pr. `eth0-fp-0`) ima svoj zasebni *ring buffer odnosno* međumemoriju na razini mrežne kartice (hardvera). U našem slučaju je to kombinirano sučelje koje se koristi i za slanje i primanje (RX i TX) odatle i naziv: *`Combined`*. U slučajevima kada želimo povećati broj ovih kombiniranih "*Multiqueue mrežnih sučelja"* to radimo korištenjem velikog slova **`L`** kao prekidača naredbe `ethtool`.

U slučaju da imamo 12 CPU jezgri, povećali bi broj ovih kombiniranih (RX+TX) kanala/sučelja na 12 i to ovako:

```
ethtool -L eth0 combined 12
```

Sve navedeno treba ponovili za sva mrežna sučelja (kartice) za koja želimo ovakve promjene.

Ova konfiguracija ovisi o upravljačkom programu, a u nekim (rjeđim) slučajevima potreban je drugačiji način konfiguracije preko upravljačkog programa mrežne kartice.

***Za ovu radnju vam je potreban direktan pristup računalu jer će vam se prekinuti mrežna konekcija, dakle, oprez!***

Pogledajmo i takav primjer u kojem prvo moramo isključiti trenutni *kernel modul* koji je zadužen za našu mrežnu karticu.

U ovakvom slučaju je to upravljački program (driver) `bnx2x` pa ćemo isključivanje *kernel modula* napraviti na sljedeći način:

```
modprobe -r bnx2x
```

Tek sada ponovno možemo učitati naš upravljački program, ali s navedenim promjenama (`num_queues=12`):

```
modprobe bnx2x num_queues=12
```

Sada će se sva mrežna i ***Multiqueue*** *mrežna sučelja* odnosno broj komunikacijskih kanala ponovno inicijalizirati i bit će ih 12.

U slučaju kada želimo ove opcije uključiti tijekom pokretanja sustava, potrebno je a ovisno o samoj mrežnoj kartici i njenom upravljačkom programu (*kernel modulu*), obično kreirati novu datoteku (imena istog kao *kernel modul*) u direktoriju: `/etc/modprobe.d/`. U našem slučaju (za *kernel modul* `bnx2x`) bi to bila datoteka imena: `/etc/modprobe.d/bnx2x.conf` koja bi trebala sadržavati sljedeći redak (za našu opisanu konfiguraciju):

```
option bnx2x num_queues=12
```

Primjerice za *Intelovu* 10Gbps mrežnu karticu čiji upravljački program je `ixgbe` opcija ***RSS*** se zove `RSS` pa bi tada morali kreirati datoteku imena `/etc/modprobe.d/ixgbe.conf` koja bi za kreiranje 12 *RSS* kanala morala sadržavati redak:

```
options ixgbe RSS=12
```

Dok je za konkretnu *Intelovu* mrežnu karticu moguće definirati i broj kombiniranih nizova s opcijom `QueuePairs`; primjerice:

```
options ixgbe QueuePairs=12
```

***Ne zaboravite:*** *Osim* ***RX*** *nîzova* ***(RSS)*** *za primanje, moguće je imati i hardverske nîzove za slanje odnosno* ***TX*** *nîzove ili kao u prethodnom primjeru kombinirane nizove (RX+TX), sve ovisno o konkretnoj mrežnoj kartici!*

Ako naša mrežna kartica podržava zasebna ***Multiqueue*** mrežna sučelja za primanje (RX) i zasebna za slanje (TX), pojavilo bi nam se po jedno mrežno sučelje za primanje (RX) i po jedno za slanje (TX). Tada bi datoteka s popisom signala prekida izgledala ovako, u slučaju kada imamo uključen Linux servis `irqbalance`, tj. imat ćemo sljedeće stanje (ispis je skraćen):

```
cat /proc/interrupts

 IRQ        CPU0   …   CPU5      CPU6      CPU7
 82:       18219   …  37240     28930     33548   IR-PCI-MSI-edge   eth0
 84:      455438   … 286025         0     60654   IR-PCI-MSI-edge   eth0-rx-0
 85:     1255690   … 332616   1489774    178173   IR-PCI-MSI-edge   eth0-tx-0
 86:    33950485   …    372         0         0   IR-PCI-MSI-edge   eth0-rx-1
 87:           3   …      0     21599         0   IR-PCI-MSI-edge   eth0-tx-1
```

U ispisu je vidljivo kako se signali prekida (*IRQ*) koji se aktiviraju kod primanja i/ili slanja mrežnih paketa stalno prebacuju s jedne CPU jezgre na drugu, pa je i sâmim time broj signala prekida prilično ravnomjerno raspodijeljen između CPU jezgri.
To radi tako da se jedan signal prekida i rad s mrežnim paketima konkretno, obrađuje na jednoj CPU jezgri, pa se prebacuje na drugu CPU jezgru koja ga obrađuje neko vrijeme, te se potom prebacuje na treću CPU jezgru i tako u krug.

Pogledajmo kako povećati broj zasebnih *Multiqueue mrežnih sučelja* i za primanje (RX) i za slanje (TX), naravno sâmo za one mrežne kartice koje to podržavaju. Povećanje radimo na osam kanala za RX i osam kanala za TX:

```
ethtool -L eth0 rx 8 tx 8
```



**RSS** značajka je standardno, u većini slučajeva uključena (**uz napomene iz prethodnog poglavlja!**).



Za analizu i praćenje statistike mrežnih paketa, zbog što bolje optimizacije sustava, pogledajte poglavlje: **25.3 Statistike, analiza i praćenje mrežnih paketa.**

U slučajevima velikih brzina mreže: ***10/20/40/100+ Gbps*** ponekada takvo povezivanje signala prekida prema CPU jezgrama nije optimalno pa je potrebno ručno konfigurirati afinitet odnosno *povezivanje* IRQ-CPU ili isključiti ili dodatno konfigurirati servis: `irqbalance`. Međutim novije inačice `irqbalance` su svjesne ***NUMA*** arhitekture pa ga stoga ipak nije loše imati pokrenutog. Ipak to možemo optimirati na način koji ćemo dolje opisati.



Za Za to prvo pogledajte poglavlje: **10.7.2.1.1 CPU afinitet (CPU Affinity)** odnosno vezano za *IRQ* i afinitet**: 10.5.1.1 IRQ i Linux,** a posebno cjelinu: **Afinitet signala prekida prema CPU jezgrama (*IRQ affinity*).**

Dakle u slučaju kada recimo želimo za signale prekida (*IRQ*): **84**, **85**, **86** i **87** napraviti sljedeće ručno povezivanje, vidljivo u tablici:

| IRQ | CPU jezgra |
|---|---|
| 84 | CPU 0 |
| 85 | CPU 1 |
| 86 | CPU 2 |
| 87 | CPU 3 |

Moramo napraviti sljedeće promjene na sustavu:

```
echo „1‟ > /proc/irq/84/smp_affinity
echo „2‟ > /proc/irq/85/smp_affinity
echo „4‟ > /proc/irq/86/smp_affinity
echo „8‟ > /proc/irq/87/smp_affinity
```

Odnosno kako bi to bilo trajno, gore navedene retke treba dodati u neku skriptu koja će se izvršiti tijekom svakog pokretanja računala.
Druga preporuka nekih proizvođača je korištenje njihovih *startup* skripti za tu namjenu, pogledajte primjere dva proizvođača:

- ***Mellanox:*** http://www.mellanox.com/related-docs/prod_software/MLNX_EN_Linux_README.txt.
  Pogledajte preporuke i mjerenja ovog proizvođača na izvorima informacija: **(372) i (373).**
- ***Chelsio***: http://service.chelsio.com/OEM-IBM/Linux/Unified_Wire/ChelsioUwire-2.9.1.0/README.txt

CPU jezgre koje su prema NUMA topologiji preporučene za korištenje za svaku pojedinu fizičku mrežnu karticu i sve njene pôd kartice (*RSS/rx* kanale) definirane su u datoteci imena: */sys/class/net/KARTICA/device/local_cpulist* pa je za `eth0` karticu to datoteka `/sys/class/net/eth0/device/local_cpulist`. Pogledajmo što sadrži ova datoteka:

```
cat /sys/class/net/eth0/device/local_cpulist
0-7,16-23
```

Ovo što vidimo (`0-7,16-23`) znači kako se za ovu karticu preporuča korištenje samo sljedećih CPU jezgri: **0** do **7** i od **16** do **23**. Ovo je konkretno ***NUMA*** sustav pa su ove preporučene vrijednosti dobro postavljene s obzirom na *NUMA* raspored CPU jezgri koji smo vidjeli s naredbom `numactl` na sljedeći način (ispis smo skratili na, za sada, važne dijelove):

```
numactl -H

available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 32733 MB
node 0 free: 9903 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 24575 MB
node 1 free: 6300 MB
```

Odnosno znamo kako je *NUMA* ***node0*** (fizički CPU 1) spojen na *PCI express* sabirnicu ma matičnoj ploči na koju je spojena naša `eth0` mrežna kartica. Pogledajmo i kako to znamo:

```
cat /sys/class/net/eth0/device/numa_node
0
```

U prethodnom ispisu vidimo vrijednost `0`, a tako je sve i postavljeno u datoteci za našu mrežnu karticu `eth0`: `/sys/class/net/eth0/device/local_cpus` pa ju i pogledajmo na sljedeći način:

```
cat /sys/class/net/eth0/device/local_cpus
00000000,00ff00ff
```

Dakle vidimo kako je sve uredno postavljeno: drugi niz: `00ff00ff` koji upravo to i označava: prve i druge četiri CPU jezgre (zadnje *FF* oznake) koje definiraju CPU jezgre: 0-7 te treća i četvrta *FF* oznaka koje označavaju CPU jezgre: 16-23.
Stanje (brojače) opterećenja hardverskih signala prekida (IRQ) možemo gledati pokretanjem naredbe `mpstat` ovako:

```
mpstat -P ALL 1
```

Gledajući stupac `%irq` za svaku od CPU jezgri (iz prvog stupca), dok su softverski signali prekida vidljivi u stupcu `%soft`.
Dodatno je važno paziti i na to da se i programi koji se pokreću, pokreću na ispravnim CPU jezgrama (kod *NUMA* arhitekture).

**Pokrenut ćemo sljedeći test**
Koristit ćemo `iperf3` program za mjerenje mrežnih performansi na *NUMA* poslužitelju s dva *NUMA node*-a (fizički CPU).
I to sve na istom računalu, kako uopće ne bismo testirali stvarni promet preko mreže, već promet preko mreže unutar istog poslužitelja. Svrha ovog testa je utvrditi ima li razlike pokreću li se program za testiranje na NUMA ***node0*** ili NUMA ***node1*** i to klijentski i poslužiteljski programi. Dakle pitanje je koje performanse će program imati, ako nam je klijentska aplikacija pokrenuta na *NUMA* ***node0*** jezgrama (prvi fizički procesor), kao i poslužiteljska (testovi ***B1*** i B*2*), te što će se dogoditi kada poslužiteljsku aplikaciju pokrenemo na *NUMA* ***node1*** jezgrama odnosno na drugom fizičkom procesoru (testovi **A*1*** i **A*2***).

Mrežna kartica na kojoj testiramo (`eth0`) je 10Gbps ***Broadcom Limited NetXtreme II BCM57810***. Ova mrežna kartica je spojena na PCI *express* sabirnicu spojenu na *NUMA node* (**CPU 0**) odnosno prvi fizički procesor.
Poslužitelj ima 2 x *Intel Xeon* CPU: E5-2658 na 2.10GHz. Dakle ukupno imamo 32 CPU jezgre, raspoređene na sljedeći način:

- Fizički **CPU 0** (*NUMA čvor 0*) ima sljedeće CPU jezgre: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23.
  - + 32 GB RAM na NUMA čvoru 0.
- Fizički **CPU 1** (*NUMA čvor 1*) ima sljedeće CPU jezgre: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31.
  - + 32 GB RAM na NUMA čvoru 1.

**Test A1:**
Pokrećemo `iperf3` poslužitelj na CPU jezgrama unutar ***NODE 0*** (prvi fizički CPU). IP adresa poslužitelja je 192.168.1.1

```
numactl -N0 iperf3 -s
```

Potom pokrećemo i klijenta: namjerno na drugom fizičkom *NUMA* CPU, dakle *NUMA* ***NODE 1***:

```
numactl -N1 iperf3 -c 192.168.1.1
```

Pokrenut ćemo isti test deset (10) puta. Dobili smo prosječnu brzinu prijenosa podataka: **19.3Gbps**

**Test B1:**
Pokrećemo `iperf3` poslužitelj na CPU jezgrama unutar ***NODE 0*** (prvi fizički CPU). IP adresa poslužitelja je 192.168.1.1

```
numactl -N0 iperf3 -s
```

Potom pokrećemo i klijenta: na **istom** fizičkom NUMA CPU dakle isto na *NUMA* ***NODE 0***:

```
numactl -N0 iperf3 -c 192.168.1.1
```

Pokrenut ćemo isti test deset (10) puta, i sada dobivamo prosječnu brzinu prijenosa podataka: **31.6Gbps**
Pri oba testa nismo uopće optimizirali hardverske signale prekida (I*RQ*) i mrežne kartice te dodatno imamo uključen servis: `irqbalance`. Sada ćemo isključiti `irqbalance` i ponoviti sve testove
ž

**Test A2:**
Pokrećemo `iperf3` poslužitelj na CPU jezgrama unutar ***NODE 0*** (prvi fizički CPU). IP adresa poslužitelja je 192.168.1.1

```
numactl -N0 iperf3 -s
```

Potom pokrećemo i klijenta: namjerno na drugom fizičkom *NUMA* CPU dakle *NUMA* ***NODE 1***:

```
numactl -N1 iperf3 -c 192.168.1.1
```

Pokrenut ćemo isti test deset (10) puta i dobivamo prosječnu brzinu prijenosa podataka: **19.6Gbps**
Dakle dobili smo ubrzanje u odnosu na test **A.1**.

**Test B2:**
Pokrećemo `iperf3` poslužitelj na CPU jezgrama unutar ***NODE 0*** (prvi fizički CPU). IP adresa poslužitelja je 192.168.1.1

```
numactl -N0 iperf3 -s
```

Potom pokrećemo i klijenta: na **istom** fizičkom NUMA CPU dakle isto na *NUMA* ***NODE 0***:

```
numactl -N0 iperf3 -c 192.168.1.1
```

Pokrenut ćemo isti test deset (10) puta, i dobivamo prosječnu brzinu prijenosa podataka: **32.3Gbps**
I ovdje smo dobili malo ubrzanje u odnosu na test **B1**.

Ovo je pokazatelj kako se čak i optimirane aplikacije, ako se pokreću na jezgrama procesora na istom fizičkom procesoru (*NUMA* NODE-u) rade znatno brže nego kada se pokreću na jezgrama procesora koje pripadaju različitim fizičkim procesorima (*NUMA* NODE-ovima). Zbog lakšeg razumijevanja, pogledajmo naša mjerenja u tablici:

| Test | Lokacija klijenta: | Lokacija poslužitelja | Rezultat komunikacije: Klijent → poslužitelj |
|---|---|---|---|
| Test **A1** | **NODE1 (drugi** fizički CPU) | **NODE0** (prvi fizički CPU) | **19.3 Gbps** |
| Test **A2** | **NODE1 (drugi** fizički CPU) | **NODE0** (prvi fizički CPU) | **19.6 Gbps** |
| Test **B1** | **NODE0** (prvi fizički CPU) | **NODE0** (prvi fizički CPU) | **31.6 Gbps** |
| Test **B2** | **NODE0** (prvi fizički CPU) | **NODE0** (prvi fizički CPU) | **32.3 Gbps** |



Podsjetite se poglavlja:
**10.7.2.2.2. NUMA CPU i RAM afinitet .**

## Dodatne optimizacije softvera (aplikacije)

Međutim naša aplikacija je visoko optimirana za *NUMA* arhitekturu, što nije slučaj za veliku većinu (modernih) aplikacija. Stoga ćemo ponoviti sva četiri testa korištenjem naredbe `iperf3` i s prekidačem `-Z` koja će tada koristiti posebnu (*Zero* copy) metodu kreiranja paketa (na strani `iperf3` klijenta) u kojoj se RAM memorija minimalno koristi, a tada očekujemo dobiti još veće brzine u sva četiri mjerenja, jer se mrežni paketi brže generiraju, pa dobivamo sljedeće rezultate:

| Test | Rezultat |
|---|---|
| Test **A1** | 44.1 Gbps |
| Test **B1** | 44.7 Gbps |
| Test **A2** | 42.1 Gbps |
| Test **B2** | 44.4 Gbps |

Naime ovdje je vidljivo koliko je odličnom optimizacijom (`iperf3 -Z`) softvera smanjen efekt upotrebe hardverskih signala prekida i kašnjenja koje uvodi **NUMA** arhitektura, ako su CPU jezgre za pokretanje programa krivo odabrane, u odnosu na CPU jezgre koje su zadužene za mrežni hardver i sâm program (aplikaciju). Naime većina programa odnosno aplikacija nije uopće ili je vrlo loše optimirana za mrežnu komunikaciju ili naravno znatnije koristi RAM memoriju (od `iperf3 -Z`) ili intenzivnije troši sistemske resurse. Dodatno mnogi programi nisu u stanju niti prepoznati **NUMA** arhitekturu.

Međutim koliko god da je `iperf3` optimiran za kreiranje mrežnih paketa, on nije u mogućnosti koristiti više od jedne jezgre procesora, pa mu je brzina procesora ograničavajući faktor. Nadalje ovdje kreiramo sâmo jednu konekciju preko koje se radi test performansi, a programi koji rade s tisućama, stotinama tisuća ili milijunima istovremenih konekcija još su (drastičnije) više podložni ovim problemima *NUMA* arhitekture pogotovo, ako su u stanju koristiti više od jedne jezgre procesora.
Mada nama u slučaju testa koji slijedi na 10Gbps niti ovo nije ograničenje jer možemo kreirati pakete brzinom preko 40Gbps. Konačno, sada kada znamo naša ograničenja sustava o broju mrežnih paketa koje može kreirati i primiti, a koji su definitivno iznad 10Gbps, možemo krenuti s testiranjem na pravoj mreži odnosno između dva identična poslužitelja, preko 10Gbps međuveza koje imamo.

Stoga ćemo sada napraviti sâmo testove u kojima ćemo na obje strane (i klijent i server) namjerno pokretati `iperf3` na CPU jezgrama drugog *NUMA node-a* (`numactl -N1`); vidljivo kao ***Test 1*** te u drugom testu na onim *NUMA* jezgrama koje bi morale biti bolje optimizirane NUMA arhitekturi jer su unutar istog *NUMA node-a* (`numactl -N0`); vidljivo kao ***Test 2***.

| Test | Izmjerena brzina |
|---|---|
| Test 1: | 9.42Gbps |
| Test 2: | 9.42Gbps |

Test pokazuje kako nije bilo razlike u mjerenjima, a dobili smo brzine oko 9.5Gbps što je vrlo dobar rezultat. To znači kako u svakom slučaju imamo vrlo snažan hardver i činjenicu da se poslužitelj trenutno ne koristi za ništa drugo odnosno da je potpuno neopterećen drugim zadacima, što obično nije slučaj.
Konačni je zaključak, kako za svaki novi sustav treba napraviti mjerenja i potrebne optimizacije, uz praćenje opterećenja sustava, od CPU-a (naredbe: `top`, `atop` i `mpstat -P ALL 1`) kao i hardverskih signala prekida (*IRQ)*, u statistici pod: `%irq`, te softverskih signala prekida (u statistici pod: `%soft`), vidljivih u ispisu naredbe: `mpstat -P ALL 1`.

Za ispis broja *Multiqueue* kanala u upotrebi za sva mrežna sučelja (po pojedinom sučelju), možemo koristiti i naredbu:

```
ip -d addr
```

*Multiqueue* će tada biti vidljiv u ispisu svakog mrežnog sučelja kao *numtxqueues* i *numrxqueues*, primjerice:

```
. . . numtxqueues 4 numrxqueues 4 . . .
```



Podsjetite se veze ***Multique*** i *Receive Side Scalling* (***RSS***-a), pa proučite i poglavlje:
**25.2.1. *Receive Side Scaling* (*RSS*).**

**Izvori informacija: (363),(364),(365),(366),(367),(373),(396), `man modprobe`, `man numactl`, `man mpstat`, `man ethtool`, `man iperf3`, `man 5 proc`, `man ip`. Pogledajte i plakat P5.**

### 25.2.2.1. Multiqueue i virtualizacija

Upotreba ***Multique*** značajke nije ograničena samo na fizička računala odnosno na fizičke mrežne kartice, već je moguća i unutar virtualnih računala, i to ako koristimo *VirtIO* (paravirtualizirana) mrežna sučelja. *VirtIO* mrežno sučelje nudi nam mogućnost korištenja više nîzova odnosno kanala u mrežnoj komunikaciji, tako da svaki komunikacijski kanal može opsluživati po jedna CPU jezgra (**RSS**). Dakle, ako naše virtualno računalo ima više CPU jezgri, svaka od njih može odrađivati zadatke vezane za primanje i slanje paketa prema mreži odnosno u ovom slučaju prema *hipervizoru* na dalje.
Ova funkcionalnost ovisi o podršci u *hipervizoru*, ali i nižim slojevima kao što je primjerice ***TAP*** sučelje.

*Od kernela inačice 3.8, Linux podržava Multiqueue značajku i na* ***tun/tap*** *mrežnim sučeljima!*



Vezano za ***TAP*** sučelje, pogledajte poglavlje:
**20.6.3. TUN i TAP - posebna mrežna sučelja.**

Ako primjerice imamo dvije CPU jezgre za virtualno računalo (*vCPU*) i želimo uključiti ovu funkcionalnost, tada će opcije i parametri s kojima pokrećemo naše virtualno računalo biti nešto poput *[ovisi o inačici QEMU/KVM hipervizora]*

```
...-device virtio-net-pci,mq=on,vectors=6,…
```

Pri čemu smo broj šest (**6**) dobili na sljedeći način: 2(rx + tx queue nîz) x **2CPU** + 2za konfiguraciju i kontrolu (config + control).
Tada, ako unutar virtualnog računala pokrenemo slijedeću naredbu vidjet ćemo ova dva (2) ***Multiqueue*** nîza:

```
ethtool -l eth0
Channel parameters for eth0:
Pre-set maximums:
Combined:  2
Current hardware settings:
Combined:  2
```

Dakle sada vidimo da se za mrežno sučelje **eth0** koriste dva (2) *multique* niza. Na ovom testom virtualnom računalu imamo samo jedno ***VirtIO*** mrežno sučelje (Ethernet). Pogledajmo detalje o njemu na drugačiji način (skratili smo ispis):

```
ip -d a
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
1000
    link/ether e6:91:ab:6a:ea:a7 brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 68 maxmtu
65535 numtxqueues 2 numrxqueues 2 gso_max_size 65536 gso_max_segs 65535
    inet 192.168.1.102/24 brd 192.168.1.255 scope global dynamic noprefixroute eth0
       valid_lft 863998917sec preferred_lft 863998917sec
    inet6 fe80::e491:abff:fe6a:eaa7/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Vidimo da se koriste dva *Multiqueue* nîza kako smo i očekivali (`numtxqueues 2`) i `(numrxqueues 2)`.
To znači dva niza za primanje (RX) i dva za slanje (TX).
Vidjet ćemo i kako se *VirtIO* mrežno sučelje raspodjeljuje između CPU jezgri zbog ***Multiqueue*** (filtrirali smo ispis, samo za ***VirtIO***), tako da imamo više signala prekida (IRQ), a pri tome je za jedan IRQ zadužena pojedina jezgra procesora:

```
cat /proc/interrupts
         CPU0      CPU1
  0:      117         0   IO-APIC    2-edge      timer
 10:      207         0   IO-APIC   10-fasteoi   virtio0
 24:        0         0   PCI-MSI 131072-edge    virtio2-config
 25:        0        14   PCI-MSI 131073-edge    virtio2-virtqueues
 26:        0         0   PCI-MSI 81920-edge     virtio1-config
 27:        0         0   PCI-MSI 81921-edge     virtio1-control
 28:        0         0   PCI-MSI 81922-edge     virtio1-event
 29:     1981         0   PCI-MSI 81923-edge     virtio1-request
 30:        0      2680   PCI-MSI 81924-edge     virtio1-request
 31:        0         0   PCI-MSI 294912-edge    virtio3-config
 32:       18       649   PCI-MSI 294913-edge    virtio3-input.0
 33:      157        54   PCI-MSI 294914-edge    virtio3-output.0
 34:      173         9   PCI-MSI 294915-edge    virtio3-input.1
 35:       70        76   PCI-MSI 294916-edge    virtio3-output.1
```

Signali prekida (IRQ) se nalaze u prvom stupcu, dok je njihov broj prema CPU jezgrama (***CPU0*** i ***CPU1***) vidljiv u drugom i trećem stupcu. U krajnjem desnom stupcu su ***VirtIO*** ulazna (`input`) i izlazna (`output`) sučelja koja to sve i odrađuju.



Pogledajte i poglavlje:
**27.1.2. Rad s virtualizacijom (KVM+QEMU).**

**Izvori informacija: (703),(885),(1591)**,`man 5 proc`**. Pogledajte i plakat P5.**

### 25.2.3. Receive packet steering (RPS), Receive flow steering (RFS) i Transmit Packet Steering (XPS)

U ovoj cjelini govorit ćemo o dvije tehnologije: ***Receive packet steering*** **(RPS)** i ***Receive flow steering*** **(RFS).**
Naime postoje i dvije implementacije slične ***RSS*** mehanizmu, ali na razini softvera, a to su upravo navedene tehnologije:

- Koordinacija odnosno mehanizam upravljanja zaprimanjem paketa (***RPS***).
- Mehanizam upravljanja protokom zaprimljenih paketa (***RFS***).

Naime kako smo već vidjeli, poboljšanje performansi može se postići usmjeravanjem zahtjeva za hardverske signale prekida (*IRQ*) na željene CPU jezgre. Dodatno, određena CPU jezgra nije rezervirana samo za prihvaćanje mrežnih paketa već i za druge programe. Nadalje ako mrežna kartica nema ***Multique*** (i ***RSS***) značajku, pomoću koje može imati više nizova (*queue*), pri čemu svaki niz ima svoj hardverski signal prekida, koji može obrađivati neka druga jezgra procesora i time drastično ubrzati (paralelnom obradom) dohvaćanje paketa, ponovno imamo problem.
Međutim svaka pojedina CPU jezgra koja je dohvatila ili poslala mrežni paket preko hardverskih signala prekida, mora u konačnici taj paket proslijediti na obradu aplikaciji koja je zapravo krajnje odredište za mrežne pakete.
Zbog toga je pogotovo kod *NUMA* sustava, ali i kod klasičnih *SMP* sustava, potrebno uzeti u obzir i aplikacijski lokalitet.
To znači kako bi bilo dobro paziti koje CPU jezgre odrađuju signal prekida (*IRQ*) pa bi bilo dobro da iste CPU jezgre pokreću i aplikaciju ili da se barem, kod *NUMA* sustava, aplikacija nalazi odnosno pokreće unutar istog fizičkog *NUMA* procesora.
Što u konačnici rezultira većom brzinom obrade, smanjenom latencijom i boljom upotrebom hardvera, a što je rezultat i boljeg korištenja CPU međumemorija (*L1*, *L2* i *L3*) te manjeg konteksta prebacivanja aplikacija (*context switch*).

Dakle ***RPS*** je praktično implementacija na razini softvera onoga što je kod ***RSS***-a (*Multique*) implementirano u hardveru. To radi tako da **RPS** balansira raspodjelu softverskih signala prekida između više procesorskih jezgri, slično kao što *Multique* (***RSS***) radi s hardverskim signalima prekida (znanim kao **IRQ**). To znači kako se ***RPS*** može koristiti i na sustavima koji nemaju posebne mrežne kartice koje imaju hardversku podršku za ***RSS***.
To isto tako znači kako se ***RPS*** može koristiti i kada već koristimo ***RSS***, jer se ***RPS*** ionako nalazi na višoj razini OS-a odnosno u softveru. Mada se to češće ne preporuča nego preporuča jer često možemo više usporiti sustav nego dobiti neke prednosti.

Dodatno za *RPS* možemo lagano dodavati nove filtere za (nove) mrežne protokole, što nije baš najjednostavnije (ili moguće) za *RSS*. Osim toga *RPS* u radu ne povećava broj hardverskih signala prekida (IRQ), već sâmo softverskih (*soft IRQ*) koji su ionako znatno brži od hardverskih. Mala negativna strana je veće korištenje signala prekida među procesorima (*inter-processor interrupts* (**IPIs**)) u radu, a što ne predstavlja nužno problem (ovisno o opterećenju sustava).
Povećanje odnosno sve statistike softverskih signala prekida možemo vidjeti praćenjem sadržaja posebne datoteke u koju se zapisuje statistika softverskih signala prekida: `/proc/net/softnet_stat`.



Navedena statistika je objašnjena u poglavlju:
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*]** i to dio:
„***Kako biti siguran dolazi li do prepunjavanja backlog međumemorije***“.

Dakle unutar datoteke `/proc/net/softnet_stat` u desetom stupcu, polje: `sd->received_rps`, vidimo koliko puta je CPU aktiviran za procesiranje mrežnih paketa preko ***IPI*** signala prekida (*Inter-processor Interrupt)* s kojim jedna CPU jezgra šalje signal prekida (traži procesorsko vrijeme) prema drugoj CPU jezgri i to konkretno za potrebe ***RPS*a**.
Dodatno, uključivanjem ***RPS***a dolazi do većeg opterećenja procesora zbog dodatnog procesiranja paketa koji se obrađuju pomoću softverskih signala prekida (*soft IRQ*) pa se statistika ovih aktivnosti može vidjeti u datoteci: `/proc/softirqs` i to u retku: `NET_RX`. Međutim, prvo provjerimo je li naš kernel uopće kompiliran s podrškom za ***RPS***:

```
grep -i CONFIG_RPS /boot/config-`uname -r`
CONFIG_RPS=y
```

Ako imamo vrijednost `=y` tada je ova funkcionalnost dostupna. Slično kao i kod ***RSS***-a i ovdje postoji više mrežnih nîzova (*queue*) odnosno međumemorija za primanje paketa, ali softverskih. Za svaki od ovih nîzova moguće je definirati pripadnost željenoj/željenim CPU jezgrama, za svako mrežno sučelje odnosno mrežnu karticu. Za našu mrežnu karticu `eth0` pogledajmo koliko *RPS* RX (prijemnih) nîzova imamo dostupno. Oni su definirani unutar direktorija:
`/sys/class/net/`**`UREĐAJ`**`/queues/`. Pri tome **`UREĐAJ`** predstavlja mrežno sučelje: *`eth0`*, *`bond0`*, …

```
ls -l /sys/class/net/eth0/queues/ | grep rx-

drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-0
drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-1
drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-2
drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-3
drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-4
drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-5
drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-6
drwxr-xr-x 2 root root 0 Oct 18 10:49 rx-7
```

U našem slučaju vidimo da imamo osam (0-7) ***RPS RX*** nîzova. Unutar svakog nîza nalaze se datoteke u kojima se definira pripadnost CPU jezgrama a naziv ove datoteke je `rps_cpus`. Unutar ove datoteke je definirana bît maska za CPU jezgre, ista kao i z *RSS* (o tome smo već govorili u poglavlju prije).

Ako su sve vrijednosti za sve RX nîzove (*queues*) jednake nuli, poput:

```
00000000,00000000
```

To znači kako je *RPS* isključen, što je obično standardna postavka Linux sustava.



Za mrežne kartice koje podržavaju hardversku ***Multique*** funkciju odnosno ***RSS*** nema potrebe koristiti i softversku podršku za praktično istu funkcionalnost odnosno ***RPS***, pogotovo kod visoko performantnih sustava.

Međutim, ako imamo mrežne kartice koje podržavaju ***RSS***, upotreba ***RPS***-a u nekim slučajevima može i pomoći.
Pri tome je važno znati kako i određene vrste logičkih mrežnih sučelja, poput agregiranih/spojenih/*bond*-anih također mogu koristiti ***RPS*** funkcionalnost, kao da se radi o fizičkim mrežnim karticama.
Dakle ovdje govorimo o mrežnim sučeljima imena: `bond0`, `bond1`, `bond2`,… Statistiku koliko je *RPS* tokova podataka obrađeno za pojedini RPS kanal (*queue)*, možemo pronaći u datoteci imena: `rps_flow_cnt` koja se nalazi unutar direktorija koji smo gore spomenuli. Ako želimo uključiti RPS za nîzove: `rx-0` do `rx-7` moramo za svaki od njih zasebno postaviti `rps_cpus` vrijednosti.

To ćemo napraviti na sljedeći način (izvršavanjem jedne po jedne naredbe navedene dolje):

```
echo „00000000,00000001" > /sys/class/net/eth0/queues/rx-0/rps_cpus
echo „00000000,00000002" > /sys/class/net/eth0/queues/rx-1/rps_cpus
echo „00000000,00000004" > /sys/class/net/eth0/queues/rx-2/rps_cpus
echo „00000000,00000008" > /sys/class/net/eth0/queues/rx-3/rps_cpus
echo „00000000,00000010" > /sys/class/net/eth0/queues/rx-4/rps_cpus
echo „00000000,00000020" > /sys/class/net/eth0/queues/rx-5/rps_cpus
echo „00000000,00000040" > /sys/class/net/eth0/queues/rx-6/rps_cpus
echo „00000000,00000080" > /sys/class/net/eth0/queues/rx-7/rps_cpus
```

Postavili smo vrijednosti kojima se svaki *RPS RX* nîz dodjeljuje po jednoj jezgri procesora, redom: CPU0 – CPU7.
Ovdje je moguće i definirati ograničenja tokova paketa, koja završavaju unutar određenih *RPS queue*/nîzova/kanala.

*RPS* standardno zaprima mrežne pakete i prosljeđuje ih u jedan od *RPS* nîzova/kanala, bez mehanizma koji bi radio dodatno preslagivanje paketa. Naime paketi koji pripadaju istom logičkom toku podataka, uvijek završe u istom *RPS* nîzu, a koji obrađuje uvijek ista jezgra procesora (koju smo definirali gore). Međutim u graničnim slučajevima, kada između više tokova paketa postoji velika razlika u brzini dolaska paketa, određeni nîz paketa koji završava u nekom *RSS* kanalu može zagušiti tu jezgru procesora i dalju obradu. Takav tok podataka stoga može uzrokovati probleme odnosno usporavanja u radu mrežnog podsustava. Stoga je uveden dodatni mehanizam za ograničenje toka (engl. ***RPS Flow Limit***) koji se aktivira u graničnim slučajevima. On radi tako da se prioritiziraju manji/sporiji tokovi paketa, a na bržim tokovima se počinju odbacivati paketi, ali tek u slučajevima kada su CPU jezgre koje obrađuju tokove koji uzrokuju vrlo veliko opterećenje CPUa preopterećene.
Naime jednom kada se centralni ulazni nîz (*queue)* prepuni s više od polovice, a čija je veličina definirana u varijabli (`net.core.netdev_max_backlog`) počinje se s prebrojavanjem paketa, na razini svakog nîza odnosno svakog *RPS* kanala, sve do zadnjih 256 primljenih paketa.

Ako taj tok prelazi 50% (ova granica se može mijenjati) svi novi paketi u tom toku će biti odbačeni, a što vrijedi sâmo za problematičan tok paketa, dok ostali tokovi odnosno *RSS* kanali ne bivaju zagušeni. *RPS Flow limit* je ponekada aktiviran kao opcija i to, ako je *RPS* trenutno aktivan. Provjerimo *RPS Flow limit* na razini kernela:

```
grep -i CONFIG_NET_FLOW_LIMIT /boot/config-`uname -r`
CONFIG_NET_FLOW_LIMIT=y
```

Ako imamo postavljenu vrijednost `=y` tada je moguće koristiti *RPS Flow limit* funkcionalnost što znači kako je naš kernel kompiliran s njom. Ovdje imamo dvije `sysctl` opcije:

- `net.core.flow_limit_cpu_bitmap` – ako definiramo vrijednost ove varijable (nakon što je RPS aktiviran), odnosno, ako joj postavimo bît masku za CPU jezgre kojima dodjeljujemo ***RPS Flow Limit*** *funkcionalnost* tada ovu funkcionalnost i aktiviramo.
- `net.core.flow_limit_table_len` - ako je prva opcija aktivna tada ovdje definiramo veličinu *hash* tablice koja se koristi za ograničavanje RPS toka (***RPS Flow Limit***).

**Izvori informacija: (365),(397),(398),(399),(400),(401), (1037),(1038),(1378), `man sysctl`,`man 5 proc`, `man 5 sysfs`. Pogledajte i plakat P5.**

## ***Receive Flow Steering*** *(RFS)*

Druga tehnologija o kojoj ćemo govoriti je ***Receive Flow Steering*** *(RFS)*. Naime **RFS** tehnologija s druge strane ne koristi samo *hash* funkciju za preraspodjelu primljenih mrežnih paketa prema CPU jezgrama (tzv. pseudo raspodjelu) već puno bolji način raspodjele pomoću dodatne tablice, a koja kao rezultat im bolju raspodjelu paketa na CPU jezgre na kojima i trebaju završiti paketi ovisno o aplikacijama koje ih koriste. Međutim *RFS* se prvo oslanja na *RPS* koji mu je potreban za rad.
U svakom slučaju navedena tehnologija može donijeti znatna ubrzanja rada mreže. Važno je razumjeti kako je ova tehnologija implementirana unutar softvera, a ne hardvera. poput **aRFS** koja je u hardveru i koju primjerice ***Intel*** naziva *Ethernet Flow Director*, koja je podržana na mrežnim karticama poput *Intel*: X520, X540, XL710 ili novijim (snažnijim).



Za ***RPS*** tehnologiju za *FreeBSD* Unix pogledajte izvor informacija **(17)** te za ***RFS*** izvore informacija**: (18),(22).**

**Vratimo se na Linux.**
Kada *task scheduler* prebacuje programske nîti odnosno obradu na novi CPU dok je prijašnji paket prethodno zaprimljen na staroj CPU jezgri, paketi mogu početi stizati izvan reda kojim bi trebali biti obrađeni i proslijeđeni.

Kako bi se to izbjeglo ***RFS*** koristi drugu tablicu protoka za praćenje paketa. Međutim kako bi *RFS* bilo moguće koristiti, prvo i sâm kernel mora biti kompiliran s ovom mogućnošću odnosno prvo s podrškom za *RPS*.
Provjerimo imamo li podršku za *RPS* na razini kernela (bez obzira koristimo li ovu funkcionalnost ili ne):

```
grep CONFIG_RPS /boot/config-`uname -r`
CONFIG_RPS=y
```

Pošto imamo `=y` sve je u redu jer imamo podršku za nju iako je ova funkcionalnost obično standardno isključena kao i *RFS*. Ako ***RFS*** želimo koristiti prvo moramo uključiti ***RPS*** (opisano u prijašnjem poglavlju). Važno je znati kako *RFS* koristi već navedenu globalnu tablicu raspodjeljivanja svih tokova. Veličina ove tablice može se podesiti postavljanjem *sysctl* varijable: `net.core.rps_sock_flow_entries`.
Ona je dostupna i kao posebna datoteka u */proc* strukturi direktorija:
`/proc/sys/net/core/rps_sock_flow_entries`. Ako smo sve uključili, možemo ju primjerice povećati na 32.768, što označava maksimalan broj svih očekivanih istovremenih mrežnih konekcija na sustav:

```
sysctl -w net.core.rps_sock_flow_entries=32768
```

Dodatno potrebno je odrediti i broj ovih tokova za svako RX sučelje naše mrežne kartice (pr. `eth0`).
Stoga krenimo od prvog RX sučelja (**rx-0**) naše `eth0` mrežne kartice:

```
echo 2048 > /sys/class/net/eth0/queues/rx-0/rps_flow_cnt
```

Isto tako trebamo napraviti za sva ostala (***rx-0*** do ***rx***-XY).
S ovime smo kreirali 2.048 nîzova na svakom RX sučelju naše mrežne kartice `eth0`.

---

## *Hardverski ubrzani RFS (aRFS)*

Neke snažnije mrežne kartice poput navedenih ***Intelovih*** X520, X540, XL710 i novijih, podržavaju i hardverski ubrzanu mrežnu ***RFS*** funkcionalnost koja se tada naziva ***Accelerated RFS*** ili ***aRFS***. Naime poput klasičnog *RFSa*, paketi se ovdje prosljeđuju na temelju lokacije aplikacije koja treba obraditi određeni paket odnosno nîz paketa. Međutim za razliku od tradicionalnog *RFSa*, paketi se šalju izravno na CPU jezgru koja je lokalno u upotrebi za izvršavanje programa ili njegove programske nîti, koja i treba te podatke, odnosno koja ih treba obraditi.
Dakle govorimo o CPU jezgri koja pokreće tu aplikaciju ili fizički procesor u NUMA arhitekturi prema CPU jezgrama koje su lokalne za taj CPU NUMA *node*. Za ovu funkcionalnost je prvo potreban kernel (modul) koji ju uopće podržava odnosno koji je kompiliran s podrškom za nju.

Provjerimo imamo li ovu funkcionalnost u našem kernelu, sa sljedećim nîzom naredbi:

```
grep CONFIG_RFS_ACCEL /boot/config-`uname -r`
CONFIG_RFS_ACCEL=y
```

Pošto imamo `=y` sve je u redu što znači kako imamo podršku u kernelu.
Drugi korak je provjera podržava li to mrežna kartica, što možemo vidjeti s naredbom `ethtool` na sljedeći način:

```
ethtool -k eth0 | grep ntuple
```

Ako dobijemo nešto poput:

```
ntuple-filters: off
```

Odnosno imamo li ovdje vrijednost `off` te nemamo ništa u zagradi nakon toga, odnosno u zagradi nemamo i (`fixed`) to znači kako ipak imamo hardversku podršku. Ovdje vidimo pojam *ntuple* koji predstavlja ***aRFS*** filtriranje.
Pogledajmo primjer u kojem mrežna kartica ne podržava (hardverski) *aRFS*:

```
ntuple-filters: off [fixed]
```

Ako naša mrežna kartica ipak podržava *aRFS*, ali on nije uključen, probajmo ga uključiti sa:

```
ethtool -k eth0 ntuple on
```

Ako je sve u redu, provjerit ćemo status:

```
ethtool -k eth0 | grep ntuple
```

I dobit ćemo:

```
ntuple-filters: on
```

Dakle s ovime smo uključili ovu funkcionalnost.
Isključivanje ove funkcionalnosti za mrežnu karticu `eth0` možemo napraviti sa:

```
ethtool -k eth0 ntuple off
```

Dodatno u posebnim rijetkim slučajevima, moguće je i ručno kreirati *ntuple* (**aRFS**) filtere prema posebnim potrebama. Recimo da za sve mrežne pakete koji dolaze s IP adrese: 10.10.10.10, a trebaju doći na IP adresu: 20.20.20.20 želimo da uvijek završe u **aRFS** nîzu (*queue*) broj pet (**5**):

```
ethtool --config-ntuple flow-type tcp4 src-ip 10.10.10.10 dst-ip 20.20.20.20 action 5
```

Moguće je raditi i dodatne optimizacije, koje sada nećemo objašnjavati.
Ako želimo izlistati sve *ntuple* filtere koje smo dodali, to možemo napraviti sa (za `eth0` mrežnu karticu):

```
ethtool --show-ntuple eth0
```

Ili ako želim obrisati točnom određeni (filter broj **N**) od ručno dodanih filtera, potrebno je napraviti sljedeće:

```
ethtool --config-ntuple eth0 delete N
```



Važno je znati da osim što je za **aRFS** potrebna hardverska podrška na razini mrežne kartice, potrebna je i podrška unutar upravljačkog programa za konkretnu mrežnu karticu (*kernel modula*). To ponekad znači kako je potrebno instalirati ili kompilirati novi upravljački program (*kernel modul*) za konkretnu mrežnu karticu.

**Izvori informacija: (365),(365.1),(402.1),(459), `man ethtool`, `man sysctl`. Pogledajte i plakat P5.**

---

## ***Transmit Packet Steering*** *(XPS)*

***Transmit Packet Steering*** *(**XPS**)* tehnologija je slična ***RFS*** tehnologiji, ali djeluje u suprotnom smjeru: ne za primanje mrežnih paketa već za njihovo slanje. Dakle svaki zasebni tok mrežnih podataka se šalje na drugu CPU jezgru. Dakle ovdje se radi o upravljanju prijenosom paketa u kojemu je mehanizam za odabir za odašiljanje baziran na odabiru reda slanja (*queue*) na koji će se paket poslati. Ovdje govorimo o redu za čekanje (*TX queue*) koji je implementiran u mrežnoj kartici, a vrijedi za one mrežne kartice koje to hardverski podržavaju. To se može postići upotrebom dvije metode:

- (**1**) Povezivanjem procesorskih jezgri na hardverski red slanja (*TX queue*).
- (**2**) Povezivanjem redova primanja (*RX queue*) prema redovima slanja (*TX queue*).

***XPS*** je dostupan sâmo, ako je omogućen u kernelu, a vidljiv kao opcija `CONFIG_XPS` što je standardno za *SMP* sustave. Međutim ova funkcionalnost ostaje onemogućena sve dok nije izričito konfigurirana.

Da biste omogućili *XPS* potrebno je konfigurirati sustav pomoću *sysfs* datoteke i to:

- Ako se odlučimo za prvi mehanizam (**1**) što se definira kao CPU maska (koju smo već prije objasnili), a podešava se u `/sys/class/net/<dev>/queues/tx-<n>/xps_cpus`.
  Pri tome je `<dev>` mrežna kartica a `tx-<n>` *TX queue* za koji konfiguriramo XPS prema prvom modelu (**1**).
- Ako se pak odlučimo za drugi model upotrebe (**2**), on se podešava u: `/sys/class/net/<dev>/queues/tx-<n>/xps_rxqs`.
  Pri tome je isto `<dev>` mrežna kartica, a `tx-<n>` *TX queue*.

Kod prvog mehanizma (**1**) koristimo standardnu CPU masku, s kojom definiramo koja CPU jezgra će koristiti koji TX kanal pa tako možemo napraviti sljedeće. Ako primjerice koristimo prva četiri TX kanala (*TX-0* do *TX-3*):

```
echo 01 > /sys/class/net/eth0/queues/tx-0/xps_cpus
echo 02 > /sys/class/net/eth0/queues/tx-1/xps_cpus
echo 04 > /sys/class/net/eth0/queues/tx-2/xps_cpus
echo 08 > /sys/class/net/eth0/queues/tx-3/xps_cpus
```

Dakle ovdje smo za svaki TX kanal definirali pripadnost točno definiranim CPU jezgrama, koristeći CPU bit masku.

**Izvori informacija:** (**20**),(**365**),(**365.1**),(**465**),(**466**), **`man 5 sysfs`. Pogledajte i plakat P5.**

---

# 25.3. Statistike, analiza i praćenje mrežnih paketa

U svim gore navedenim, ali i drugim primjerima upotrebe, zbog potrebe analize ponašanja rada i grešaka na mreži, potrebno je pratiti statistike rada mrežne kartice i mrežnih paketa. U ovoj cjelini ćemo prvo analizirati moguće uzroke problema u mreži.
Prvi korak je analiza rada mrežnog sučelja (kolokvijalno rečeno mrežne kartice) po pitanju signala prekida (***IRQ***), kao i eventualnu prilagodbu afiniteta upotrebe određene/određenih CPU jezgri zaduženih za rad s mrežnom karticom i ***IRQ-om.***
Obratite pažnju na hardverske signale prekida (***IRQ***), koji se vide u posebnoj datoteci `/proc/interrupts`, ali i pažnju na statistike vezane za softverske signale prekida u datoteci: `/proc/softirqs` i to poglavito statistike vezane za slanje paketa (`NET_TX`) kao i primanje (`NET_RX`), prema vidljivim jezgrama CPU-a. Stoga pogledajte poglavlje: **10.5.1.1. IRQ i Linux**.

Nadalje, provjerite koristite li upravljački program koji je kompiliran s podrškom za tzv. **NAPI** i *Pooling (pr. za* `eth0` *mrežnu karticu),* što nam daje napredne mogućnosti rada. Za ovu provjeru koristimo naredbu `ethtool` na sljedeći način;

```
ethtool -i eth0

driver: e1000
version: 7.3.21-k8-NAPI
```

Ispis smo skratili sâmo na dio u kojemu se vidi kako konkretan upravljački program (`e1000`) ima podršku za ***NAPI*** (`NAPI`).


Za detalje proučite poglavlja: **10.5.2.1.Pooling i IRQ** kao i mehanizam znan kao ***Interrupt moderation*** koji je opisan u poglavlju: **10.5.2.2. *Interrupt moderation***.

Sljedeća stvar optimizacije ili eventualni uzrok problema može biti upotreba tzv. ***IOAT*** odnosno ***DCA*** tehnologije koja je opisana u poglavlju: **10.6.1. Napredne DMA tehnologije.**

Također uzrok problema mogu biti i optimizacije upravljačkih programa mrežnih kartica (pogledajte poglavlje: **11.1.1.3.).**
Osim toga postavki brzine, *duplex* i drugih parametara rada mrežne kartice isto mogu uzrokovati probleme u radu.


Pogledajte poglavlje: **19.6.1. Konfiguracija brzine te *duplex* i *auto negotiation* načina rada** te povezana poglavlja.

Ono na što još trebamo obratiti pažnju je optimizacija upotrebe prstenaste međumemorije fizičkih mrežnih kartica (tzv. *Ring Buffer*) za slanje (***TX***) i primanje (***RX***). To je međumemorija u koju se prvo spremaju mrežni paketi, kada dođu do mrežne kartice ili kada se šalju (poglavlje: **25.1.1**). Njenu veličinu (za `eth0` mrežnu karticu) možemo vidjeti sa sljedećom naredbom:

```
ethtool -g eth0
```

Osim toga, moguće je da se radi i o ne optimiziranosti na razini međumemorije za slanje paketa (***TX***) na višem sloju mreže, a opet specifično za svako pojedino mrežno sučelje (tzv. ***txqueuelen***), koje možemo vidjeti s naredbom (kao `qlen` broj):

```
ip link show eth0
```

To je posebice izraženo za ***TAP*** mrežna sučelja koja se u virtualizaciji preko ***QEMU/KVM*** hipervizora koriste kao veza s mrežnom karticom unutar virtualnog računala. Dakle u slučajevima kada je procesor hipervizora (fizičkog računala) na kojem se pokreće virtualno računalo, sâmo povremeno preopterećen, te ne stiže obraditi sve pristigle mrežne pakete prema virtualnom računalu, može doći do odbacivanja mrežnih paketa. Tada povećanje ove (***TX***) međumemorije (prema virtualnom računalu) na ***TAP*** mrežnom sučelju, može ublažiti ovakva vršna zagušenja sustava, koja mogu uzrokovati odbacivanje paketa (*packet drop*), ali će unijeti dodatno kašnjenje (latenciju).


Pogledajte primjere u poglavlju: **27.1.2. Rad s virtualizacijom (KVM+QEMU).**

Nadalje, moguće je da su greške uzrokovane neoptimiziranim radom sloja mreže koji je zadužen za raspodjeljivanje mrežnih paketa što je vidljivo u datoteci `/proc/net/softnet_stat`***** koja sadrži statistike koje nam ovdje mogu pomoći.


Vezano za softverske signale prekida pogledajte primjere u poglavljima: **10.7.2.3.6. Naredba *mpstat* za CPU.**
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

Ono na što trebamo obratiti pažnju u ovoj statistici***** su: *drugi stupac* (***dropped***) odnosno broj odbačenih mrežnih paketa, koji su odbačeni jer ih sustav nije mogao odnosno nije stigao obraditi. Stanje se može poboljšati povećanjem međumemorije u koju se oni spremaju: `net.core.netdev_max_backlog`, sve dok broj odbačenih paketa (***dropped***) ne dođe na nulu.
Zatim imamo i *treći stupac* statistike***** (***time_squeeze***) koji se povećava, ako su radnje tijekom prihvata paketa (**RX**) prekinute jer im je istekao vremenski okvir za obradu, što je moguće poboljšati povećavanjem vrijednosti: `net.core.netdev_budget`, sve dok se ne smanji broj odbačenih paketa. I na kraju imamo statistiku***** u *devetom stupcu* (***cpu_collision***) koja govori koliko puta je došlo do kolizije tijekom pristupa mrežnom sučelju zbog operacija slanja paketa na mrežu (***TX***).

**Zatim uzroci problema mogu biti razne druge optimizacije, mehanizmi ili hardverski ubrzane značajke, poput**:

- ***RSS*** (*Receive Side Scaling*) i ***Multiqueue*** funkcionalnost/mehanizam. Odnosno, ako imamo mrežnu karticu koja podržava ***RSS*** odnosno ***Multiqueue*** funkcionalnost; naša mrežna kartica se dijeli na više logičkih pseudo hardverskih mrežnih kartica od kojih svaku od njih (i njen *IRQ*) odrađuje druga jezgra procesora. To možemo vidjeti u statistici s hardverskim signalima prekida (*IRQ*) u datoteci `/proc/interrupts`. Odnosno podršku za ovu značajku možemo vidjeti (za `eth0` mrežnu karticu) sa sljedećom naredbom:

```
ethtool -l eth0
```


Pogledajte poglavlje: **25.2.2. Multiqueue funkcionalnost.**

- ***RPS***, ***RFS*** i ***XPS*** mehanizmi - ako koristimo mehanizam poput ***RPS*** (softverske implementacije **RSS**-a), u datoteci u kojoj se zapisuju statistike rada mrežnih sučelja: `/proc/net/softnet_stat` se u *desetom stupcu* vide statistike vezane za **RPS**. Dakle u *desetom stupcu* zapravo vidimo koliko puta su CPU jezgre zatražile preko **IPI** signala prekida, signal za prebacivanje obrade ***RPS***-a na drugu CPU jezgru. Što veći broj označava veću obradu odnosno veće prebacivanje s jedne na drugu CPU jezgru, što može usporiti sustav (ili mrežu). Vršna aktivnost zbog ***RPS***-a će uzrokovati i veće vrijednosti brojača u statistici: `/proc/softirqs` i to u retku: `NET_RX`.
  Ostale detalje oko ***RFS*** i ***XPS*** pogledajte u gore navedenom poglavlju kao i za **aRFS**, koji možemo provjeriti sa:

```
ethtool -k eth0 | grep ntuple
```


Pogledajte poglavlje: **25.2.3. Receive packet steering (RPS), Receive flow steering (RFS) i Transmit Packet Steering (XPS)**.

- ***LRO i GRO***. ***LRO*** se nekada naziva i ***TPA***, a povećava propusnost dolaznog prometa, ako tu značajku podržava mrežna kartica. Kako provjeriti imamo li podršku za nju (***LRO/TPA***):

```
ethtool -k eth0 | grep large-receive-offload
```

Odnosno statistike njene upotrebe i rada možemo vidjeti sa:

```
ethtool -S eth0 | grep lro
```

Ili ako je ona označena kao ***TPA***, tada ćemo njene statistike moći vidjeti s naredbom:

```
ethtool -S eth0 | grep tpa_aggregat
```


Pogledajte poglavlje: **25.5.1. Large receive offload (*LRO*) i Generic receive offload (*GRO*)** **(pogledajte napomene za *Wireshark*).**

- ***LSO, TSO i GSO*** mehanizama, opisanih u poglavlju: **25.5.2.** Pogledajte napomene za *Wireshark*.
- ***Checksum offload*** mehanizma, opisan u poglavlju: **25.5.3.** Pogledajte napomene za *Wireshark*.
- ***Scatter-gather*** mehanizma, opisanog u poglavlju: **25.5.4.**
- Kao i svih drugih akceleriranih funkcionalnosti (značajki) mrežnih kartica, koje možemo vidjeti sa:

```
ethtool -k eth0
```

Naime sve one (akcelerirane funkcionalnosti) eventualno mogu prouzročiti probleme u mreži, a naročito ako se koriste unutar virtualnih računala, na kojima sve dodatno ovisi o *Hipervizoru*, preko kojega se one emuliraju ili propuštaju do virtualnog računala i njegovog mrežnog sučelja (mrežne kartice).

Sljedeće na redu, kao potencijalni uzrok problema na mreži su postavke brzine, *duplex* i drugih parametara rada mrežne kartice; pogledajte poglavlje: **19.6.1.** i povezana poglavlja, kao i kontrola protoka na **OSI** sloju dva (poglavlje:**20.5.**) te sva povezana mrežna sučelja na OSI sloju dva.


Pogledajte poglavlje**: 20.6. Mrežna sučelja na OSI sloju 2**.

Zatim slijede eventualni problemi na OSI sloju dva koje čine razna mrežna sučelja na ovom sloju kao i ***ARP*** protokolu odnosno eventualno (ne)osvježavanju *ARP* tablice i postavkama mehanizama čišćenja iste.


Pogledajte poglavlja:
**20.6. Mrežna sučelja na OSI sloju 2.**
**20.7.1.2. Rad s ARP protokolom.**

Na još višem sloju mreže, a to je u ovom slučaju ***IP** sloj **(OSI 3)***, moguće je uslijed fragmentacije paketa, da se pojavljuju problemi kojima uzrok može biti u (ne)fragmentaciji mrežnih paketa. Prvi problem ovdje može biti veza između mreža s različitim ***MTU*** veličinama te sâmim time fragmentacija, koja je normalna, ali će uzrokovati veće zauzeće procesora zbog obrade fragmenata, na uređaju/računalu koji povezuje takve mreže. Međutim u nekim mrežama može biti zabranjeno fragmentiranje mrežnih paketa, pa će mrežni paketi s većom ***MTU*** vrijednosti od one koju dopušta bilo koja prolazna mreža, mrežni uređaj ili računalo, biti odbačeni umjesto da se fragmentiraju (razlome).
Stoga treba pažljivo prilagoditi ***MTU*** vrijednost na svojim mrežnim sučeljima. Važno je znati da u TCP/IP komunikaciji određeni komunikacijski kanal može biti ostvaren tako da se od trenutka uspostave veze (pr. *TCP SYN*, *TCP SYN ACK*, *TCP ACK*) te dalje u komunikaciji, postavi takozvani „***don't fragment bit***" s kojim se za tu komunikaciju zabranjuje fragmentiranje mrežnih paketa. Standardni primjeri ovakvog rada su *HTTP*, *HTTPS* i neki drugi protokoli.

Slijede i drugi eventualni problemi s ***MTU*** postavkama; odnosno s velikim mrežnim okvirima (*Jumbo*).



Pogledajte poglavlja:
**23.4.2. Maximum Transmission Unit (*MTU*).**
**23.4.3. Veliki mrežni okviri (*Jumbo frames*).**



***U svakom slučaju, MTU postavke moraju biti usklađene na svim prolaznim mrežnim sučeljima i uređajima.***

Na još višim slojevima mreže, a poglavito na ***TCP*** sloju (***OSI 4***), možemo imati mnoge ***TCP*** parametre, vidljive u cijelom poglavlju o ***TCP***-u (poglavlje: **24.2. TCP**), a poglavito:

- Problemi s dostupnim maksimalnim brojem upotrebljivih ***TCP*** (***SCTP*** ili ***UDP***) portova, čiji opseg se definira u *sysctl* varijabli: `net.ipv4.ip_local_port_range`.



Pogledajte poglavlje: **19.8.1. TCP/UDP Portovi.**

- Opcije vezane za ponovno slanje paketa (Engl. *Retry*) u slučaju čestih i konstantnih gubitaka paketa na mreži, kako je opisano:



Pogledajte poglavlje: **24.2.2. Kako se uspostavlja TCP veza**.

- Opcije vezane za sigurnosne aspekte uspostavljanja veze su opcije `net.ipv4.tcp_syncookies` i `net.ipv4.tcp_max_syn_backlog`.



Ove opcije su opisane u poglavlju: **24.2.3. Sigurnosni aspekt uspostavljanja TCP veze (*SYN*).**

- Opcija za skaliranje TCP prozora koje znatno utječu na propusnost ***WAN*** veza.



Pogledajte poglavlje: **24.2.8.1. Veza između latencije, propusnosti i TCP Window size parametra**.

- Odabira TCP algoritma za nadzor zagušenja (definiran u varijabli: `net.ipv4.tcp_congestion_control`), koji također utječe na propusnost mreže.



Pogledajte poglavlje: **24.2.9. Nadzor zagušenja (*Congestion control*).**

- *Timera* odnosno trajanja TCP veze (poglavlje: **24.2.4. Trajanje TCP veze**), ali i stanja veze; poglavlje: **24.2.16.1** kao i drugih opcija.



Vezano za uzroke **odbacivanja** TCP veza pogledajte poglavlje: **24.2.17. Uzroci odbacivanja TCP veza.**

Ono što također može uzrokovati probleme u radu mreže su problemi s protokolima za redundanciju na OSI sloju tri (protokoli ***HSRP/VRRP***).



Pogledajte poglavlje:
**23.4.5. Redundancija na OSI sloju tri (OSI 3) i *VRRP* protokol.**

## Dostupni programi i alati za praćenje stanja mreže i mrežnih paketa

Sa standardno dostupnim naredbama, statistike mrežnih paketa možemo vidjeti pokretanjem neke od naredbi: `netstat -i` ili `netstat -s` ili `nstat`, ali i s naredbom `ip`; primjerice za sučelje **`eth0`** bi to bilo:

```
ip -s -s link ls dev eth0
```



Pogledajte i primjere upotrebe naredbe **`netstat`** (pogotovo primjer **2.**) u poglavlju: **25.7.4. *Naredba netstat*.**

Za prikupljanje mrežnih statistika za sva mrežna sučelja u trajanju od deset (10) puta po jednu (1) sekundu, možete pokrenuti:

```
sar -n DEV 1 10
```

Osim ili uz `DEV` (*device*) moguće je koristiti i naprednu statistiku mrežnog sučelja pomoću `EDEV`(*enh. device*) ili filtere prema protokolima, poput: `IP`, `EIP`(*enh. IP*), `ICMP`, `EICMP`(*enh.IP*), `TCP`, `ETCP`(*enh.TCP*), `UDP`, `IP6`, `ICMP6`, ili `SOCK`(*socket*) `SOCK6`(IPv6 *socket*). Dodatni parametri se ovdje mogu odvajati zarezom (`,`).

← Ovdje se prikazuje trenutno opterećenje mreže →

Pokrenimo prikupljanje napredne statistike mrežnih paketa na svim mrežnim sučeljima u trajanju od deset (10) puta po jednu (1) sekundu (izrezali smo sâmo jedan mali dio):

```
sar -n EDEV 1 10
```

```
07:38:36  AM  IFACE  rxerr/s  txerr/s  coll/s  rxdrop/s  txdrop/s  txcarr/s  rxfram/s  rxfifo/s  txfifo/s
17:38:37 AM  eno1  0.00     0.00    0.00   0.00      0.00      0.00       0.00       0.00      0.00
17:38:37 AM  eno2  0.00     0.00    0.00   0.00      0.00      0.00       0.00       0.00      0.00
```
Pogledajmo opise nekih od polja vidljivih tijekom ispisa naredbe **sar -n EDEV 1 10**:

- U prvom stupcu je vrijeme prikupljanja statistike (pr. `07:38:36`)
- `IFACE` - označava na kojem mrežnom sučelju se radi statistika (pr. `eno1`).
- `rxerr/s` - označava ukupan broj loših paketa zaprimljenih (***RX***) u sekundi.
- `txerr/s` - označava ukupan broj loših paketa poslanih (***TX***) u sekundi.
- `coll/s` - označava ukupan broj kolizija tijekom slanja (***TX***) paketa u sekundi.
- `rxdrop/s` - označava ukupan broj zaprimljenih paketa koji su odbačeni zbog prepunjene međumemorije (*buffers*).
- `txdrop/s` - označava ukupan broj paketa u slanju koji su odbačeni zbog prepunjene međumemorije (*buffers*).
- `txcarr/s` - označava ukupan broj paketa koji imaju greške tipa „*carrier-errors*", u slanju paketa. Uzrok može biti krivi odabir *Duplex* načina rada ili greške u mediju (kabel, optika).
- `rxcarr/s` - označava ukupan broj paketa koji imaju greške tipa „*carrier-errors*", u primanju paketa. Uzrok može biti krivi odabir *Duplex* načina rada ili greške u mediju (kabel, optika).
- `rxfram/s` - označava ukupan broj paketa s greškom „*frame alignment error*" tijekom primanja. Uzrok može biti da mrežni okvir (paket) nema pâran broj okteta ili ***CRC*** greške, zbog grešaka na mediju (kabel, optika) ili u odašiljačkom dijelu sklopa mrežnog sučelja onoga koji šalje ove pakete.
- `rxfifo/s` - označava ukupan broj paketa koji imaju grešku „*FIFO overrun*" tijekom primanja. Uzrok tome može biti da strana od koje primamo pakete, šalje iste prevelikom brzinom ili ih mi ne stižemo obraditi (CPU) na vrijeme pa se *FIFO* međumemorija prepuni prije nego kernel/CPU stigne obraditi te mrežne pakete.
- `txfifo/s` - označava ukupan broj paketa koji imaju grešku „*FIFO overrun*" tijekom slanja. Uzrok je sličan kao u prethodnoj statistici.

Međutim da smo koristili samo `sar -n DEV` dobili bi statistike koje su isto zanimljive (opisat ćemo njih sâmo nekoliko):

- `rxpck/s` - označava ukupan broj zaprimljenih paketa u sekundi **(što je korisno za praćenje broja paketa na mrežnom sučelju)**.
- `txpck/s` - označava ukupan broj poslanih paketa u sekundi **(što je korisno za praćenje broja paketa na mrežnom sučelju)**.
- `rxkB/s` - označava ukupnu količinu kilobajta zaprimljenih u sekundi.
- `txkB/s` - označava ukupnu količinu kilobajta poslanih u sekundi.
- `rxmcst/s` - označava ukupan broj zaprimljenih ***multicast*** paketa u sekundi.

Moguće je i čitati statistike iz već postojeće ***SAR*** datoteke (koja je već snimljena), pa filtrirati sâmo mrežne statistike (`DEV`), poput:
```
sar -f /var/log/sa/sa01 -n DEV
```

Važno je znati i kako se sve mrežne statistike stalno zapisuju u datoteku: `/proc/net/dev`
```
cat /proc/net/dev
```

Ako želimo, možemo vidjeti i osnovne statistike za mrežna sučelja sa sljedećom naredbom:
```
ip -s link
```
Dobit ćemo nešto poput sljedećeg ispisa (fokusirali smo se na ispis za mrežno sučelje `eth0`):
```
eth0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond0 state UP
    link/ether d8:9d:67:62:1d:e0 brd ff:ff:ff:ff:ff:ff
    RX: bytes       packets     errors   dropped overrun mcast
    169155055796    505207543   0        0       0       30812993
    TX: bytes       packets     errors   dropped carrier collsns
    173110866044    575253943   0        0       0       0
```
Za još detaljnije statistike (primjerice za mrežno sučelje `eth0`) mogli bismo koristiti i sljedeću naredbu:
```
ip -s -s link show dev eth0
```
Objasnit ćemo osnovne brojače, koji su isti za primanje (***RX***) i slanje (***TX***):

- `bytes` i `packets` – ovo su količine bajta odnosno paketa koji su zaprimljeni; ako je u pitanju ***RX***, ili koji su poslani, ako je u pitanju ***TX***.
- `errors` – ovo su paketi s greškama koje mogu biti uzrokovane raznim uzrocima, poput loše dogovorene brzine (10/100/1000/10000 Mbps) ili *duplex* načina rada (*full/half*), grešaka na kabelu i slično.
- `dropped` – ovo je broj mrežnih paketa koji je odbačen; ili od strane lokalnog *vatrozida* (*firewalla*) ili zbog nekih drugih problema poput prepunjene neke od međumemorija; na mrežnoj kartici, neke od softverskih međumemorija ili zbog toga jer je sustav bio preopterećen pa nije uspio obraditi pakete na vrijeme (pogledajte neke od uzroka navedenih u tekstu na početku poglavlja).
- `overrun` – ovo je broj koliko puta je mrežnom sučelju međumemorija prepunjena prije nego je kernel/CPU stigao obraditi mrežne pakete.
- `mcast` – ovo je broj poslanih *multicast* paketa; što ne označava greške već samo statistiku.
- `carrier` (za slanje ***TX***) – označava greške uzrokovane ili krivim odabirom *duplex* načina rada ili greškama na kabelu ili zbog grešaka na preklopniku na koji smo spojeni.
- `collsns` je broj kolizija koje su se dogodile, a koje smiju postojati samo ako koristimo *half duplex* način komunikacije. U *full duplex* načinu rada u pravilu, ako smo spojeni na preklopnik, ne bi smjelo biti ovakvih grešaka.


**Možemo vidjeti i detaljnije statistike na najnižoj razini pojedinog mrežnog sučelja (mrežne kartice)!**

Ako se radi o mrežnoj kartici `eth0` tada pokrenite naredbu `ethtool` na sljedeći način, ako to podržava upravljački program mrežne kartice. Ovdje možemo gledati statistike na OSI 1 i OSI 2 slojevima mreže odnosno statistike sâmog mrežnog sučelja:

```
ethtool -S eth0
```



*Statistike na višoj razini se zapisuju u datoteke unutar direktorija:* `/sys/class/net/eth0/statistics/`*, a na još višoj razini u datoteke:* `/proc/net/dev`, `/proc/net/netstat` te `/proc/net/snmp` *iz kojih ih čitaju i mnoge naredbe poput:* `netstat`, `nstat`, `ip` *i drugih.*

Ako upravljački program mrežne kartice ne podržava statistike na ovoj najnižoj razini, to ćemo vidjeti sa(skratili smo ispis na jedan redak):

```
ethtool -i eth0

supports-statistics: no
```

Ipak, u većini slučajeva ćemo imati ove statistike pa ćemo objasniti neke od njih, koje dolaze u nekoliko kategorija.
Ove brojače (statistike) na razini mrežnog sučelja dobivamo, ako pokrenemo sljedeću naredbu:

```
ethtool -S eth0
```

- Brojači unutar RX/TX prstena (*ring*-a) za određeni nîz odnosno (*queue*), pri tome *[N]* predstavlja *queue/nîz*:
  - [*N*] `rx_bytes` - predstavlja broj bajta koji je zaprimljen unutar ovog nîza.
  - [*N*] `rx_ucast_packets` - predstavlja broj *unicast* paketa koji su zaprimljeni unutar ovog nîza.
  - [*N*] `rx_mcast_packets` - predstavlja broj *multicast* paketa koji su zaprimljeni unutar ovog nîza.
  - [*N*] `rx_bcast_packets` - predstavlja broj *broadcast* paketa koji su zaprimljeni unutar ovog nîza.
  - [*N*] `rx_discards` - predstavlja broj zaprimljenih paketa unutar ovog niza koji su odbačeni zbog nedostatka međumemorije.
  - [*N*] `rx_csum_offload_errors` - predstavlja broj paketa koji su zaprimljeni unutar ovog nîza [*N*], ali su potom odbačeni jer ih je *rx checksum offload* mehanizam nakon provjere odbacio, pošto imaju neispravan provjerni zbroj *(checksum)*.
  - [*N*] `tpa_aggregations` ili `rx_lro_bytes` - predstavlja broj *agregacija (povezanih podataka)* ili bajta unutar ovog nîza, koje su povezane pomoću ***LRO***/***TPA*** mehanizma.
  - [*N*] `tpa_aggregated_frames` ili `rx_lro_packets` - predstavlja broj paketa koji su zaprimljen unutar ovog nîza, a spojeni (povezani) su pomoću ***LRO/TPA*** mehanizma.
  - Iste statistike imamo i za TX (slanje), a one počinju s: `tx_*`.

---

- Brojači na razini cijelog Linuxa; za ***RX*** – prijem(**slični su i za slanje odnosno za TX**):
  - `rx_bytes` - predstavlja ukupan broj bajta koji su zaprimljeni na mreži
  - `rx_error_bytes` - predstavlja ukupan broj bajta koji su zaprimljeni na mrežna sučelja i imaju greške.
  - `rx_ucast_packets` - predstavlja broj *unicast* paketa koji su zaprimljeni.
  - `rx_mcast_packets` - predstavlja broj *multicast* paketa koji su zaprimljeni.
  - `rx_bcast_packets` - predstavlja broj *broadcast* paketa koji su zaprimljeni.
  - `rx_crc_errors` ili `rx_crc_errors_phy` - predstavlja ukupan broj paketa koji su zaprimljeni na mreži, ali nisu prošli provjeru integriteta odnosno imaju grešku u ***FCS*** (*Frame Check Sequence*) polju na OSI sloju dva (OSI 2). Uzrok ovoga može biti takozvana *MAC frame CRC* greška (greška izračuna *CRC/FCS* na razini mrežnog okvira). U ovom slučaju uzrok je pošiljatelj koji nije dobro izračunao **FCS** zaglavlje na mrežnom okviru. Uzrok tome može biti problematična mrežna kartica pošiljatelja, loše kabliranje, kolizija (za *half duplex* mreže), problematičan upravljački program i slično.
  - `rx_align_error` - predstavlja greške koje se događaju na najnižem OSI sloju jedan (OSI 1) kada dolazi do pogrešaka koje se događaju na samoj „*žici*“ odnosno električnim signalima tijekom perioda slanja paketa na drugu stranu (našu). Uzrok može biti kolizija (kod *half duplex* mreža), loše kabliranje, ili neispravna mrežna kartica odnosno detektiran je nepâran broj okteta u paketu ili ***CRC*** greške.
  - `rx_undersize_packets` ili `rx_length_errors` - predstavlja greške koje se događaju kada je detektiran mrežni okvir koji je premale veličine (***undersize***) i koji će biti odbačen. Uzrok može biti druga strana koja odašilje manje mrežne pakete nego konkretna mreža dozvoljava, ali je vjerojatnija neka greška na razini mrežne kartice odašiljača ili problem s usmjerivačem (*routerom*).
  - `rx_oversize_packets` ili `rx_length_errors` - predstavlja greške koje se događaju kada je detektiran mrežni okvir koji je prevelike veličine (***oversize***) a koji će biti odbačen. Uzrok može biti druga strana koja odašilje veće mrežne pakete nego konkretna mreža dozvoljava, ali je vjerojatnija neka greška na razini mrežne kartice odašiljača ili problem s usmjerivačem (*routerom*).
  - `rx_framents` - predstavlja broj primljenih paketa koji su odbačeni zbog duljine kraće od 64 bajta i imaju **FCS** pogrešku na fizičkom mrežnom sučelju. Ako se ovaj brojač povećava, vjerojatni uzrok je taj da pošiljatelj paketa ima krivo konfiguriran ***MTU*** ili je došlo do kolizije (kod *half duplex* mreža).
  - `rx_jabbers` - predstavlja broj primljenih paketa koji su odbačeni jer su veći od standardno definiranog ***MTUa*** (obično 1500 bajta) i obično uz to imaju i krivi ***CRC***. Ako se ovaj brojač povećava, vjerojatni uzrok je taj da pošiljatelj paketa ima krivo konfiguriran ***MTU***.
  - `rx_discards` - predstavlja broj primljenih paketa koji su odbačeni zbog nekih drugih razloga ili višeg protokola. Uzrok mogu biti krivo konfigurirani VLAN-ovi na strani preklopnika i/ili lokalnoj strani.
  - `rx_filtered_packets` - predstavlja broj primljenih paketa koji su odbačen zbog toga što im dijelovi zaglavlja na OSI sloju dva (*OSI* 2) ne odgovaraju odnosno kada je MAC adresa navedena kao odredišna

(za ovu mrežnu karticu) neka potpuno druga MAC adresa što znači kako nam mrežni paket nije namijenjen, pa će paket normalno biti odbačen, što je normalno u radu (ako ovaj broj nije prevelik).

- `rx_brb_discard` - (*BRB=on chip buffer*) predstavlja mrežne pakete koji su došli do *međumemorije* (*buffera)* mrežne kartice i počeli su prepunjavati ovu međumemoriju jer ih sustav nije bio u stanju preuzeti i obraditi dovoljno brzo. Ovo indicira da je sustav bio preopterećen ili u slučaju konstantno velikog broja paketa ovdje, da sustav nije dobro optimiziran ili da mu je CPU preslab za prihvat i obradu paketa ovom brzinom.
- `rx_brb_truncate` - (*BRB=on chip buffer*) predstavlja slično stanje kao i prethodno odnosno predstavlja mrežne pakete koji su došli do *međumemorije* (*buffera)* mrežne kartice i počeli su prepunjavati ovu međumemoriju jer ih sustav nije bio u stanju preuzeti i obraditi dovoljno brzo. Ovo indicira da je sustav bio preopterećen ili u slučaju konstantno velikog broja paketa ovdje, da sustav nije dobro optimiziran ili da mu je CPU preslab za prihvat paketa ovom brzinom.
- `rx_pause_frames` - predstavlja broj primljenih *pause* mrežnih okvira.

 Pogledajte poglavlje: **20.5. Kontrola protoka (*flow control*) na OSI sloju 2.**

- `rx_csum_offload_errors` - predstavlja greške koje se pojavljuju, ako imamo uključenu hardversku akceleraciju vidljivu kao: `rx-checksumming: on`, pri kojoj mrežna kartica odrađuje provjeru integriteta zaprimljenih mrežnih okvira na OSI sloju dva (OSI 2) odnosno *RX checksum*. Stoga su mrežni okviri koji se ovdje broje zapravo mrežni paketi kojima je tijekom primitka, nakon što je ponovno izračunat *provjerni* zbroj (*cheksum*) odnosno provjera integriteta mrežnog okvira/paketa, otkrivena greška.

Ova greška znači kako je mrežni okvir neispravan; a to može značiti kako je neispravno poslan od strane koja ga je poslala (pr. neispravna mrežna kartica ili upravljački program za nju) ili od prolaznog uređaja kroz koji je paket prošao. Moguće je da je došlo i do neke druge greške na mreži, prema odredištu, čak i na OSI sloju jedan (OSI 1) odnosno na razini signala.

---

- Brojači na razini cijelog Linuxa; za ***TX*** – slanje, koji su specifični za slanje paketa su sljedeći:
  - `tx_carrier_errors` - ovo su greške koje mogu biti uzrokovane greškama u kabliranju ili u *duplex* načinu rada.
  - `tx_single_collisions` - ovo je broj kolizija koje su detektirane kod slanja paketa, uzrok može biti rad u *half duplex* načinu rada.
  - `tx_multi_collisions` - ovo su greške koje se isto pojavljuju u *half duplex* načinu rada, kada je detektirano nekoliko kolizija iako su paketi poslani prema *CSMA/CD*.
  - `tx_deferred` - ovo je brojač koji govori koliki broj puta je mrežno sučelje pokušalo poslati paket na mrežu, ali je prvi pokušaj slanja pomoću *Carrier Sense* mehanizma zaustavljen. Ovo vrijedi za *half duplex* mreže odnosno način rada mrežnog sučelja.
  - `tx_excess_collisions` - ovo je brojač grešaka koje se događaju kada se 16 puta za redom detektira kolizija na mreži, za *half duplex* način rada, što znači kako je mreža prezagušena.
  - `tx_late_collisions` - ovo su greške koje se također događaju kod *half duplex* načina rada, i to su detektirane kolizije koje su detektirane zakašnjelo na paketima koji su veći od 64 bajta, odnosno detektirane su tek kada se pročitalo prvih 64 bajta paketa. Najčešće se ovo događa kod malih mrežnih paketa u *half duplex* načinu rada.
  - `tx_total_collisions` - ovo je suma svih *collision* grešaka koje smo već naveli gore.

Za više detalja o parametrima rada (na sloju veze [*OSI 2*]) primjerice `eth0` sučelja, možete pokrenuti naredbu:

```
ip -d link show eth0

eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group
default qlen 1000
link/ether 02:cc:bc:05:c7:f6 brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 46 maxmtu
3712 addrgenmode none numtxqueues 8 numrxqueues 8 gso_max_size 65536 gso_max_segs
65535
```

Ovdje osim svega već navedenog (**MTU** veličine, statusa sučelja, ***txqlen*** [`default qlen`], …) vidimo i:

- `qdisc mq` – odabrani *Queueing Discipline* mehanizam je `mq`.
- `addrgenmode none` – označava standardnu dodjelu MAC adrese sučelja odnosno da nije ručno mijenjana.
- `numtxqueues 8` i `numrxqueues 8` – označavaju broj TX i RX nizova za slanje/primanje paketa.
- `gso_max_size 65536` i `gso_max_segs 65535` – definiraju maksimalnu veličinu **GSO** (*Generic Segmentation Offload*) segmenta te maksimalan broj segmenata koje sučelje može generirati prilikom segmentacije paketa s **GSO**.

---

Za statistike na višim slojevima (OSI 3 i 4), možemo koristiti naredbe: `netstat` i `nstat` ili `ss`.

Sve napredne TCP, UDP, IP, ICMP ili *socket* (i *buffer*) statistike za sva mrežna sučelja možemo dobiti s naredbama:

```
nstat -az
```

ili

```
netstat -s
```

 Za nastavak analize proučite poglavlja koja slijede, a za najvišu razinu mrežnog stôga proučite i transportne protokole (**UDP** i **TCP**) te sve opcije i parametre vezane za njih (specifične za pojedini mehanizam transportnih protokola)**!**

**Izvori informacija: (495),(496),(497),(498),(499),(1039),** **`man ip link`**, **`man 5 proc`**, **`man sar`**, **`man ethtool`**, **`man nstat`**, **`man netstat`**, **`man ss`**, ***poglavlje: 23.2.1.1.*. Pogledajte i plakat P5.**

---

## 25.4. Koje još mrežne tehnologije i protokoli postoje ?

Ako sâmo pogledamo najnovije generacije poslužiteljskih ***Intel, Broadcom, Mellanox*** i drugih naprednih mrežnih kartica koje osim što su u stanju odrađivati sve više funkcionalnosti bez posredovanja procesora (CPU), opremljene su i raznim dodatnim naprednim tehnologijama, od kojih ćemo navesti sâmo neke:

- ***Intel® Bord Direct I/O*** – pomoću ove tehnologije (*Intel*) moguća je direktna komunikacija mrežne kartice s međumemorijom procesora (CPU) bez prolaska kroz sistemsku memoriju. Ovom tehnologijom se dodatno smanjuje latencija (kašnjenje) i povećava propusnost (i naravno potrošnja el. energije).

- ***TCP/IP checksum offload*** tehnologija je s kojom mrežna kartica sâma izračunava TCP/IP provjerni zbroj (*checksum*) za svaki TCP/IP mrežni paket te tako rasterećuje *CPU.*

 Pogledajte poglavlje**: 25.5.3. Checksum offload.**

- ***Large send offload** (LSO) i **TCP segmentation offload** (TSO)* tehnologije. U normalnom radu *CPU* podatke koji se trebaju poslati na mrežu tj. prvo prema mrežnoj kartici razlama na male segmente, veličine koje je definirana kao **MTU** (Engl. *Maximum Transmission Unit*), a koji se onda pretvaraju u mrežne pakete i šalju na mrežu. Ovaj proces se zove segmentacija i odrađuje ga *CPU* na *TCP* sloju (ako je TCP odabrani transportni protokol). U slučaju kada se prema mreži mora poslati 64KB podataka *CPU* ih prvo mora razlomiti na manje dijelove, primjerice veličine 1500 *bajta (**MTU** veličine)*. Dakle *CPU* mora prema mrežnoj kartici slati: 64 KB = 65.536 *bajta* / 1500 bajta = 44 segmenta, jedan po jedan te ih mrežna kartica tako i šalje na mrežu (kako su došli). Upotrebom ***LSO*** odnosno ***TSO***, *CPU* ne mora sâm segmentirati (razlamati) podatke u male dijelove, već može odjednom poslati svih 64KB prema mrežnoj kartici, koja ih sama razlama (segmentira) te šalje na mrežu.

 Pogledajte poglavlje: **25.5.2. LSO, TSO i GSO.**

- ***Large Receive Offload** (LRO)* tehnologija radi slično poput ***LSO***, ali u drugom smjeru. Dakle nîz paketa koji je zaprimljen na mrežnu karticu se pohranjuje u međumemoriji na mrežnoj kartici te se svi ti segmenti odjednom šalju višim slojevima *TCP/IP*-a na obradu prema *CPU*, a ne jedan po jedan što bi bio slučaj bez upotrebe ove tehnologije.

 Pogledajte poglavlje: **25.5.1. Large receive offload (*LRO*) i Generic receive offload (*GRO*).**

- ***TCP Offload Engine** (TOE)* koji nudi dodatna rasterećenja *CPU*a jer praktično mrežna kartica preuzima gotovo svu obradu nad mrežnim segmentima/paketima.

- ***Quick Assist*** (integrirana i u neke modele CPUa) *(Intel)* tehnologija koja hardverski ubrzava operacije poput:
    - Kriptiranja i dekriptiranja (AES, DES, 3DES, ARC4)
    - *Hash* operacija (SHA-1, MD5, SHA-2, SHA-224, SHA-256, SHA-384, SHA-512)
    - Generiranja slučajnih brojeva (*random number generation*) te komprimiranja i dekomprimiranja (DEFLATE, LZV), ali i ostale kriptografske funkcije: RSA, Diffie-Hellman, ECDSA i ECDH, …

 Pogledajte poglavlje: **26.4.4.2. Upotreba hardverski ubrzanih kripto funkcija.**

- ***Interrupt moderation (coalescing)*** - grupira više mrežnih paketa prije nego se kreira signal prekida.

 Pogledajte poglavlje: **10.5.2.2. *Interrupt moderation*.**

- ***Virtual Machine Device Queues** (VMDq)* *(Intel)* u kombinaciji sa *Single-Root I/O Virtualization (**SR-IOV**)* koji ubrzava rad u virtualizaciji, poglavito u slučajevima kada koristite usmjerivač ili vatrozid kao virtualno računalo.

- ***Scatter-gather*** – kod mrežne komunikacije omogućava sustavu da izvodi DMA ulazno/izlazne operacije na više memorijskih međuspremnika istovremeno.

 Pogledajte poglavlje: **25.5.4. *Scatter-gather*.**

- ***VLAN** offload* – hardverski ubrzan rad sa ***VLAN*** paketima.

 Pogledajte poglavlje: **20.6.2. VLANovi odnosno virtualne lokalne mreže.**

Dalje u tekstu proći ćemo osnovnu konfiguraciju nekih od navedenih naprednih mehanizama i tehnologija dostupnih u mrežnim karticama.

**Izvori informacija:** (23),(24),(25),(55),(55.1)**. Pogledajte i plakat** P5**.**

# 25.5. Hardverski ubrzane mrežne funkcionalnosti

U ovoj cjelini opisat ćemo neke od mrežnih funkcionalnosti koje mogu biti ubrzane (akcelerirane) od strane pojedinih (naprednijih) mrežnih kartica te stoga znatno rasteretiti CPU.

## 25.5.1. Large receive offload (*LRO*) i Generic receive offload (*GRO*)

### Large receive offload (*LRO*)

*LRO* je tehnika za povećanje propusnosti za dolazni mrežni promet, smanjivanjem potrebe za kontaktiranjem procesora (CPU-a) kod zaprimanja svakog pojedinog mrežnog paketa. *LRO* djeluje skupljanjem višestrukih dolaznih paketa iz jednog tôka, u veći međuspremnik prije nego što se svi oni proslijede na više mrežne slojeve. Čime se smanjuje broj paketa koje treba obraditi. Linux implementacije općenito koriste *LRO* zajedno s novim API-jem (*NAPI*) kako bi dodatno smanjili i broj signala prekida (***IRQ***), koji bi se događao tijekom svakog zaprimljenog mrežnog paketa. Prema nekim mjerenjima čak i primjena ove tehnologije u potpunosti implementirane u softveru može znatno povećati performanse mrežnog sustava i mreže. Naime ovaj mehanizam može biti implementiran u upravljačkom programu ili u hardveru; a čak i *LRO* emulacija u upravljačkom programu ima prednosti.
LRO je široko podržan od strane 10Gbps ili bržih mrežnih kartica pod Linuxom. *LRO* ne bi trebao raditi na strojevima koji rade kao usmjerivači, jer može praviti probleme u komunikaciju između krajnjih točaka komunikacije (*end-to-end communication*).

***LRO*** se nekada naziva i ***TPA*** (*Transparent Packet Aggregation*) ubrzanje odnosno akceleracija.



*Napomena za upotrebu **LRO**:* Ako postoje važne razlike između zaglavlja dolaznih i odlaznih paketa (u obradi), one će se izgubiti. Naime, ako paket ima neke opcije ili zastavice u zaglavlju one mogu biti izgubljene u ovom procesu.
To znači kako određeni protokoli neće raditi, pogotovo, ako sustav služi kao usmjerivač.



*Za funkcionalnost usmjeravanja je važno da se ne smije mijenjati zaglavlje paketa u prolazu odnosno da i ako se mijenja, da je mehanizam koji ih mijenja sposoban to odraditi i sa svim opcijama i parametrima u zaglavlju ili ovisno o protokolu, što **LRO** ponekad i ne odrađuje najbolje. Međutim sve ovisi o implementaciji **LRO** mehanizma, ali i ovisno o mrežnoj kartici te njenom pripadajućem upravljačkom programu.*

**Kako uključiti LRO**
Prvo provjerimo podržava li ***LRO*** naša mrežna kartica (*eth0*) i u kojem je trenutnom stanju, pomoću naredbe `ethtool`:

```
ethtool -k eth0 | grep large-receive-offload
```

Sve dok imamo sljedeći odgovor:

```
large-receive-offload: off
```

Odnosno dok nemamo `off [fixed]` sve je u redu jer imamo (hardversku) podršku za ***LRO***. Uključiti ga možemo sa:

```
ethtool -K eth0 lro on
```

Naravno uz napomenu kako se neki protokoli ne ponašaju najbolje s uključenim ***LRO***.
Kako pratiti statistike obrade mrežnih paketa koji su vezani za ***LRO***, pogledajmo upotrebom sljedeće naredbe:

```
ethtool -S eth0 | grep lro
```

Dobit ćemo nešto poput sljedećeg ispisa:

```
     rx_lro_packets: 0
     rx_lro_bytes: 0
     rx0_lro_packets: 0
     rx0_lro_bytes: 0
     rx1_lro_packets: 0
     rx1_lro_bytes: 0
```

U slučaju kada mrežna kartica ima hardverski podržan ***LRO***, a ovisno o konkretnoj mrežnoj kartici (konkretno *za Broadcom NetXtreme II*: 10 Gbps LAN: `bnx2x`) ***LRO*** statistike su vidljive pod nazivom ***TPA*** **(***Transparent Packet Aggregation).*
Tada ih možemo vidjeti sa sljedećom naredbom:

```
ethtool -S eth0 | grep tpa_aggregat
```

Pa ćemo tada dobiti nešto poput:
```
[0]: tpa_aggregations: 27288
[0]: tpa_aggregated_frames: 531667
[1]: tpa_aggregations: 9232511
[1]: tpa_aggregated_frames: 18790860
     tpa_aggregations: 9353343
     tpa_aggregated_frames: 21200446
```
Dakle ovdje vidimo koliko je mrežnih okvira agregirano (spojeno), prema RX kanalu: `[0]` i `[1]` te ukupno (zadnja dva reda).

**Kako privremeno isključiti LRO**
```
ethtool -K eth0 lro off
```
Za mrežne kartice poput navedene *Broadcom NetXtreme II*: 10 Gbps LAN; koja koristi: `bnx2x` kernel modul, u nekim slučajevima **(1040)** se preporuča da se *LRO* (odnosno *TPA* kako ga naziva *Broadcom*) u potpunosti isključi na razini hardvera mrežne kartice. Za navedenu mrežnu karticu to možemo riješiti permanentno (trajno) dodavanjem dolje navedenog retka u datoteku `/etc/modprobe.d/bnx2x.conf` te restartom računala.
```
options bnx2x disable_tpa=1
```


Ako gledate mrežne pakete s programima ***wireshark/tcpdump***, a **LRO/TSO** su uključeni, moguće je da ćete vidjeti mrežne pakete veće od **MTU-a.** To u ovom slučaju (***LRO***) nije problem jer to nije realan pokazatelj pošto ovi programi pakete dohvaćaju na niskoj razini mreže. *Umjesto programa* ***Wireshark*** *možemo koristiti i program* ***`tcpdump`*** (**Poglavlje:25.7.8**).

## Generic receive offload (*GRO*)

*Generic Receive Offloading* (*GRO*) je praktično softverska implementacija *LRO* funkcionalnosti uz još neke optimizacije.
Rješenje za potencijalno problematično ponašanje *LRO* je upotreba ***GRO*** (*Generic receive offload*) mehanizma koji je također dostupan u Linuxu. Naime u *GRO*-u su kriteriji prema kojem se mogu skupljati i povezivati paketi, uvelike ograničeni; ***MAC*** zaglavlje mora biti identično, a sâmo nekoliko TCP ili IP zaglavlja može se razlikovati. Zapravo skup zaglavlja koji se razlikuju je prilično ograničen: provjerni zbrojevi (*checksumovi*) su nužno različiti, a polje unutar IP zaglavlja: **ID** dopušteno je povećavati. Čak i *TCP timestamps* oznake moraju biti identične. Kao rezultat ovih ograničenja, spojeni paketi mogu se *resegmentirati* bez nekih potencijalnih grešaka u komunikaciji. Još jedna zanimljivost kod *GRO* je ta, da za razliku od *LRO*, on nije ograničen sâmo na TCP/IPv4 protokol. Možemo reći kako je *GRO* pouzdaniji od većine implementacija *LRO*-a, zbog namjerno uvedenih ograničenja, kako ne bi došlo do problema. Pogledajmo je li *GRO* uključen za našu `eth0` mrežnu karticu:
```
ethtool -k eth0 |grep generic-receive-offload

generic-receive-offload: on
```
Vidimo kako je uključen (`on`), a ako bi bilo potrebno uključiti ga kada je isključen, to bi napravili sa sljedećom naredbom:
```
ethtool -K eth0 gro on
```


***GRO*** je obično standardno uključen u Linuxu!

Ako ga ipak želimo isključiti, što u nekim slučajevima može biti korisno (u slučaju problema u radu), to možemo napraviti sa:
```
ethtool -K eth0 gro off
```


Veličina **GRO** (i **TSO**) memorije je standardno ograničena na 64kB (65536 bajta), prvenstveno zbog ograničenja takozvane ***SKB*** (*socket*) međumemorije koja se koristi, kako za slanje, tako i za primanje mrežnih paketa. To postaje problem na mrežnim sučeljima propusnosti 10Gbps i više. Za više detalja o **SKB** međumemoriji pogledajte sliku 52 na stranici **359** odnosno za sve tehničke detalje i opise, pogledajte poveznicu: https://www.kernel.org/doc/html/latest/networking/skbuff.html.

Međutim od Linux kernela **5.19** uvedena je podrška za takozvani ***Big TCP***, koji za **IPv6** proširuje **SKB** međumemoriju.
Pogledajmo kako za mrežno sučelje **eth0** povećati **GRO** međumemoriju (za IPv6) na 185.000 bajta:
```
ip link set dev eth0 gro_ipv6_max_size 185000
```

***Big TCP*** je od Linux kernela 5.19 podržan i za druga mrežna sučelja i mehanizme; primjerice za `tcp_cubic` mehanizam te za sučelja: ***ipvlan***, ***bonding***, ***macvlan, loopback*** i druge.
Od kernela **6.3.** ***Big TCP*** je uveden i za **IPv4,** te se može konfigurirati:
```
ip link set dev eth0 gro_ipv4_max_size 128000
```

**Izvori informacija za LRO i GRO: (473),(474),(475),(500),(1040),(1295),(1450),** **`man ethtool`. Pogledajte i plakat P5.**

## 25.5.2. LSO, TSO i GSO

***Large Send Offload*** (*LSO*) je tehnika za povećanje propusnosti za odlazni promet smanjivanjem kontaktiranja centralnog procesora (CPU) tijekom slanja svakog pojedinog mrežnog paketa. Ona djeluje tako što skuplja više mrežnih paketa s viših mrežnih (OSI) slojeva u jedan veći međuspremnik koji se onda šalje na mrežnu karticu. Mrežna kartica zatim sâma razlama taj međuspremnik u zasebne pakete. Ova tehnika kada se primjenjuje na TCP se naziva ***TCP segmentation offload*** (*TSO*) odnosno generički gledano mehanizam koji nije specifičan samo za TCP protokol se naziva ***Generic segmentation offload*** (*GSO*).



Ako gledate mrežne pakete s programima **wireshark ili tcpdump** (25.7.8), a **LSO/TSO/GSO** su uključeni, možete vidjeti mrežne pakete veće od **MTU-a** što inače nije normalno odnosno značilo bi da je došlo do nekakve greške.

S mrežnim sučeljem (mrežnom karticom) koje podržava ***TSO***, kernel može pripremiti mnogo veće pakete od standardnih 1500 bajta, do maksimalno 64 KB (65.536 bajta) za odlazne podatke, a mrežna kartica će zatim sâma ponovno segmentirati podatke u manje pakete koji će se poslati na mrežu (žicu). Ovakav mehanizam smanjuje opterećenje kernela za faktor od 40. Naime inače u ovom procesu TCP sloj šalje nižim slojevima mreže odnosno upravljačkom programu mrežne kartice pakete čija veličina je maksimalno definirana kao ***MTU*** veličina (obično **1500** bajta), pa tada imamo znatno veći broj paketa. Upotrebom ove tehnologije TCP može slati znatno veće pakete podataka prema upravljačkom programu mrežne kartice, a tada hardver mrežne kartice bez posredovanja CPU-a te velike pakete razlama na veličinu definiranu *MTU*-om i šalje ih na mrežni medij (žicu/optičko vlakno). Kako smo već spomenuli, kernel zapravo ima i generički mehanizam praćenja segmentacije nazvan ***GSO***, koji nije ograničen sâmo na TCP protokol. U praksi se događa da se performanse poboljšavaju čak i ako se ova značajka emulira u upravljačkom programu mrežne kartice.
Pogledajmo kako uključiti **TSO** za `eth0` mrežnu karticu, upotrebom naredbe `ethtool` na sljedeći način:

```
ethtool -K eth0 tso on
```

Pogledajmo i što smo sada dobili:

```
ethtool -k eth0 | grep -i tcp
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp6-segmentation: on
```

Dakle vidimo globalnu vrijednost (`tcp-segmentation-offload`) uključenu (`on`). Isto tako vidimo kako je **TSO** uključen i za TCP v.4 (`tx-tcp-segmentation`) i TCP v.6. (`tx-tcp6-segmentation`). Sada je potrebno napraviti mjerenje performansi sustava, i vidjeti je li se mreža ubrzala. **Ako ipak želimo isključiti *TSO*, trebamo napraviti sljedeće:**

```
ethtool -K eth0 tso off
```

---

## Generic segmentation offload (GSO)

Ako želimo uključiti ***GSO*** upotrebom naredbe `ethtool`, to možemo učiniti na sljedeći način:

```
ethtool -K eth0 gso on
```

Pogledajmo stanje koje imamo sada:

```
ethtool -k eno1 | grep -i generic-segmentation-offload
generic-segmentation-offload: on
```

U drugom slučaju kada ipak želimo isključiti ***GRO***, to možemo napraviti sa:

```
ethtool -K eth0 gso off
```

Veličinu memorije za ***GSO*** ne bi trebalo mijenjati, ali je i to moguće. Primjerice ako ga za `eth0` želimo smanjiti na 32.768, sa:

```
ip link set eth0 gso_max_size 32768
```



Preporuka je da ne treba koristiti i **TSO** i **GSO** istovremeno. U nekim slučajevima se ***GSO*** ponaša optimalnije (iako nije implementiran u hardveru). Međutim u virtualizaciji[(504),(505)] se može dogoditi da i ***TSO*** i ***GSO*** uzrokuju probleme u radu virtualnih računala, pa je tada preporuka da se i ***TSO*** i ***GSO*** isključe prvenstveno na razini *hipervizora* i to sâmo, ako imate nekih mrežnih problema s virtualnim računalima koji bi upućivali na *TSO* ili *GSO*. Drugi primjeri ipak upućuju na pozitivan učinak upotrebe ***TSO*** ili ***GSO*** u virtualizaciji[(858)] naročito na kernelima novijim od inačice 3.x.

**Pogledajte napomenu za veličinu GRO međumemorije (SKB) i mrežna sučelja na brzinama 10Gbps (i više)!**

Pogledajmo kako (*od linux kernela 5.19*) za mrežno sučelje **eth0** povećati **GSO** međumemoriju (za IPv6.) na 185.000 bajta:

```
ip link set dev eth0 gso_ipv6_max_size 185000
```

Od kernela **6.3.** ***Big TCP*** je uveden i za **IPv4,** te se može konfigurirati (povećati) ovako:

```
ip link set dev eth0 gso_ipv4_max_size 128000
```

**Izvori informacija: (501),(502),(503),(504),(505),(858),(1295), (1450), `man ethtool`,`man ip link`.**
**Pogledajte i plakat P5.**

---

## 25.5.3. Checksum offload

Tehnologija naziva ***checksum offload*** je tehnologija s kojom mrežna kartica sama izračunava TCP/UDP ili IP provjerni zbroj (*checksum*) za svaki TCP/IP mrežni paket te tako rasterećuje *CPU* koji bi bez ove *akceleracije* sâm morao izračunavati provjerne zbrojeve. Ne zaboravimo kako se TCP (ili UDP i IP) provjerni zbroj mora izračunati za svaki pojedini poslani mrežni paket, ali se mora i ponovno izračunati i za svaki pojedini zaprimljeni paket, kako bi se novo izračunati provjerni zbroj usporedio s onim koji se nalazi u zaprimljenom paketu, na osnovu čega se može zaključiti je li primljeni paket ispravan.

Pogledajmo kako uključiti ovu funkcionalnost na našoj `eth0` mrežnoj kartici i to za RX (primanje) i za TX(slanje), upotrebom naredbe `ethtool` na sljedeći način:

```
ethtool -K eth0 rx on tx on
```

Sada provjerimo što smo promijenili:

```
ethtool -k eth0 | grep -i check

rx-checksumming: on
tx-checksumming: on          (tx-checksum-ipv4: on)    (tx-checksum-ipv6: on)
```

**Ako ipak *checksum offload* iz nekog razloga želimo isključiti, to možemo napraviti sa:**

```
ethtool -K eth0 rx off tx off
```



U načelu je dobro uključiti i `rx` i `tx` *checksumming* jer ubrzavaju proces izračuna TCP *checksuma* pošto se on tada odrađuje unutar hardvera mrežne kartice (ako ona to podržava). Međutim valja biti svjestan kako postoje i neki rijetki slučajevi u kojima je ova funkcionalnost loše izvedena odnosno ima greške u implementaciji.

Postoje i slučajevi u kojima mrežni paket prolazi kroz više slojeva, poput primjene u virtualizaciji, u kojoj imamo veliki broj mrežnih slojeva, poput:

*virtualno računalo (LAN) → virtualni bridge → virtualni switch → LAN kartica hipervizora → preklopnik (switch) → **mreža***



U virtualizaciji stoga imamo cijel nîz slojeva mrežnih elemenata, od kojih mnogi od njih mogu i moraju raditi razne operacije na svakom mrežnom paketu, a koje mogu uključivati i ponovni izračun TCP *checksuma*.

Ako tome nadodamo i mogućnost *tuneliranja* (pr. **GRE** ili **VxLAN**) stvar postaje još kompleksnija, pa tada imamo sljedeću povezanost:

*virtualno računalo (LAN) → virtualni bridge → virtualni switch→ GRE/VxLAN tunel→ LAN kartica hipervizora → preklopnik (switch) → **mreža***



*U ovo treba uračunati i drugu stranu koja prima ili šalje mrežne pakete, a koja može biti jednako kompleksna. Stoga imate li nekih mrežnih problema s virtualnim računalima, koji bi upućivali na **`rx` i `tx` checksumming**, najbolje je da ga isključite*[(509),(510)] *i to potencijalno na razini hipervizora, i/ili na drugim razinama, primjerice unutar virtualnog računala (VM).*



***TCP Checksum* offload** mehanizam može utjecati i na pogrešan prikaz mrežnih paketa, ako ih gledamo s programima poput **`wireshark`** ili **`tcpdump`** (25.7.8) jer je moguće da ćemo imati poruku „***cksum incorrect***" zbog toga što se ovdje TCP *checksum* izračunava na mrežnoj kartici a **`wireshark`** ili **`tcpdump`** će prikazati onaj koji je ponovno izračunat od strane kernela, koji je tada krivo izračunat te prikazan u navedenim programima, iako se ne koristi.

Dakle ignorirajte ove greške u navedenim programima, ako imate uključenu ovu funkcionalnost.

**Izvori informacija: (506),(507),(508),(509),(510),(516),(858), `man ethtool`, Pogledajte i plakat P5.**

### 25.5.4. *Scatter-gather*

U računalnim mrežama vektorski ulaz/izlaz (Engl. *Vectored I/O*) također je poznat kao *Scatter/Gather* I/O. To je metoda ulaza i izlaza podataka u kojoj se jednim pozivom sekvencijalno čitaju podaci iz više memorijskih međuspremnika, a potom zapisuju u jedan nîz ili tok podataka. S druge strane je to metoda u kojoj se čita podatke iz toka podataka, a zapisuje ih se na više memorijskih međuspremnika, kako je definirano u vektoru međuspremnika. *Scatter/gather* se pri tome odnosi na proces prikupljanja podataka iz, ili raspršivanje podataka u, dani skup međuspremnika. Vektorski I/O može raditi sinkronizirano (sinkrono) ili asinkrono. Glavni razlozi za korištenje ove metode rada je očita učinkovitost i praktičnost.

Pomoću *Scatter/gather* odnosno raspršivanja/sakupljanja sustavu se dopušta da izvodi DMA ulazno/izlazne (I/O) operacije na memorijskim međuspremnicima koji su raspršeni kroz fizičku memoriju. Razmislite primjerice o slučaju velikog memorijskog međuspremnika stvorenog u korisničkom memorijskom prostoru (*user space*).
Sâma aplikacija će vidjeti ovaj kontinuirani raspon virtualnih adresa, ali fizičke stranice memorije iza tih adresa gotovo sigurno neće biti međusobno susjedne.

Međutim, ako se taj međuspremnik (njegov sadržaj) treba zapisati na neki uređaj (disk ili mrežu) u jednoj ulazno/izlaznoj operaciji (I/O operaciji), morat će se izvršiti jedna od dvije stvari:
(1) Podaci moraju biti kopirani u regije memorije koje su međusobno susjedne odnosno koje su u nîzu.
(2) Uređaj mora biti u stanju raditi s popisom fizičkih memorijskih adresa i njihovih duljina, prikupljajući podatke iz svake memorijske lokacije odnosno iz svih segmenata memorije.

*Scatter/gather I/O* eliminira potrebu za kopiranjem podataka u memorijske lokacije u nîzu, pa tek potom dohvaćanjem istih.
On može uvelike povećati učinkovitost istodobnim dohvaćanjem potrebnih, ali raspršenih (*Scatter* dio procesa) memorijskih lokacija grupirajući odnosno ulančavajući ih zajedno (*Gather* dio procesa).
Važno je napomenuti kako i hardver i upravljački program moraju podržavati ovu funkcionalnost.

Pogledajmo trenutno stanje naše mrežne kartice upotrebom naredbe `ethtool`:

```
ethtool -k eth0 | grep -i gather
scatter-gather: on
        tx-scatter-gather: on
```

Vidimo kako je *Scatter/gather* uključen (`on`). Pogledajmo kako ga ručno isključiti za `eth0` mrežnu karticu:

```
ethtool -K etho sg off
```

Odnosno pogledajmo i kako ga prema potrebi uključiti:

```
ethtool -K etho sg on
```



Za druge hardverski ubrzane funkcije, pogledajte poglavlje:
**26.4.4.2. Upotreba hardverski ubrzanih kripto funkcija.**



**Mala povijest razvoja mrežnog stôga Linuxa i drugih mrežnih tehnologija u Linuxu.**

**U prvim implementacijama mrežnog stôga Linuxa SKB (socket međumemorija) mogla je sadržavati samo jedan linearni mrežni segment (jedan MSS segment prema TCP terminologiji).**

**Zatim je dodan Scatter Gather (SG) kako bi se moglo podijeliti (i dohvaćati) zaglavlje odnosno podatke u više područja memorije.**

**Proizvođači mrežnih kartica počeli su podržavati TSO i LRO**

**Dodan je i GSO/GRO kao softverska (rezervna) optimizacija.**

**Pri tome su GSO i GRO dizajnirani s praktičnim ograničenjima dostupnih mrežnih kartica i maksimalne duljine IPv4 paketa (64kB) jer se oslanjaju na (IPv4 *tot_len*) polje koje je 16 bitno ($2^{16}$=65535). Ista ograničenja su i za IPv6. Poboljšanje je uvedeno za IPv6 u RFC2675 koje je za IPv6 moguće koristiti od kernela 5.19. kao tzv. Big TCP.**

**Od kernela 6.3 uveden je takozvani Big TCP i za IPv4 (uz IPv6) koji za GRO i GSO omogućuje znatno povećanje međumemorije, što može drastično ubrzati mrežnu obradu, posebno za brzine od 10 Gbps na više!**

**Izvori informacija: (511),(512),(513),(514),(515),(1302),(K-14), `man ethtool`, RFC 2675. Pogledajte i plakat P5.**

### 25.5.5. Pregled navedenih dostupnih tehnika, tehnologija i protokola

Pregled navedenih dostupnih tehnika, tehnologija i protokola koje možemo optimirati možemo vidjeti sumirano na slici 235.1.

**Slika 235.1. (P1) Kratki pregled svih dosada navedenih tehnologija i protokola kao i mogućnosti optimizacije.**

VM
Aplikacije
Mrežni stog
Virtual NIC
QEMU EMULATOR
TAP [tap0]
Linux Bridge [br0]
Sistemski pozivi (*System calls*) [SCI] i *NetLink* pozivi
Aplikacije (Programi)
SKB
RMEM
Linux Socket
WMEM
Socket buffer
Linux Socket
Mrežni stôg Linuxa
UDP
TCP
Window scaling (skaliranje TCP prozora)
Congestion Control (algoritmi za nadzor zagušenja)
Timers (za zatvaranje i/ili istek TCP veze)
TCP Synqueue & Limits
GSO
IP Mogućnost usmjeravanja (IP Forwarding), Non local IP,Local port range (per IP), ..
RFS
RPS
GRO
(*Optmem*) međumemorija za metapodatke
[nazivi mrežnih sučelja, zaglavlja, ...]
(RX) Backlog buffers
NetLink
netfilter
Netfilter (vatrozid)
BRIDGE
VLAN, VxLAN, VETH, MACVLAN, ...
*Bonding/teaming (agregacija sučelja)*
*TC (Traffic control)*
Network scheduler međumemorija
Network scheduler (qdisc)
TX queue
TXqueuelen buffer
Upravljački program → primjerice: ixgbe → [eth0]
*NAPI* (Ovisno o upravljačkom programu)
*Niži sloj upravljačkog programa*
IRQ
*XDP (ako je omogućen)*
*eXpress Data path (ako je podržan)*
*(FIFO) Ring buffers*
Prstenaste memorije
RX
TX
TCP/UDP/IP Checksum
LRO
TSO
RSS
aRFS
VLAN, VxLAN, GRE, ESP, TLS, MACSEC, Traffic Control (TC), ...
Ethernet Checksum
Mrežna kartica (pr. Intel® i82599)

Na slici 235.1. pokušali smo spomenuti sve najosnovnije postavke, parametre i vrijednosti na koje bi trebali obratiti pažnju kod praćenja ili optimizacije sustava. Što se tiče ***txquelen*** međumemorije ona je namijenjena za slanje, ali je važno s koje strane promatrate „slanje" (TX) paketa. Ako recimo govorimo o **`TAP`** mrežnom sučelju na koje se spaja virtualno računalo, tada je ova (TX) međumemorija namijenjena za slanje paketa od fizičkog računala prema mreži virtualnog računala. Pogledajte i tablicu s vezom između OSI i TCP/IP slojeva te mrežnih protokola prema slojevima kao i posebnih (logičkih i virtualnih) mrežnih sučelja u Linuxu, na tim slojevima, koji ih također koriste.

| TCP/IP model | Mrežni protokoli na njemu | OSI model | Dodatna mrežna sučelja u Linuxu koja ih mogu koristiti |
|---|---|---|---|
| Aplikacijski sloj | **DNS, FTP, HTTP, HTTPS, LDAP, NTP, SMTP, POP3, IMAP, SSH,TELNET, SNMP, DAP, MQTT, NNTP,RTP, RIP, SIP, BGP, XMPP.** *NetBIOS, PAP, SMB, RPC* **…** | **Slojevi 7,6 i 5** | |
| Transportni sloj | **TCP, UDP, SCTP** *AH, ESP, NetBIOS, iSCSI***…** | **Sloj 4** | **`VXLAN`** |
| Internet sloj | **IP, ICMP, IGMP,** *VRRP,RIP,OSPF,IPSEC,ARP***…** | **Sloj 3** | **`TUN,IPVLAN`** |
| Sloj mreže | **MAC** (za *Ethernet* mreže) **ARP**, **RARP**, **NDP**, *STP,LACP,PPP,PPTP,PAP,L2TP…* | **Sloj 2 i Sloj 1** | **`TAP,VETH,BRIDGE,BOND, VLAN(802.1Q),MACVLAN,IPVLAN, MACVTAP,MACSEC,VCAN,VXCAN`** |

**Pogledajmo i usporednu tablicu s Linux naredbama i slojevima mreže na kojima rade (prema slici 235.1.).**

| Sloj mreže | Naredbe, sysctl varijable ili posebne datoteke | |
|---|---|---|
| **Network socket i mrežni stôg** | Network *socket* (***SKB***): `netstat -tunap` i `ss -anm` (*stupci **Recv-Q** i **Send-Q***) ***Recv-Q*** *i* ***Send-Q*** se mogu povećavati sa: `net.core.rmem_max` i `net.core.wmem_max`. Statistike (mrežni stôg) su vidljive u datotekama: `/proc/softirqs`,`/proc/net/softnet_stat`, `/proc/net/dev`,`/proc/net/netstat` i `/proc/net/snmp` | |
| | Statistike se mogu dobiti i s naredbama: `netstat -i`, `netstat -s` i `ss` te: `nstat` i `sar -n DEV` | |
| **TCP/UDP** | TCP i UDP međumemorije se mogu povećati sa: `net.ipv4.tcp_mem` i `net.ipv4.udp_mem`. | |
| **Niži sloj Linux stôga** | Za ***RPS*** [za `eth0`]: `/sys/class/net/eth0/queues/` | |
| | Za ***XPS*** [za `eth0`]: `/sys/class/net/eth0/queues/tx-XX/xps_cpus` | |
| | Za ***GRO*** (za uključivanje) [za `eth0`]: `ethtool -K eth0 gro on` | |
| | Za ***GSO*** (za uključivanje) [za `eth0`]: `ethtool -K eth0 gso on` | |
| **Network Scheduler** | Odabrani network **scheduler** se postavlja u ***sysctl*** varijabli: `net.core.default_qdisc` | |
| **Među memorije** | Međumemorija za opcije: `net.core.optmem_max` | |
| | **Backlog** međumemorija za sva mrežna sučelja: `net.core.netdev_max_backlog` | |
| **Upravljački program** | **TX queue** međumemorija mrežnog sučelja [za `eth0`]: `ip link set eth0 txqueuelen XX` | |
| | Niži sloj upravljačkog programa (za mrežne statistike): | `ip -s link` te `sar -n EDEV` kao i `ethtool -S eth0` |
| | Statistike direktno iz datoteka: | `/proc/net/dev` ili `/proc/net/netstat` ili `proc/net/snmp` |
| **Mrežno sučelje** | Prstenasta međumemorija (povećanje za TX i RX): `ethtool -G eth0 rx YY tx ZZ` | |
| | **RSS** značajka (za uključivanje) [za `eth0`]: `ethtool -L eth0 rx XX tx YY` | |
| | **aRFS** značajka (za uključivanje) [za `eth0`]: `ethtool -K eth0 ntuple on` | |
| | **LRO** značajka (za uključivanje) [za `eth0`]: `ethtool -K eth0 lro on` | |
| | **TSO** značajka (za uključivanje) [za `eth0`]: `ethtool -K eth0 tso on` | |
| | **Checksum offload** značajka (za uključivanje) [za `eth0`]: `ethtool -K eth0 rx on tx on` | |
| | **Scatter-gather** značajka (za uključivanje) [za `eth0`]: `ethtool -K eth0 sg on` | |
| | **Hardverska Podrška za VLAN** (za uključivanje) [za `eth0`]: `ethtool -K eth0 rxvlan on txvlan on` | |
| | **Postavka mrežne kartica na 1Gbps (1000 Mbps), Full duplex** [za `eth0`]**:** `ethtool -s eth0 speed 1000 duplex full` | |

## 25.5.6. Optimizacije mrežnih sučelja propusnosti veće od 10Gbps

**Kako radi mrežna kartica**
Mrežni paket u dolasku pohranjuje se u prstenastu memoriju (engl. *ring buffer*) mrežne kartice pomoću *Direct Memory Access* (**DMA**) metode pristupa i to bez intervencije centralnog procesora (CPU). Prstenasti spremnik je vrsta memorije koja omogućuje efikasan i brz pristup podacima. Tada dolazi do kreiranja hardverskog signala prekida (**IRQ**) koji nalaže procesoru da treba privremeno prekinuti trenutni zadatak (program) te pod hitno pozvati rutinu za obradu signala prekida (interrupt handler). Ova rutina je dio upravljačkog programa (kernel modula/drivera) mrežne kartice.
Nakon što se hardverski signal prekida potvrdi, sve ovisi o mrežnoj kartici i njenom upravljačkom programu:

5. **Za starije kartice i upravljački program**: CPU jezgra preuzima mrežni paket i prosljeđuje ga mrežnom stôgu, a potom prazni *ring buffer* međumemoriju te se vraća izvršavanju prathodnog zadatka (programa ili programskoj nîti).
6. **Za novije kartice i upravljački program**: CPU jezgra koja je reagirala na hardverski signal prekida aktivira softverski signal prekida i vraća se svom prethodnom zadatku. Međutim tada, u pozadini, Linux kernel pokreće poseban kernel proces `kSoftirqd` koji u ovom slučaju dohvača i šalje pakete mrežnom stôgu za daljnju obradu te prazni *ring buffer*. ***KSoftirqd*** je specifičan i po tome što se pokreće na svakoj CPU jezgri odnosno obrađuje softverske signale prekida na svakoj CPU jezgri jer se softverski signali prekida znatno koriste i za druge namjene. Važno je znati da su softverski signali prekida drastično brži od hardverskih signala prekida.
7. **Za najnovije kartice i posebni (NAPI) upravljački program**: CPU jezgra koja je reagirala na hardverski signal prekida aktivira softverski signal prekida i vraća se svom prethodnom zadatku. Međutim ovaj softverski signal prekida aktivira takozvani *Pooling* mehanizam (*NAPI*) upravljačkog programa (ako on to podržava). Ovakav (*NAPI)* upravljački program tada isključuje hardverske signale prekida pa se od tog trenutka procesor (CPU) brine o provjeri pristiglih paketa: njihovom preuzimanju i pražnjenju *ring buffer* međumemorije, što je najbrža moguća metoda rada, čija jedina mana jest veće iskorištenje procesora, uz ekstremno veću mrežnu propusnost.
8. **Pooling način rada bez podrške za NAPI u upravljačkom programu:** i ovakav način rada se koristi u Linuxu, a prema performansama je negdje između **2.** i **3.** metode rada.



Za detalje pogledajte poglavlja:
**10.5.1. IRQ (*Interrupt request*).**
**10.5.1.3. Servis *ksoftirqd i softverski signali prekida*.**
**10.5.2.1.1. Kako to (NAPI) radi, još malo detaljnije.**

**Postavke BIOS-a**
U slučaju primjene u virtualizaciji Intel® Virtualization Technology for Directed I/O (Intel® *VT-d*) treba biti omogućen u BIOS-u, Intel preporučuje da **IOMMU** bude u prolaznom načinu rada za optimalnu izvedbu mreže.
To se postiže dodavanjem dvije preporučene opcije u kernel, što ćemo napraviti na sljedeći način:

```
grubby --update-kernel=ALL --args=intel_iommu=on
grubby --update-kernel=ALL --args=iommu=pt
```



Za detalje pogledajte poglavlje:
**10.7.1.4. Ulazno-izlazni memorijski kontroler (IOMMU).**

**Pravilan odabir sabirnice u koju spajamo (ugrađujemo) PCI/PCIe mrežnu karticu**
Za mrežne kartice koje zbog svoje propusnosti zahtijevaju veću propusnost sabirnice i primjerice zahtjevaju PCIe x16 utore i to može biti izazov. Naime neki fizički PCIe x16 utori konfigurirani su električno kao x8 utori. Ovi utori nemaju dovoljnu propusnost za punu izvedbu sa 100Gbps ili četverostrukim mrežnim priključkom od 25Gbps.
Srećom upravljački program može detektirati ovu situaciju te ispisati poruku poput:

```
63.008 Gb/s available PCIe bandwidth, limited by 8 GT/s x8 link at 0000:xx:xx.0
(capable of 252.048 Gb/s with 16 GT/s x16 link
```

Dakle prije ugradnje mrežne kartice potrebno je provjeriti specifikacije PCIe utora u koji ju ugrađujemo.



Za detalje o sabirnici pogledajte poglavlje:
**10.2. (Re)konfiguracija sabirnice i uređaja na njoj.**

**Bond (Team) mrežna sučelja**
Linux *bonding* napredna je i vrlo moćna značajka koja drastično poboljšava performanse mreže, a osim toga omogućava redundanciju i toleranciju na pogreške. Međutim konfiguracija *bonding*-a često zahtijeva kompatibilni mrežni hardver (mrežni preklopnik), kao i ispravnu konfiguraciju na strani poslužitelja, kako bi ispravno funkcionirala.



Vezano za bonding pogledajte poglavlje:
**20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).**

**Optimizacije hardverskih signala prekida (IRQ)**
Jedan od mehanizama vezanih za hardverske signale prekida jest prilagodljivo moderiranje signala prekoida (*Interrupt moderation*) koje je često uključeno prema zadanim postavkama. Ono je osmišljeno za postizanje niske iskorištenosti procesora i visokih performansi mreže. Iako zadane prilagodljive stope moderiranja dobro funkcioniraju u većini uvjeta, za neka radna opterećenja ručno drefiniranje ovih postavki može pokazati bolje performanse. Dakle moguće je u potpunosti isključiti *Interrupt moderation* mehanizam na sljedeći način (pr. za `eth0` mrežno sučelje):

```
ethtool -C eth0 adaptive-rx off adaptive-tx off
```

Odnosno prema potrebi ga možemo ponovno uključiti sa:

```
ethtool -C ethX adaptive-rx on adaptive-tx on
```

Kada se onemogućuje adaptivno moderiranje signala prekida, mora se navesti vrijednost za brzine prijema i slanja.
Ove vrijednosti mogu biti u rasponu od 0-236 mikrosekundi (0=bez ograničenja), pružajući efektivni raspon od 4.237 do 250.000 signala prekida u sekundi. Dobar početak općeg podešavanja je 80 µs ili ~12.500 signala prekida po sekundi. Za postavljanje moderiranja prekida na fiksnu stopu prekida od 80 µs (12.500 prekida u sekundi) možemo napraviti sljedeće:

```
ethtool -C eth0 adaptive-rx off adaptive-tx off rx-usecs 80 tx-usecs 80
```

← Ovdje definirani `rx-usecs` i `tx-usecs` omogućuju da se umjesto generiranja hardverskog signala prekida za svaki pojedinačni paket, CPU obavijesti tek nakon proteka određenog vremena. To može smanjiti opterećenje CPU-a jer se generira manje signala prekida. Dakle postavljanje većih vrijednosti za *rx-usecs* i *tx-usecs* može smanjiti opterećenje CPU-a jer se generira manje signala prekida. Međutim, to može dovesti i do povećanja kašnjenja, jer paketi moraju dulje čekati u međuspremniku prije nego što se obrade. S druge strane postavljanje manjih vrijednosti za *rx-usecs* i *tx-usecs* može smanjiti kašnjenje, ali može dovesti i do većeg opterećenja CPU-a jer se generira više signala prekida. Najbolji rezultati se stoga dobivaju samo testiranjem i pronalaskom najoptimalnije vrijednosti za vaše potrebe.



Za detalje pogledajte poglavlja:
**10.5.2.2. *Interrupt moderation*.**

Preporuka je nakon ovih promjena pratiti brojače odnosno pokazatelje rada mreže.
Ove statistike na mrežnom sučelju `eth0` možete pratiti sa:

```
ethtool -S eth0 | grep rx_dropped
```

Primjerica ako se `rx_dropped` brojači povećavaju tijekom generiranja mrežnog prometa prometa, vjerojatno imate prespor CPU, nedovoljno međuspremnika (zbog veličine prstenaste međumemorije mrežne kartice) ili prenisko postavljenu stopu signala prekida prekida.



Vezano za postavljanje međuspremnika prstenaste međumemorije (`ethtool -G`), pogledajte poglavlje:
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

Nadalje, i `tx-frames-irq` i `rx-frames-irq` su dobre dodatne opcije za konfiguraciju signala prekida mrežne kartice. One određuju minimalan broj paketa koji se mora akumulirati u međuspremniku mrežne kartice prije nego što se generira signal prekida za slanje (*tx-frames-irq*) ili za prijem (*rx-frames-irq*) mrežnih paketa.

Pravilno podešavanje ovih parametara može povećati efikasnost mrežnog prometa i smanjiti latenciju.

**Slučajevi kada mrežna kartica i njen upravljački program podržavaju NAPI metodu za dohvaćanje paketa**
U slučajevima kada mrežna kartica i njen upravljački program podržavaju NAPI metodu za dohvaćanje paketa iz svoje međumemorije, moguće je ograničiti koliko će maksimalno paketa sustav moći dohvatiti u jednom NAPI ciklusu povlačenja paketa, sa svih mrežnih kartica na sustavu, koje podržavaju NAPI metodu dohvaćanja.
Ova *sysctl* opcija jest: `net.core.netdev_budget` i standardno je postavljena na vrijednost od **300** paketa. Naravno u nekim slučajevima je moguće i potrebno malo povećati ovu vrijednost (uz praćenje statistike gubitka paketa ili njihovog kašnjenja).



Vezano za postavke ovih međuspremnika također pogledajte poglavlje:
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

Na razini od upravljačkog programa prema redovima čekanja (*queuing*), moguće je definirati i maksimalni broj paketa koji se mogu spremiti u posebnu međumemoriju prema ovom redu čekanja. Standardno postavljena vrijednost ove međumemorije jest 1.000. Ona označava broj paketa koji se mogu spremiti u ovu međumemoriju u slučajevima kada je računalo (CPU) preopterećeno s obradom te kernel ne stiže obraditi mrežne pakete. Ova međumemorija je zajednička za sva mrežna sučelja, tijekom dohvaćanja mrežnih paketa (RX). Ona je vidljiva kao *sysctl* opcija: `net.core.netdev_max_backlog`.

Dakle možemo ju eventualno povećavati sa standardno postavljenih 1.000 na 3.000:
```
sysctl -w net.core.netdev_max_backlog=3000
```

**Minimalna latencija i maksimalan broj signala prekida**
U slučaju potpuno drugačijeg smjera optimizacije, ako nam je što manje latencija najvažnija, tada je moguće enerirati signal prekida za svaki pojedini mrežni paket (što će naravno nevjerojatno opteretiti CPU i vjerojatno usporiti mrežnu obradu):
```
ethtool -C eth0 adaptive-rx off adaptive-tx off rx-usecs 0 tx-usecs 0
```

**Softverski IRQ odnosno softverski signali prekida**
SoftIRQ odnosno softverski signali prekida su rutine kernela koje se primjerice izvršavaju nakon što je mrežna kartica poslala hardverski signal preikda naznačavajući da su pristigli mrežni paketi. Svrha SoftIRQ-a je tada dohvatiti mrežni paket i poslati ga prema mrežnom stôgu linuxa te pražnjenje međuspremnika (*prstenaste međumemorije*) mrežne kartice.
Softverski signali prekida se mogu vidjeti i u alatima za praćenje procesa kao što su `ps` i `top`.
Statistike vezane za Soft IRQ možemo vidjeti i s naredbom:
```
mpstat -I SCPU
```

Ili s naredbom:
```
lsirq -S
```

Kao i statistike u datoteci `/proc/net/softnet_stat`, u kojoj se vidi veza softverskih signala prekida vezanih za mrežu:
```
cat /proc/net/softnet_stat | awk '{print $2}'
```

Ono na što ovdje trebamo obratiti pažnju jest filtrirani drugi stupac (`dropped`) odnosno broj odbačenih mrežnih paketa, koji su odbačeni jer ih sustav nije stigao obraditi. Stanje se može poboljšati povećanjem međumemorije u koju se oni spremaju: `net.core.netdev_max_backlog`, sve dok broj odbačenih paketa (dropped) ne dođe na nulu.



Vezano za statistike oko softverskih signala prekida i mrežni podsustav, pogledajte i:
**25.2.3. Receive packet steering (RPS), Receive flow steering (RFS) i Transmit Packet Steering (XPS).**

**Optimizacija prstenaste memorije mrežnog sučelja**
Moguće je da pri trenutnoj brzini dolaska mrežnih paketa, kao i generiranja hardverskih signala prekida, dolazi do prepunjavanja prstenaste međumemorije unutar mrežne kartice. Naime RX prstenasti međuspremnik koristi se za pohranu dolaznih paketa dok ih upravljački program uređaja ne obradi. Upravljački program uređaja potom prazni RX prstenastu međumemoriju, obično putem softverskih signala prekida. Povećanjem veličine ovog međuspremnika može se smanjiti visoka stopa odbacivanja novo pristiglih paketa. Takvo stanje možemo vidjeti sa povečanjem brojčanika u ispisu sljedeće statistike:
```
ethtool -S eth0 | grep rx_dropped
```

Tada je prvo potrebno optimizirati rad s hardverskim signalima prekida, kako smo prethodno naveli, a tek potom raditi dodatne optimizacije po pitanju prstenste međumemorije (tz. RX i TX ring buffer).
Za provjeru trenutnog stanja prstenste međumemorije, možemo pokrenuti naredbu:
```
ethtool -g eth0
```

Odnosno prema potrebi ju možemo povećati (do hardverske granice naše mrežne kartice), primjerice na 4096:
```
ethtool -G eth0 rx 4096 tx 4096
```



Vezano za postavljanje međuspremnika prstenaste međumemorije (`ethtool -G`), pogledajte poglavlje:
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

**Odabir mehanizma za raspodjelu (queuing) mrežnih paketa**
Prilikom slanja paketa od aplikacije kroz mrežni stog Linuxa, jedna od posljednjih komponenti je tzv. *network scheduler*.
On upravlja nizom mrežnih paketa za slanje (TX), stavljajući ih u zasebne redove obrađujući ih te prosljeđujući dalje prema nižim slojevima mreže, a u konačnici i međumemoriji koja se nalazi u upravljačkom programu mrežnog sučelja (tzv. driver queue). Ovu komponentu možemo nazvati i mrežnim raspoređivačem paketa na najnižoj razini, a kojih postoji poveći broj. Naime svaki od njih implementira drugačiji algoritam za raspodjelu mrežnog prometa (paketa).
Neki od njih impleniraju napredne mehanizme, ali pri tome unose određena kašnjenja i zahtjevaju dodatnu obradu od strane procesora. Stoga, ako nam je samo brzina obrade mrežnih paketa prioritet, preporuka je koristiti neki od što jednostavnijih (što bržih).



Vezano za raspodjelu (*queuing*) mrežnih paketa, pogledajte poglavlje:
**25.1.4. Mehanizmi za raspodjelu (*queuing*) mrežnih paketa.**

**Optimizacije TX međumemorije**
Na razini ispod mehanizma za raspodjelu mrežnih paketa prema upravljačkom programu nalazi se sloj u kojem se nalazi međumemorija za slanje (TX), a koja je fiksno definirane veličine za sve pakete, a u Linuxu se naziva: txqueuelen.
Pogledajmo ovu međumemoriju koja je standardne veličine 1.000 (paketa).

```
ip link show eth0
2: eth0:<BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond0 state
UP mode DEFAULT qlen 1000 link/ether d8:9d:67:62:1d:e0 brd ff:ff:ff:ff:ff:ff
```

Sada ga postavimo odnosno povećajmo na 2.000 paketa:

```
ip link set eth0 txqueuelen 2000
```

← veća brojka znači da će se moći pohraniti više paketa prije slanja, ali to znači i uvođenje kašnjenja (latencije) stoga treba ili ostaviti zadanu vrijednost ili ju pažljivo povećavati**.**



Vezano za TX međumemoriju mrežnog sučelja (za slanje mrežnih paketa), pogledajte:
**25.1.2. Optimizacija TX međumemorije (txqueuelen) mrežnog sučelja.**

## Optimizacije CPU-a

Ako imamo samo nekoliko programskih niti koje su zadužene za mrežni promet, možda je dovoljno koristiti samo nekoliko procesorskih (CPU) jezgri. Međutim, ako su te jezgre opterećene 100%, propusnost vaše mreže vjerojatno je ograničena.
Stoga je potrebno analizirati sljedeće:

- Optimizirati IRQ moderiranje mrežnom sučelju.
  ← Pogledajte prethodno navedenu cjelinu: "**Optimizacije hardverskih signala prekida (IRQ)**"
- Optimizirati veličinu prstenaste međumemorije na mrežnom sučelju.
  ← Pogledajte prethodno navedenu cjelinu: "**Optimizacija prstenaste memorije mrežnog sučelja**"

Potrebno je razmotriti povećanje broja aplikacijskih (programskih) niti kako bismo rasporedili opterećenje CPU-a na više jezgri. Naime ako sve jezgre koje koriste vašu aplikaciju rade na 100%, vaša je aplikacija možda više ograničena CPU resursima nego mrežom.

**CPU i IP forwarding optimizacije**
Kod IP prosljeđivanja (engl. *IP forwarding*), za postizanje boljih performansi, par redova čekanja (transmit/receive queue) za prijenos/primanje trebao bi koristiti istu procesorsku jezgru te tako smanjiti bilo kakvu sinkronizaciju predmemorije između različitih jezgri procesora. Ovo se može izvesti dodjeljivanjem hardverskih signala prekida točno određenim jezgrama procesora.

IP prosljeđivanje *(IP forwarding)* se trenutno uključuje sa:

```
sysctl -w net.ipv4.ip_forward=1
```

**NUMA CPU optimizacije vezane za lokalitet mrežne kartice na sabirnici**
Svaki fizički procesor u NUMA arhitekturi računala (na računalima koj imaju dva ili više fizičkih procesora) osim što ima svoj memorijski kontroler i svoju lokalnu RAM memoriju, on također ima i pristup na svoju lokalnu sabirnicu, poput primjerice ***PCI Express*** sabirnice. Stoga već u NUMA sustavima s dva fizička procesora (dva NUMA *node-a*) imamo i nerijetko dvije odvojene ***PCI Express*** sabirnice: pri tome jse svaka spojena na svoj fizički procesor.



Vezano za ***NUMA*** arhitekturu procesora i vezu sa sabirnicom, pogledajte:
**10.7.2.2.1. NUMA I/O topologija.**

**NUMA i IRQ**
Vezano za takozvani NUMA I/O *afinitet* jasno je da bi najbolje bilo za su za hardversk isignal prekida (***IRQ***) i uređaje na koje se odnosi zadužene CPU jezgre najbliže samom uređaju odnosno sabirnici na koju je uređaj i spojen. Stoga treba paziti na IRQ affinitet prema NUMA CPU jezgrama.



Za detalje pogledajte poglavlje:
**10.5.1. IRQ (*Interrupt request*).**

**NUMA i CPU afinitet**
Ako imamo mrežno vrlo intenzivnu aplikaciju odnosno program (pr. na brzinama 10+ Gbps), a mrežna kartica koju ona koristi se nalazi na *NUMA I/0* koji je spojena na NUMA *node 0* (prvi fizički procesor), tada bi se i ta aplikacija trebala nalaziti odnosno pokretati na istom *NUMA node-u*. Točnije na bilo kojoj CPU jezgri na tom *NUMA node-u*. Očekivane performanse tada mogu porasti čak do dva (2) puta ili čak i više.

**NUMA i RAM afinitet**
Slično kao i kod **SMP** sustava, na **NUMA** sustavima je također moguće postaviti CPU afinitet. Ali ovdje se radi o CPU afinitetu koji je svjestan NUMA arhitekture odnosno pripadnosti RAM memorije određenom memorijskom kontroleru unutar konkretnog fizičkog procesora.
U primjeru koji slijedi, naš proces (program) ćemo vezati za *NUMA NODE 0* te će moći koristiti samo RAM memoriju od tog fizičkog procesora. Za to možemo napraviti sljedeće:

```
numactl --localalloc --cpunodebind=0 IME_PROGRAMA
```

Pogledajmo primjer kako pokrenuti aplikaciju za testiranje brzine mreže (`iperf -s`) koja će smjeti/moći koristiti samo CPU jezgre 0-7, koje se na našem poslužitelju nalaze na *NUMA node 0*.

```
numactl --physcpubind=0-7 iperf -s
```



Za detalje pogledajte poglavlje:
**10.7.2.2.2. NUMA CPU i RAM afinitet.**



Neki programi prilikom pokretanja već imaju ugrađenu mogućnost konfiguracije CPU jezgri na kojima se moraju izvršavati. To je primjerice slučaj u virtualizacijji, gdje je za određeno virtualno računalo moguće definirati CPU jezgre na kojima će se izvršavati. Drugi primjer su mrežne specijalizirane aplikacije.

## Multiqueue odnosno Tx/Rx redovi izvršavanja (čekanja) odnosno *TX/RX Queues*

Moderne mrežne kartice podržavaju višestruke hardverske redove odnosno nizove za izvršavanje (tzv. *Multiqueue* značajka). Svaki prijemni nîz (*receive queue*) pri tome koristi svoj hardverski signal prekida (IRQ), a svaki signal prekida se veže za točno određenu CPU jezgru. Isto je i sa svakim nizom za slanje. U svakom slučaju, bilo bi ih dobro koristiti ih maksimalno onoliko koliko imamo dostupnih CPU jezgri na računalu ili u ekstremnim slučajevima, točno onoliko koliko ih želimo iskoristiti za tu namjenu (za one jezgre procesora za koje ih želimo optimizirati). Pogledajmo našu mrežnu karticu (`etho`):

```
ethtool -l etho
Channel parameters for eth0:
Pre-set maximums:
RX:             n/a
TX:             n/a
Other:          1
Combined:       63
```

```
Current hardware settings:
RX:             n/a
TX:             n/a
Other:          1
Combined:       1
```

← Vidimo da da ona ima hardverski dostupna 63 kombinirana reda (niza) čekanja, ali da ih koristi samo jedan.
Ako naš poslužitelj ima 32 procesorske jezgre to bi u pravilu trebali povećati na 32, što ćemo postići na sljedeći način:

```
ethtool -L eth0 combined 32
```



Vezano za *multiqueue* postavke pogledajte:
**25.2.2. Multiqueue funkcionalnost.**

**Praćenje statistika mrežnog sučelja**
Za praćenje statistika mrežnog sučelja (na OSI slojevima 1 i 2), po pitanju grešaka, možemo koristiti:

```
watch -d 'ethtool -S eth0 | egrep "error|drop|discards"| egrep -v :\ 0 | column'
```

← Sve vrijednosti koje se ovdje povećavaju mogu predstavljati potencijalan problem:

- `rx_dropped` - povećanje brojača odnosno ove vrijednosti ukazuje na prespor ili preopterećen CPU ili prepunjavanje prstenaste međumemorije jer procesor ne stigne dohvatiti ili obraditi pristigle mrežne pakete.



Vezano za ove statistike pogledajte i:
**10.5.1.3. Servis *ksoftirqd i softverski signali prekida*.**
**10.5.1.3.1. Softverski signali prekida detaljnije.**
**10.7.2.3.6. Naredba *mpstat* za CPU.**
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

**Druge mrežne statistike na višim OSI slojevima**
Za praćenje statistika na višim slojevima mreže (OSI 3 i 4), za odbacivanje, probleme s međumemorijama, *socketom* i slično, možemo koristiti naredbe poput:

```
netstat -s | egrep -i "lock|socket\ buf"
```

ili

```
nstat -az
```



Za detalje ovakvih statistika na **IP** sloju, pogledajte:
**23.1. Oblik IP poruke (paketa).**

Vezano za detalje statistika na **UDP** sloju pogledajte:
**24.1.1. Oblik UDP poruke (paketa).**

Vezano za detalje oko ovakvih statistika na **TCP** sloju, pogledajte:
**24.2.3. Sigurnosni aspekt uspostavljanja TCP veze (*SYN*).**
**24.2.17. Uzroci odbacivanja TCP veza.**

Pogledajte i primjere upotrebe naredbi:
**25.7.4. *Naredba netstat*.**
**25.7.10. Naredba ethtool.**

**LRO i GRO**
LRO je tehnika za povećanje propusnosti za dolazni mrežni promet, smanjivanjem potrebe za kontaktiranjem procesora kod zaprimanja svakog pojedinog mrežnog paketa. LRO djeluje skupljanjem višestrukih dolaznih paketa iz jednog tôka, u veći međuspremnik prije nego što se svi oni proslijede na više mrežne slojeve. Čime se smanjuje broj paketa koje treba obraditi. Linux implementacije općenito koriste LRO zajedno s novim API-jem (NAPI) kako bi dodatno smanjili i broj signala prekida (IRQ), koji bi se događao tijekom svakog zaprimljenog mrežnog paketa. *Generic Receive Offloading* (**GRO**) je praktično softverska implementacija LRO funkcionalnosti. I **LRO** i **GSO** mogu utjecati na propusnost mreže.



Vezano za detalje oko **LRO** i **GRO**, pogledajte:
**25.5.1. Large receive offload (*LRO*) i Generic receive offload (*GRO*).**

**Veličina mrežnih paketa**
Zamislite na trenutak kada sustav pokušava držati korak s mrežnom karticom (sučeljem) propusnosti 100 Gbps. Naime, ako koristimo standardnu maksimalnu veličinu paketa od 1538 bajta (**MTU** 1500 bajta), korištenje ovog mrežnog sučelja punom brzinom znači nositi se s više od osam milijuna paketa u sekundi. Pri toj brzini, centralni procesor (CPU) ima oko 67 ns da učini sve što je potrebno za obradu svakog paketa, što opće nije puno vremena; primjerice samo jedan promašaj u dohvaćanju podataka iz predmemorije procesora može unijeti dovoljno kašnjenja da se vremenski okvir za obradu (67ns) probije.



Za detalje vezane za rad procesora, pogledajte poglavlje i primjere o obradi mrežnih paketa:
**10.7.1.3. CPU registri još detaljnije.**

Međutim situacija postaje bolja, ako se smanji broj paketa, a to se može postići povećanjem veličine paketa. Stoga nije iznenađujuće da visokoučinkovite mreže koriste veće veličine paketa. Uz standardnu konfiguraciju (**jumbo** paketi) moguće je koristiti i puno veće pakete, primjerice **MTU** veličine oko 9000 bajta (cca. 9kB), što značajno poboljšava situaciju.

***Svi mrežni paketi veći od 1500 bajta (MTU) se nazivaju jumbo paketi.***



Za detalje vezane za jumbo pakete, pogledajte poglavlje:
**23.4.3. Veliki mrežni okviri (*Jumbo frames*).**

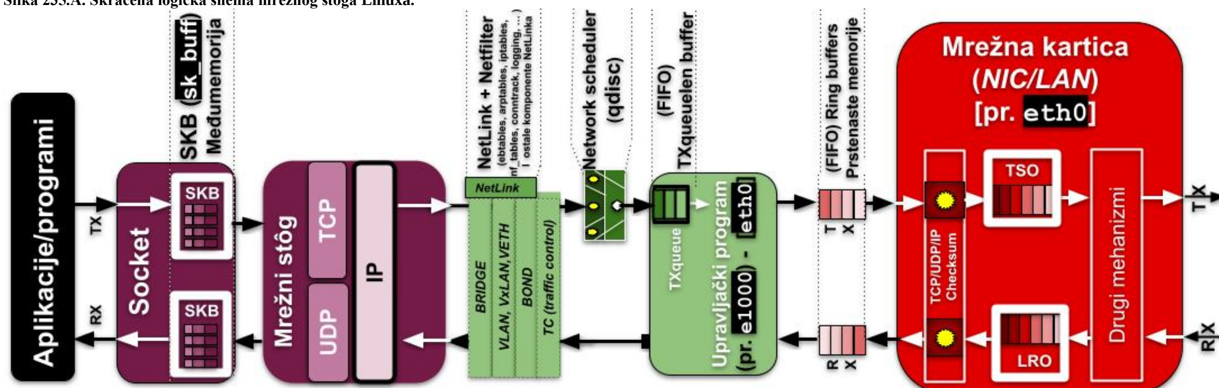
Treba znati da je veličina paketa poslanog na mrežu ograničena maksimalnom jedinicom prijenosa (tzv. **MTU**). Dakle **MTU** definira najveću veličinu paketa koji se može poslati na mrežu. To znači da veličina paketa na mreži nikada ne smije premašiti zadanu **MTU** vrijednost. Iako je prema definiciji TCP/IP protokola, moguće definirati veličinu paketa do 64kB (65.535 bajta), ovakva veličina se u pravilu nikada ne koristi jer nije podržana od većine uređaja preko kojih mrežni paketi prolaze (preklopnici, usmjerivači, vatrozidi i druga mreža oprema).

U slučajevima u kojima se kopira velika količina podataka, čak i upotreba **jumbo** paketa još uvijek ostavlja sustav s eksterno velikim brojem mrežnih paketa za obradu, a naročito na većim propusnostima mreže, poput primjerice 100Gbps, 200Gbps, 400Gbps ili više.

Veliki broj paketa je problematičan na više načina. Naime postoji značajan fiksni trošak resursa povezan sa svakim paketom koji prolazi sustavom. Svaki paket mora pronaći svoj put kroz mrežni stôg Linuxa, od gornjih slojeva protokola do upravljačkog programa mrežnog sučelja (i natrag). Više paketa znači više signala prekida (**IRQ**) od strane mrežnog sučelja. Nadalje, struktura **sk_buff** ("**SKB**") koja predstavlja pakete unutar kernela je dodatna priča, budući da mora biti sposobna podržavati skoro svaku mrežnu značajku koja se može koristiti; što dovodi do značajnog korištenja memorije po paketu, ali i troškova upravljanja memorijom, kako je slojevito vidljivo na slici **235.A**.

Dakle, postoje dobri razlozi za mogućnošću slanja podataka u veće pakete, barem za neke vrste aplikacija.

**Slika 235.A. Skraćena logička shema mrežnog stôga Linuxa.**




Duljina IP paketa pohranjuje se u IP zaglavlju. I za IPv4 i za IPv6 ta duljina definirana je u 16-bitnom polju (***Total Length***), ograničavajući maksimalnu veličinu paketa na 64 kB (65.535 bajta).
U vrijeme kada su ovi protokoli dizajnirani, paketu od 64 kB moglo je trebati nekoliko sekundi za prijenos preko internetskih veza koje su tada bile dostupne. U to vrijeme se to činilo kao nevjerojatno velik broj; sigurno bi 64 kB bilo više nego što bi itko ikada racionalno želio staviti u jedan paket. Ali vremena se mijenjaju, a 64 kB sada se može činiti kao zastrašujuće niska granica.



Za detalje vezane za **IP** protokol i izgled mrežnog paketa, pogledajte poglavlje:
**23.1. Oblik IP poruke (paketa).**

To sve znači da se podaci (sada paketi) koji putuju prema transportnom protokolu prvo moraju pohraniti u navedenu **SKB** međumemoriju, koja je ograničena na 64kB. U konačnici ***SKB*** međumemorija sadrži jedan linearni segment (praktično jedan **MSS** s gledišta *TCP* protokola). Tek potom *Scatter Gather* mehanizam može rasporediti ove podatke u višestruke međuspremnike (područja memorije). I tek tada **GSO** i **GRO** mehanizmi (i/ili **TSO**), mogu odraditi svoje grupiranje odnosno agregiranje više segmenata u jedan veći. Zatim primjerice **GRO**, kada završava agregaciju paketa za IPv4 protokol, mora u IPv4 polju (***Total Length***) osvježiti stvarnu veličinu paketa, pošto je to 16 bitno polje, otuda i ograničenje od 64kB (**$2^{16}$**). Pošto ***GSO*** radi na isti način i on ovdje ima ograničenje od 64 kB. Isto je i za IPv6 protokol.

Svijest o ovom problemu nije osobito nova, ali postoji rješenje, barem za **IPv6** protokol, a koje se može naći u **RFC 2675**, koji je usvojen 1999. godine. IPv6 specifikacija dopušta postavljanje takozvanog "*hop-by-hop*" zaglavlja s dodatnim informacija. Kao što i ime sugerira, *hop-by-hop* zaglavlje se koristi za komunikaciju opcija između dva izravno povezana sustava. **RFC 2675** omogućuje veće pakete uz nekoliko izmjena sâmog protokola.
Za slanje ovakvih "**posebnih jumbo**" paketa (engl. *Jumbo payload*), sustav mora postaviti (16-bitno) polje (duljine korisnih podataka) IP-a na nulu i dodati *hop-by-hop* zaglavlje koje sadrži stvarnu duljinu korisnih podataka. Polje u tom zaglavlju je 32 bita, što znači da ovakvi veliki paketi mogu sadržavati do 4 GB podataka; što bi sigurno trebalo biti dovoljno za sve primjene.

## *Dodatni problem(i)*

Na najnižoj razini, trenutačni kerneli (sve do 5.17.x.) ograničavaju broj fragmenata pohranjenih u ***SKB*** međumemoriju na 17, što je dovoljno za pohranjivanje paketa od 64 kB u komadima od jedne stranice memorije (engl. *memory page*).
S obzirom na ograničenja koja će očito ometati stvaranje znatno većih paketa, od kernela 5.19 povećan je maksimalan broj fragmenata na 45. Zatim je dodana i konfiguracijska opcija koja kontrolira maksimalan broj dopuštenih fragmenata za bilo koji dati paket. Međutim mnogi upravljački programi mrežnih sučelja kodiraju pretpostavke o maksimalnom broju fragmenata na koje se paket može podijeliti. Povećanje te granice bez ispravljanja upravljačkih programa moglo bi dovesti do još lošijih performansi ili čak blokiranja rada hardvera (mrežne kartice).
Iako rad rasterećenja segmentacije s ***jumbo*** paketima obično uključuje mali broj podešavanja; tek je nekoliko upravljačkih programa ažurirano s ovim zahtjevima (02.2022.g.).
U svakom slučaju, mnoge upravljačke programe trebat će ažurirati kako bi mogli rukovati s ovako velikim paketima.

Moderna mrežna sučelja obavljaju segmentaciju rasterećenja (engl. *segmentation offloading*), što znači da se veliki dio posla oko stvaranja pojedinačnih paketa obavlja unutar samog mrežnog sučelja ili unutar mrežnog stôga Linuxa; ovisno je li ova značajka implementirana u hardver ili se radi o softverskoj komponenti (**TSO**/***GRO*** ili ***GSO***). Svejedno; **GRO** (engl. *Generic receive offload*), **TSO** (engl. *TCP segmentation offload*) i **GSO** (engl. *Generic segmentation Offload*) rade tako da kernel interno može koristiti znatno veće mrežne pakete od **MTU** vrijednosti od primjerice standardnih 1500 bajta.
To radi tako da se više podataka u gore navedenim velikim segmentima sprema u međumemoriju (**SKB**), a tada se sve zajedno šalje (ili prima) prema mrežnoj kartici, koja ovisno o svom hardveru, sama razlama podatke u njoj i mreži [**MTU**] prihvatljivu veličinu (pogledajte sliku **235.A.**). Tako se drastično smanjuje broj hardverskih signala prekida (**IRQ**) od i prema mrežnoj kartici te se može značajno dobiti na propusnosti, što je drastično vidljivo na većim propusnostima mreže (10+Gbps).

Kod još većih propusnosti 100+Gbps, standardna maksimalna veličina **SKB** (TCP) međumemorije od 64kB očito više nije dovoljna. Stoga je s navedenim modifikacijama kernela što je standardno podržano od kernela 5.19+ (uz povećanje **GSO** i **GRO** dijela međumemorije), pripadajućih upravljačkih programa mrežnih kartica (sučelja), i povećanjem **MTU** iznad 1500 bajta, moguće drastično povećati propusnost mreže.
Od Linux kernela **5.19** uvedena je podrška za navedena povećanja veličine međumemorije, takozvani ***Big TCP***, koji za **IPv6,** a od kernela **6.3.** i za **IPv4**, drastično proširuje **SKB** međumemoriju (te **GRO** i **GSO**).

## GRO i GSO

Pogledajmo kako za mrežno sučelje **eth0** povećati **GRO** međumemoriju na 185.000 bajta:

```
ip link set dev eth0 gro_ipv6_max_size 185000
```

Pogledajmo kako za mrežno sučelje **eth0** povećati **GSO** međumemoriju na 185.000 bajta:

```
ip link set dev eth0 gso_ipv6_max_size 185000
```

Odnosno pogledajmo kako od kernela **6.3.** povećati obije (**GRO** i **GSO**) međumemorije za **IPv4**:

```
ip link set dev eth0 gso_ipv4_max_size 185000 gro_ipv4_max_size 185000
```



***Big TCP*** je od Linux kernela 5.19 podržan i za druga mrežna sučelja i mehanizme; primjerice za `tcp_cubic` mehanizam te za sučelja: ***ipvlan***, ***bonding***, ***macvlan, loopback*** i druge. Od kernela 6.3. **Big TCP** je podržan i za **IPv4**.



Za ostale detalje oko optimizacije 10Gbps i bržih mrežnih sučelja, pogledajte sljedeća poglavlja:
**25.3. Statistike, analiza i praćenje mrežnih paketa [**obratite pažnju na sve navedene preporuke**].**
**25.5.5. Pregled navedenih dostupnih tehnika, tehnologija i protokola.**
**25.5.6. Optimizacije mrežnih sučelja propusnosti veće od 10Gbps.**

**Izvori informacija:(260),(1295),(1300),(1301),(1302),(1450),(1639),(1643),(1644),(1645),(1646),(1647),(1648),(1768)**
`man ip link`, **RFC 2675. Pogledajte i plakat P5.**

# 25.6. Osnovna konfiguracija mreže i mrežnog podsustava

Svi primjeri i objašnjenja koja slijede, vrijede za ***Red Hat*** bazirane distribucije Linuxa, ali i mnoge druge. Mi ćemo koristiti ***CentOS/Rocky*** distribuciju Linuxa. Naime na drugim distribucijama Linuxa koje su bazirane primjerice na ***Debian*** ili ***Slackware*** Linuxu, konfiguracija se malo razlikuje, ali je koncept isti. Dakle dizajn mreže i mrežni slojevi su zajednički na svim distribucijama linuxa, a sâma konfiguracija se može malo razlikovati, ovisno o distribuciji Linuxa.



Kod svake (TCP/IP) mrežne komunikacije u pozadini sustav uvijek prvo iz imena krajnjih računala u komunikaciji odnosno tzv. *hostname-a* mora saznati njihove pripadajuće **IP** adrese jer se stvarna komunikacija događa sâmo i isključivo pomoću **IP** adresa, a ne imena računala. Sustav stoga prvo provjerava kojim redoslijedom pokušati razlučiti IP adresu iz imena računala, kako je definirano u datoteci: `/etc/nsswitch.conf`. Tek potom se kreće u stvarno razlučivanje i to obično prvo iz datoteke: `/etc/hosts`, a ako se ne uspije, tek tada kontaktiranjem **DNS** poslužitelja definiranih u datoteci: `/etc/resolv.conf`.

Konfiguracija osnovnih mrežnih parametara koja je zajednička za sve (ili barem) većinu distribucija Linuxa jest konfiguracija dostupnih DNS poslužitelja i parametara vezanih uz njih, koja se nalazi u datoteci: `/etc/resolv.conf` kao i dodatna konfiguracija povezivanja odnosno razrješavanja (engl. *resolving*) imena računala (engl. *hostame*) u IP adrese koja se nalaze popisana u datoteci: `/etc/hosts`.

---

## 25.6.1. Konfiguracijska datoteka `/etc/hosts`

Unutar datoteke `/etc/hosts` nalaze se poveznice između imena računala (*hostname*) u IP adresu, odnosno obratno; iz IP adrese u ime računala, za svako računalo za koje nije moguće na neki drugi način razriješiti IP adresu iz naziva računala.
Osim datoteke `/etc/hosts` postoje i još dvije datoteke za definiranje prava pristupa na naše računalo (poslužitelj):

- `/etc/hosts.allow` – ovdje se definira kojim se sve IP adresama <u>dopušta</u> pristup (*zadano je svima dopušteno*).
- `/etc/hosts.deny` – ovdje se definira kojim se sve IP adresama **ne dopušta** pristup (*standardno se ne koristi*).

### Sučelje povratne veze/petlje (engl. loopback)

Datoteka `/etc/hosts` je obična tekstualna datoteka kao i većina drugih koje ćemo ovdje spomenuti. Jedino što ova datoteka <u>mora</u> sadržavati jest unos za takozvano *loopback sučelje (127.0.0.1)* odnosno *sučelje povratne veze* koje zapravo označava sâmo računalo. Unos za sučelje *povratne veze* odnosno njegovu zadanu IP adresu: `127.0.0.1` je sljedećeg oblika (u ovoj datoteci):

```
127.0.0.1  localhost localhost.localdomain
```

To znači kako se posebno ime računala `localhost` razlučuje kao IP adresa: `127.0.0.1`, kako je definirano u **RFC 6761**.
Standardom je definirano, da svako razlučivanje imena računala `localhost` mora pokazivati na posebnu, takozvanu povratnu IP adresu `127.0.0.1` za IPv4 protokol. Dok za IPv6 protokol postoji IPv6 IP adresa povratne veze: `::1`, kako je definirano u **RFC 4219**. Što se tiče procesiranja odnosno obrade mrežnih paketa koji se šalju na povratnu IP adresu (`127.0.0.1`), ono je implementirano unutar mrežnog TCP/IP stôga Linuxa.

Unutar TCP/IP stôga Linuxa je definirano da mrežni paketi koji se šalju na ovu adresu, nikada **<u>NE SMIJU</u>** biti proslijeđeni niti na bilo koje mrežno sučelje: bilo ono fizičko, logičko ili virtualno, niti prema bilo kojem upravljačkom programu mrežnog sučelja. Sâmim time ne smiju i ne mogu biti proslijeđeni izvan računala, na mrežu, niti biti obrađivani od strane usmjerivača (kada bi i završili na mreži).



Ako unutar računala testiramo ili koristimo IP adresu bilo kojeg lokalnog mrežnog sučelja ((pr. `eth0`) odnosno njegove IP adrese (pr. `192.168.1.200`), odgovor će doći s *Loopback* mrežnog sučelja (`lo`). **Kada god računalo komunicira sa samim sobom putem IP adrese, taj promet prolazi isključivo preko *Loopback* sučelja**. Tako se osigurava da se lokalni promet ne šalje fizičkom mrežnom karticom, a samim time te ne prolazi niti kroz njen upravljački program, što je nepotrebno, ali i manje učinkovito. Naime prema dizajnu, kernel daje prioritet *Loopback* sučelju u odnosu na fizička sučelja za lokalni promet.
To znači da će odgovori na pakete upućene na lokalnu IP adresu biti vraćeni kroz *Loopback* (`lo`) sučelje.



Pogledajte poglavlje:
**25.7.8. Naredba *tcpdump*** ← primjer **1.** pred kraj (*snimanje lokalnog prometa [unutar računala]).*

Vezano za *loopback* IP adresu (`127.0.0.1`), podsjetite se **A** klase mreža jer je ova adresa posebna odnosno rezervirana iz opsega **A** klase mreže. Nadalje, prema standardu **RFC 3330**, definirano je kako je mreža: `127.0.0.0/8` odnosno mreža `127.0.0.0` s maskom mreže `255.0.0.0`, definirana isključivo za sučelja povratne veze (odnosno za tzv. *loopback* sučelja). U svakom slučaju IP adresa `127.0.0.1` je dobra za unutarnje testiranje, čak i u slučajevima kada nemamo niti jedno mrežno sučelje na računalu.



Podaci poslani na mrežne adrese iz opsega: 127.0.0.0 do 127.255.255.255 nikada se ne bi trebali pojaviti niti na jednoj mreži (izvan računala).

Sučelje povratne veze (*Loopback* sučelje) će postojati na sustavu, čak i kada na sustavu nemamo niti jedno drugo mrežno sučelje. To možemo vidjeti s naredbom ifconfig na sljedeći način (skratili smo ispis):

```
ifconfig -a

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
```

Isto možemo vidjeti i s naredbom: ip addr show:

```
ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
```

Dakle vidimo posebno sučelje povratne veze koje ima ime: lo kojemu je dodijeljena IP adresa: 127.0.0.1.
Tu adresu možemo primjerice testirati s naredbom ping, ili koristiti za druga testiranja.

Nadalje, moguće je kreirati i više pôd sučelja povratne veze (pr. lo:2) i dodijeliti im novu adresu iz ovog (ili bilo kojeg drugog) opsega IP adresa. Pogledajmo kako kreirati mrežno sučelje lo:2 s IP adresom 127.0.0.99 :

```
ifconfig lo:2 127.0.0.99 netmask 255.0.0.0
```

Pôd sučelja povratne veze (*loopback*) se osim za testiranje, koriste kod naprednih tehnika usmjeravanja poput ***Anycast***-a.



Za detalje oko **Anycast** komunikacije, pogledajte poglavlje:
**22.4. Anycast komunikacija.**

## Vratimo se na datoteku /etc/hosts.

Želimo li dodati unos za neko računalo zbog bilo kojeg razloga; primjerice jer ono nema svoj unos u *DNS* poslužitelju ili uopće nemamo DNS poslužitelj u svojoj mreži, sintaksa je dodavanje po jednog unosa u svaki novi redak ove datoteke.
Unosi u ovoj datoteci moraju biti u obliku (IP adresa i ime(na) računala odvajaju se razmakom ili s TAB-om):

```
IP Adresa    Ime računala (Hostname)
```

Ako primjerice računalo imena server1 i pripadajuće mu IP adrese 192.168.1.10 želimo dodati kao novi unos, u novi redak ove datoteke ćemo dodati sljedeći redak teksta:

```
192.168.1.10  server1
```

Moguće je koristiti i puno ime računala s domenom (***FQDN***) kao i dodatno ime (*alias*) ili imena (množina) računala, poput:

```
192.168.1.10  server1 srv1 server1.lab.os
```

U svakom slučaju, bilo koje od navedenih imena računala (server1 ili srv1 ili server1.lab.os) će se razriješiti kao IP adresa: 192.168.1.10 kojoj se na kraju krajeva i pristupa odnosno koja se koristi u mrežnoj komunikaciji.
Primjerice, ako se ovdje radi o *web* poslužitelju, tada ćemo do njega moći doći sa: https://server1.lab.os, a sustav će pomoću /etc/hosts datoteke razlučiti njegovo ime u IP adresu: 192.168.1.10, kojoj će potom i pristupiti.

Dodatno je moguće dodavati i komentare u ovu datoteku, pa tako sve što slijedi nakon znaka # smatra se komentarom, poput:

```
# Sada slijede poslužitelji
192.168.1.10  server1 srv1 server1.lab.os
```

U sustavu postoji i datoteka /etc/host.conf koja sadrži definiciju redoslijeda i opcija kako će se razlučivati imena računala. Osim toga u njoj se mogu definirati i napredne opcije. Standardno je u njoj zadana jedna opcija:

```
multi on
```

← Navedena opcija naznačava sustavu da svako pojedino računalo može imati više imena (sa ili bez domene)!

Ako sustav nije pomoću /etc/hosts uspio razlučiti IP adresu iz imena ili **FQDN** imena računala, on potom za razlučivanje imena računala u IP adresu pokušava kontaktirati **DNS** poslužitelje koji su definirani u datoteci: /etc/resolv.conf.



Za detalje oko datoteke /etc/resolv.conf pogledajte sljedeće poglavlje!

**Izvor informacija: (710),(711),(K-14), `man 5 hosts`, `man 5 host.conf`, `man 5 hosts_options`, `man resolv.conf`, `man hosts.allow`, `man hosts.deny`, RFC6761, RFC4219, RFC3330.**

### 25.6.2. Konfiguracijska datoteka /etc/resolv.conf

Primarno se unutar datoteke /etc/resolv.conf nalazi popis *DNS* poslužitelja na koje se naš sustav može spojiti i koristiti ih za *DNS* upite: to jest za razlučivanje IP adrese od imena računala, kao i popis domena koje se koriste i pretražuju, ali i druge opcije. Ovdje se još nalaze i upute za osnovno pretraživanje za određenu domenu ili domene, kao i definicija sâme domene računala.
Pri tome se ovdje navedeni DNS poslužitelji konzultiraju prema redoslijedu kako su navedeni, pri čemu, ako sustav od prvog (primarnog) navedenog DNS poslužitelja ne dobije odgovor na vrijeme, sustav kontaktira sljedeći (sekundarni) i tako dalje.
U slučaju kada se pretražuje ime računala bez navedenog punog imena domene (bez tzv. **FQDN**: *fully qualified domain name*) u ovoj datoteci je definirano koje ime domene će sustav automatski nadodati na to ime računala.

## Što je FQDN (*Fully qualified domain name*) ime računala?

Ako želimo pristupiti računalu po imenu, a čije ime (*hostname*) je primjerice server1 želimo i da nam se njegovo ime poveže s njegovom pripadajućom IP adresom. Naime za mrežnu komunikaciju je potrebna IP adresa, a ne ime računala te ako nismo napravili unos u /etc/hosts datoteci, već želimo da nam IP adresu razriješi *DNS* poslužitelj na kojem je napravljen unos za naš poslužitelj server1, tada nam je potrebno i puno ime računala s njegovom pripadajućom domenom. Pri tome negdje moramo imati definirane IP adrese *DNS* poslužitela, kako bi ih naše računalo uopće moglo kontaktirati.

Ako je naša domena lab.os onda tijekom pristupanja navedenom poslužitelju imena server1 ili moramo koristiti puno ime poslužitelja (*FQDN*): server1.lab.os ili njegovu domenu moramo dopisati kako bi se automatski nadodala na imena računala koja su postavljena bez pripadajućeg imena svoje domene. To je zato jer DNS poslužitelji za razlučivanje trebaju i ime domene.
Našu domenu navodimo pomoću ključne riječi **domain** dodavanjem sljedećeg retka u datoteku /etc/resolv.conf.
U našem slučaju za domenu: lab.os bi to bilo dodavanje sljedećeg retka:

```
domain lab.os
```

Potom bi slijedio jedan redak odnosno linija s ključnom riječi **search** koja označava kako će se imena računala moći koristiti i bez imena domene jer će sustav tijekom njihovog pretraživanja sâm nadodati domene navedene ovdje:

```
search lab.os
```

U slučaju kada imamo nekoliko unutarnjih domena ili domena za koje također želimo da se nadodaju kod korištenja imena računala automatski, možemo dodavati cijeli nîz domena, poput domene mreza.local.os i to svaku u novi redak, pr.:

```
search mreza.local.os
```

Dakle osnova tijekom procedure pronalaska IP adrese od nekog imena računala s domenom (FQDN) je da se ime računala izvlači iz imena domene (ključna riječ domain), a tek potom se povezuje s pripadajućom mu IP adresom.
Ovdje se može definirati i cijela lista drugih domena, poput dvije domene: mreza.local.os i lab.os i to ovako:

```
search mreza.local.os lab.os
```

Na kraju moramo navesti i IP adrese našeg *DNS* poslužitelja na kojem su napravljeni unosi za naša računala i za sva ostala računala na mreži.
Ako je IP adresa našeg *DNS* poslužitelja: 192.168.1.200 tada će sljedeći redak izgledati ovako:

```
nameserver 192.168.1.200
```



*Pošto je ovo mjesto gdje moramo definirati i sve ostale* ***DNS*** *poslužitelje koje koristimo za pristup drugim mrežama i internetu, moramo dodati i njih.*

Stoga ćemo dodati:

- Dva *DNS* poslužitelja od našeg ***ISP-a*** (*pružaoca pristupa internetu odnosno Telekoma):*
  - *Primarni:* 195.29.166.116.
  - *Sekundarni*: 195.29.166.117.
- Jedan dodatni javni DNS poslužitelj s IP adresom: 1.1.1.1.

Ključna riječ za definiciju *DNS* poslužitelja je: **nameserver**. To znači da će naš unos izgledati ovako:

```
nameserver 195.29.166.116
nameserver 195.29.166.117
nameserver 1.1.1.1
```



**Bez navedenih DNS poslužitelja razlučivanje imena računala u IP adrese na internetu neće biti moguće!**

**U pravilu ne bi trebalo koristiti više od tri (3) *DNS* poslužitelja odnosno imati više od tri nameserver unosa. Gornjim unosima, *DNS* razlučivanje (*resolving*) IP adrese od imena računala će nam raditi i prema internetu.**



Kod svake (TCP/IP) mrežne komunikacije u pozadini sustav uvijek mora prvo iz imena odredišnog računala odnosno tzv. *hostname-a* saznati njegovu pripadajuću IP adresu jer se stvarna komunikacija događa sâmo i isključivo pomoću IP adresa, a ne imena računala!

I sada ćemo imati konačnu datoteku: `/etc/resolv.conf` koja će imati sljedeće retke konfiguracije:

```
search lab.os
domain lab.os
nameserver 192.168.1.200
nameserver 195.29.166.116
nameserver 195.29.166.117
```

Moguća je upotreba i drugih opcija poput:

- `option timeout` – definira maksimalno vrijeme čekanja na odgovor od DNS poslužitelja.
- `option attempts` – definira maksimalan broj pokušaja slanja upita na DNS poslužitelj.
- `option rotate` – kada se ova opcija postavi, sustav će rotirati korištene DNS poslužitelje u datoteci `resolv.conf` za svaki DNS upit. To pomaže u uravnoteženju opterećenja između više DNS poslužitelja.

***Pogledajmo kako (zapravo) svako računalo komunicira s odredišnim računalom (primjerice na Internetu):***
Ako želimo komunicirati s računalom (pr. *web* poslužiteljem) ***FQDN*** imena: ***www google.com***, događa se slijedeće:

1. Naše računalo pokušava doznati IP adresu od ***www google.com*** iz datoteke `/etc/hosts`. Ako uspije skaće na korak **4**, a ako ne uspije ide dalje na korak **2.**
2. Iz liste DNS poslužitelja definiranih u `/etc/resolv.conf` odabire se prvi (pr. IP: `192.168.1.200`) i na njega šalje DNS upit: DNS upit za računalo/poslužitelj imena (*FQDN*): ***www google.com***.
3. Od DNS poslužitelja (u primjeru `192.168.1.200`) dobivamo odgovor da je računalo s ***FQDN*** imenom: ***www google.com*** na IP adresi: `142.250.180.196`.
4. Tek sada se naše računalo može spojiti na IP adresu: `142.250.180.196` i nastaviti komunikaciju.



Vezano za detalje oko ***DNS*** protokola i razlučivanja IP adresa, pogledajte poglavlja:
**25.8.5. DNS protokol.**
**25.8.5.4. Vrste DNS upita.**

**Izvori informacija: (712),(713),(K-14), `man 5 resolv.conf`.**

## 25.6.3. Konfiguracijska datoteka `/etc/sysconfig/network`

Datoteka `/etc/sysconfig/network` koristi se za bazične mrežne postavke cijelog sustava**.**
U ovoj datoteci se definiraju informacije za cijelo računalo, a koje se tiču i usmjeravanja (engl. *Routing*) koje se primjenjuje za sva mrežna sučelja odnosno za sve mrežne kartice u računalu. Osim toga ovdje se postavlja i sâmo ime računala (*hostname*).



Za više informacija o **`/etc/sysconfig/`** strukturi direktorija i datoteka pogledajte poglavlje:
**7.4. Upravljanje konfiguracijom servisa**

Vratimo se na datoteku `/etc/sysconfig/network` u kojoj se najčešće koriste sljedeći parametri:

**`NETWORKING=`**

- `yes` - naznačava da će mreža biti konfigurirana i korištena.
- `no` - naznačava kako mreža neće biti konfigurirana.

**`HOSTNAME=`**

Ovo je sâmo ime računala (*hostname*), ali se preporuča ime računala s domenom (*Fully Qualified Domain Name*).
Primjerice unos poput: `HOSTNAME=server1.local.hr`. Dakle ovdje se definira ime našeg računala.

**`GATEWAY=`**

Ovo je IP adresa *Default Gaetway*-a (podrazumijevani usmjerivač) za ovo računalo.
Ovdje se definira podrazumijevani usmjerivač u slučajevima kada imamo više mrežnih kartica na računalu. Primjerice: `GATEWAY=192.168.1.1`

**NOZEROCONF**=

- `yes` - pri ovoj postavci, *zeroconf* se neće koristiti, što je standardna postavka.
- `no` - *zeroconf* će se koristiti. Ovo nije standardna postavka.

Napomena: *ZEROCONF* je mogućnost da se tijekom pokretanja sustava koristi IP adresa iz točno određenog opsega IP adresa. Obično iz opsega: 169.254.0.0, ali sve bez potrebe za konfiguracijom mrežne kartice. *Zeroconf* se **ne koristi** u većini slučajeva.

Pogledajmo primjer kako bi izgledala naša konfiguracijska datoteka: `/etc/sysconfig/network` da je konfiguracija našeg računala sljedeća:

- *Ime našeg računala (Hostname)*: `desktop1`.
- *Podrazumijevani pristupnik/usmjerivač (Default Gateway) IP*: `192.168.1.254`.

Pri gore navedenim odnosno definiranim vrijednostima bi datoteka: `/etc/sysconfig/network` izgledala ovako:

```
NETWORKING=yes
HOSTNAME=desktop1
GATEWAY=192.168.1.254
NOZEROCONF=yes
```

Dakle datoteka `/etc/sysconfig/network` je mjesto za definiranje imena računala (*hostname*) s njegovim pripadajućim *Default Gateway-em* uz naređenje sustavu, hoće li mreža uopće biti aktivirana ili neće, i to globalno na cijelom računalu.



Ime računala (*hostname*), potrebno je zapisati i u datoteku: `/etc/hostname` i to obično sâmo upisivanjem imena računala. U našem slučaju, za ime računala: `desktop1`, bi ova datoteka sadržavala samo sljedeći redak:

```
desktop1
```

Ime računala je moguće provjeriti s naredbom `hostname` na sljedeći način:

```
hostname
```

Odnosno moguće je promijeniti ime računala s istom naredbom uz navođenje novog imena računala.
Primjerice, ako je novo ime računala *`desktop2`* tada trebamo pokrenuti sljedeću naredbu:

```
hostname desktop2
```



Isto je moguće postići i s novijom naredbom **`hostnamectl`** iz *systemd* softverskog paketa (ako koristite *systemd*), a čije primjere možete vidjeti u poglavlju:
**7.3.2.5. Drugi korisni programi i komponente unutar *systemd* paketa.**



**Ime računala (*hostname*) se zapisuje i u *sysctl* varijablu:** `kernel.hostname` **tijekom postavljanja imena računala.**

Ime računala možemo prema potrebi i ručno promijeniti s naredbom ***sysctl*** na sljedeći način:

```
sysctl kernel.hostname=desktop2
```



Za ***Red Hat 7+*** opcije definirane u: `/etc/sysconfig/network` se mogu definirati i u datotekama:
**`/etc/sysconfig/network-scripts/ifcfg-XY`** (pr. za **GATEWAY**) ili u **`/etc/hostname`** (za ***hostname***).
Mada se **GATEWAY** (odnosi se na *Default Gateway*), ako na računalu imamo više mrežnih sučelja, i dalje definira ovdje.



Od ***Red Hat 9x*** više se ne koriste konfiguracijske datoteke u direktoriju: `/etc/sysconfig/network-scripts/` (tzv. **ifcfg** datoteke). Naime od ove inačice se isključivo koristi **Network Manger** servis koji konfiguracijske datoteke sprema u direktorij: **`/etc/NetworkManager/system-connections/`**.



Za više informacija o **`Network Mananager`** servisu i novom načinu konfiguracije, pogledajte poglavlje:
**25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**

**Izvori informacija: (714),(715),(716),(K-14), `man sysctl`, `man 5 hostname`.**

## 25.6.4. Nazivi mrežnih kartica u Linuxu

Standardno mrežne kartice odnosno mrežna sučelja u Linuxu se nazivaju na sljedeći način, vidljiv u tablici:

| Ime mrežne kartice odnosno mrežnog sučelja | Opis |
|---|---|
| eth0 | Označava prvu mrežnu karticu. |
| eth1 | Označava drugu mrežnu karticu. |
| eth2 | Označava treću mrežnu karticu. |

Navedeni način označavanja mrežnih sučelja (kartica) vrijedi sve do ***Red Hat 7.x*** koji počinje koristiti drugačiji način označavanja, koji ćemo objasniti malo kasnije!.

Osim standardnih mrežnih sučelja moguće je koristiti i takozvane pôd kartice to jest pôd sučelja (Engl. *Subinterface*), koja se vežu na postojeću (pr. fizičku) mrežnu karticu i ponašaju se kao logički zasebne mrežne kartice spojene na primarnu odnosno fizičku karticu. Za ove mrežne kartice potrebno je definirati sve parametre mreže kao i za normalne odnosno fizičke mrežne kartice. Pogledajmo tablicu s nazivima ovakvih pôd sučelja odnosno takozvanih pôd kartica:

| Ime mrežne kartice | Opis |
|---|---|
| `eth0:0` | Prva pôd kartica na prvoj mrežnoj kartici (`eth0`). |
| `eth0:1` | Druga pôd kartica na prvoj mrežnoj kartici (`eth0`). |
| … | … |
| `eth1:0` | Prva pôd kartica na drugoj mrežnoj kartici (`eth1`). |
| `eth1:1` | Druga pôd kartica na drugoj mrežnoj kartici (`eth1`). |

Ove pôd kartice uočljive su po tome što u nazivu imaju dodatak na fizičku (ili logičku) mrežnu karticu `:Broj`.
Primjerice na `eth0` mrežnoj kartici bi prva pôd kartica odnosno sučelje bila `0` pa će se onda to mrežno sučelje zvati: `eth0:0`



Moguća je upotreba i ***VLAN*** mrežnih sučelja, što je objašnjeno u naprednom poglavlju:
**20.6.2. VLANovi odnosno virtualne lokalne mreže.**

*VLAN* mrežna sučelja moraju pripadati fizičkoj mrežnoj kartici, pôd kartici ili nekoj drugoj vršnoj vrsti mrežnog sučelja ili kartice.
*VLAN* mrežna sučelja definiraju (označavaju) se s točkom `.` na postojeću mrežnu karticu nakon koje slijedi broj *VLAN*-a. Tako bi se primjerice **VLAN 14** mrežno sučelje na `eth0` mrežnoj kartici zvalo: `eth0.14`.
*VLAN* mrežna sučelja se ponašaju kao i fizičke mrežne kartice te je za njih isto potrebno konfigurirati sve IP parametre mreže.

Pogledajmo primjer *VLAN* mrežnih sučelja koja se vežu na fizičke mrežne kartice `eth0` i `eth1` za `VLAN` broj **`44`**

| Ime mrežne kartice | Opis |
|---|---|
| `eth0.44` | ***VLAN 44*** mrežno sučelje na prvoj mrežnoj kartici (`eth0`). |
| `eth1.44` | ***VLAN 44*** mrežno sučelje na drugoj mrežnoj kartici (`eth1`). |

Primjer *VLAN* mrežnog sučelja koje se veže na pôd kartice `0.` i `1.` od fizičke mrežne kartice `eth0` također za `VLAN` broj **`44`**

| Ime VLAN pôd mrežne (*subinterface*) kartice | Opis |
|---|---|
| `eth0:0.44` | ***VLAN*** br. **44** prve mrežne kartice `eth0` i to na njenoj prvoj pôd kartici (`eth0:0`). |
| `eth0:1.44` | ***VLAN*** br. **44** prve mrežne kartice `eth0` i to na njenoj drugoj pôd kartici (`eth0:1`). |

Osim navedenih vrsta posebnih mrežnih sučelja postoje i druge vrste logičkih mrežnih sučelja poput:

- ***Agregiranih*** (engl. *Bonding* ili *Teaming*) mrežnih sučelja koje služe za *agregaciju* odnosno povezivanje više fizičkih mrežnih kartica (`eth0`, `eth1`, `eth`**`N`**) u jednu logičku mrežnu karticu (`bond0`, `bond1`, `bond`**`N`**) odnosno mrežno sučelje. *Terminologija tvrtke* ***Cisco*** za ovu funkcionalnost je *Ether Channel*. **Pogledajte poglavlje: 20.6.2.**
- ***Bridge*** mrežnih kartica: one služe za spajanje više mrežnih sučelja u jedan "*Bridge*" odnosno mrežni most odnosno uređaj koji radi na OSI sloju dva (OSI 2), a koji propušta sav promet s jedne kartice na drugu i obratno. On se ponaša poput preklopnika (*switcha*): jedna mrežna kartica = jedno mrežno sučelje (port) na preklopniku. **Poglavlje: 20.6.1.**
- Ostalih specifičnih vrsta mrežnih sučelja poput: **TUN i TAP** te **VETH**, ali i drugih. Poglavlja: **20.6.3** i **20.6.4**.

**Nazivi za *Bonding*** mrežna sučelja su redom: `bond0`, `bond1`, … `bondN`.
Na *bonding* mrežna sučeljima također je moguće kreirati i pôd sučelja/kartice, ali i *VLAN* mrežna sučelja.
Za svako od tih logičkih mrežnih sučelja moraju se konfigurirati i svi mrežni parametri: *IP adresa*, maska mreže, … .

**Nazivi *Bridge*** mrežnih sučelja su redom: `br0`, `br1`, … `brN`, ali prema potrebi mogu biti i drugačiji; primjerice: `vmbr0`, …
Na *bridge* mrežna sučelja moguće je kreirati pôd sučelja/kartice te također koristiti i *VLAN*-ove unutar *bridgea*.
Za svaku od tih logičkih mrežnih sučelja moraju se također konfigurirati i svi mrežni parametri (*IP adresa*, maska mreže, …).

**O svim navedenim vrstama mrežnih sučelja/kartica govorit ćemo detaljnije u zasebnim poglavljima koja slijede.**



Od **Red Hat 7+**, pojavio se i drugačiji način označavanja mrežnih kartica zbog sustava s većim brojem mrežnih kartica te potrebe da se iz naziva kartice može vidjeti gdje je ta kartica (ako je ona hardverski uređaj) spojena:

- *On-board* odnosno nalazi se već ugrađena (zalemljena) na matičnoj ploči računala.
- Ili spojena (*ugrađena*) kao zasebna PCI/PCI-Express kartica odnosno ona koju smo sâmi ugradili u računalo.

Nazivi ovih novih mrežnih kartica su sljedeći (navest ćemo ih samo nekoliko):

| Ime mrežne kartice | Opis |
|---|---|
| `em0,em1,em2,… emN`<br>`eno0,eno1,eno2,… enoN` | Mrežne kartice koje su na matičnoj ploči (tzv. "*on-board*"). |
| `p1p1, p1p2, … pxpy` | Mrežne kartice vidljive prema *PCI* sabirnici na koju su spojeni, i to prema imenu: **p**(*broj PCI sabirnice*)**p**(*broj PCI utora*). |
| `ens0,ens1,enpYsZ,ensX` | Mrežne kartice spojene na neki PCI utor (**Y**,**Z**) ili se radi o *hotplug* uređaju (**X**). |

Ovaj novi način označavanja za sustave s više mrežnih kartica nudi jedinstvenu identifikaciju mrežnog sučelja (kartice). Naime od ***RedHat/CentOS* 7.x *Systemd*** servis odnosno njegov ***udev*** servis brine se o prepoznavanju svôg hardvera pa tako i mrežnih kartica, pri čemu se vodi sljedećom shemom naziva (od prve do zadnje sheme) nakon prepoznavanja kartica.
***Udev*** pravila koja se koriste tijekom inicijalizacije i konfiguracije mrežnih sučelja su za ***RedHat/CentOS* 7+** locirana u direktoriju: `/usr/lib/udev/rules.d/`, a najvažnija su sljedeća: `60-net.rules` (za inicijalizaciju sučelja), `71-biosdevname.rules` (za nazivanje sučelja), `75-net-description.rules` (za dodatne parametara sučelja) te `80-net-name-slot.rules` (za završne promjene u nazivu sučelja).



Pogledajte poglavlje:
**11.1.2.1.1. *Udev* pravila (*udev rules*).**

Ako je mrežna karticu ugrađena na matičnu ploču ona će dobiti neko od imena: `enoX`, a ako je ona na zasebnoj kartici (*PCI*/*PCI* express), tada će se zvati: `ensX`. Nadalje, kartica može dobiti i ime u kojemu je definicija njene pozicije na sabirnici; `enpXsY` (**X** i **Y** su oznake sabirnice i utora), kao i `enpXpY`. Zatim ju sustav može nazvati i po njenoj **MAC** adresi, poput: `enx1a2b3b4b5b6e`. I konačno, ako niti jedna od navedenih shema u nazivanju mrežne kartice (sučelja) nije pomogla, tada će ona dobiti naziv poput: `enp1sX` (obično `enp1s0`).
U konačnici, u daljim primjerima je važno shvatiti logiku konfiguracije, a sâmi nazivi mrežnih sučelja odnosno kartica, za to ionako nisu toliko važni.

---

## 25.6.5. Konfiguracija mrežnih kartica

Konfiguraciju mrežnih parametara svake mrežne kartice (hardverska kartica) odnosno mrežnog sučelja: hardverska ili logička odnosno softverska „*mrežna kartica*", možemo odraditi: **statički** (ručno) ili **dinamički** preko *DHCP* poslužitelja.
Pod mrežnim parametrima podrazumijevamo konfiguraciju:

- IP adrese te pripadajuće maske mreže (*Netmask*) kao i njihove mrežne IP adrese te IP adrese mreže.
- *Broadcast* IP adrese, kao i podrazumijevanog usmjerivača *(tzv. Default Gateway) te* ostalih parametara.

Ručno se konfiguracija odrađuje preko naredbe `ifconfig` ili novije naredbe `ip`.

Mi ćemo se bazirati na ručnoj konfiguraciji pomoću naredbe `ifconfig`, ali ćemo spominjati i noviju naredbu `ip`.

Važno je znati kako se svako pojedino mrežno sučelje konfigurira u zasebnoj datoteci (tzv. ***ifcfg*** datoteci) imena:
`/etc/sysconfig/network-scripts/ifcfg-`*`IME-MREŽNE-KARTICE`*

Pri čemu je *`IME-MREŽNE-KARTICE`* ime mrežnog sučelja odnosno kartice u Linuxu, poput primjerice: `eth0`, `eth1` i slično.



Od ***Red Hat 9+*** uvode se konfiguracijske datoteke u novom formatu, koje isključivo koristi **Network Manager** servis.
**Od Red Hat 9+ napuštena je upotreba gore navedenih ifcfg konfiguracijskih datoteka.**

#### 25.6.5.1. Statička konfiguracija

Statička konfiguracija mrežnog sučelja može biti privremena; do sljedećeg *restarta* računala.
Dok je trajna ili permanentna ona koja će biti aktivirana i nakon ponovnog pokretanja (*restarta*) računala.

##### 25.6.5.1.1. Privremena statička konfiguracija

**Privremena konfiguracija mrežnog sučelja, radi se s naredbama `ifconfig` ili `ip`**
Pomoću naredbe `ifconfig` (engl. *Interface Configuration*) možemo napraviti privremenu konfiguraciju mrežne kartice odnosno IP parametara mreže, mrežne kartice (sučelja). Navedena naredba se koristi i na većinu **UNIX** operativnih sustava.
Ovu naredbu koristimo i kada želimo sâmo vidjeti (ispisati) trenutnu konfiguraciju mrežnih kartica.
Razvojem mrežnog modela Linux kernela i novih funkcionalnosti, konfiguracija mreže s naredbom `ifconfig` se sve manje koristi jer nosi za sobom mnoga nasljeđa prošlosti koja je gotovo nemoguće popraviti. Zbog toga razvoj svih alata iz grupe `net-tools` u koju naredba `ifconfig` pripada, danas se sâmo održava i ne razvijaju se nove mogućnosti.
Naredbe iz grupe `net-tools` čine sljedeće naredbe:

- `ifconfig` – za prikaz i konfiguraciju mrežnih sučelja/kartica.
- `route` – za prikaz i konfiguraciju ruta (tablica usmjeravanja) na sustavu.
- `arp` – za prikaz i konfiguraciju ARP parametara mreže.
- `netstat` – za prikaz raznih parametara i statistika rada mrežnih sučelja i servisa.
- `iptunnel` – za prikaz i konfiguraciju mrežnih sučelja za *tuneliranje*.
- `ipmaddr` – za prikaz, konfiguraciju i rad s *multicastom*.
- `tunctl` – za prikaz i konfiguraciju *TUN* i *TAP* mrežnih sučelja.
- `brctl` – za prikaz i konfiguraciju *bridge* mrežnih sučelja.

Zbog navedenih ograničenja (o njima kasnije) se sve više prelazi na novu grupu `iproute2` alata.

Usporedna lista nekih od zamjenskih naredbi iz nove grupe u odnosu na staru je vidljiva u tablici:

| Softverski paket: **net-tools** | Novi softverski paket: **iproute2** | |
|---|---|---|
| **Stara naredba** | **Nova naredba** | **Opis rada** |
| **ifconfig** | **ip addr** ili **ip link** | Prikaz ili rad s mrežnim sučeljem/karticom. |
| **ifconfig (interface status)** | **ip -s link** | Ispis statusa rada mrežnog sučelja/kartice. |
| **route** | **ip route** | Rad s rutama i tablicama usmjeravanja. |
| **arp** | **ip neigh** | Rad s ARP protokolom. |
| **netstat** | **ss** | Ispis statistika rada mrežnih sučelja/kartica. |
| **netstat -g** | **ip maddr** | Rad s *multicast* IP adresama. |
| **netstat -r** | **ip route** | Ispis tablica usmjeravanja. |
| **iptunnel** | **ip tunnel** | Rad s mrežnim sučeljima za *tuneliranje*. |
| **ipmaddr** | **ip maddr** | Rad s *multicast* IP adresama. |
| **tunctl** | **ip tuntap** | Rad sa *TUN* i *TAP* mrežnim sučeljima. |
| **brtcl** | **bridge** | Rad s *bridge* mrežnim sučeljima. |

Pogledajmo i kratki opis funkcionalnosti pozivanjem naredbe `ifconfig`

| Naredba | Opis |
|---|---|
| **ifconfig** | Ispiše konfiguraciju svih mrežnih sučelja (kartica) koja su aktivna. |
| **ifconfig -a** | Ispiše konfiguraciju svih mrežnih sučelja (kartica): i aktivnih i neaktivnih. |
| **ifconfig eth**x | Ispiše konfiguraciju za točno određeno mrežno sučelje (karticu). Primjerice za `eth0` bi to bilo: `ifconfig eth0`. |

Pogledajmo kratki opis istih funkcionalnosti pozivanjem novije naredbe `ip`

| Naredba | Opis |
|---|---|
| **ip addr show** | Ispiše konfiguraciju svih mrežnih sučelja (kartica) koja su aktivna. |
| **ip addr show eth**x | Ispiše konfiguraciju mrežnog sučelja (kartice) **eth**x (pr. **eth0**). |
| **ip link show** | Ispiše konfiguraciju svih mrežnih sučelja (kartica) koja su aktivna, s prikazom detalja o link sloju mreže. |

**Aktivacija i deaktivacija mrežnih sučelja (mrežnih kartica/*interfacea*).**
Svako mrežno sučelje koje postoji na sustavu inicijalno je potrebno i aktivirati. U slučajevima kada nam ono više ne treba možemo ga deaktivirati, a ono će pritom i dalje postojati na razini operativnog sustava, ali neće biti "*uključeno*".
Slično kao da smo ga ugasili.



Svako mrežno sučelje može biti aktivno ili neaktivno, ali i dalje postojeće, odnosno vidljivo na sustavu!

Mrežno sučelje aktiviramo ili deaktiviramo na sljedeći način, pomoću naredbe `ifconfig`

| Naredba | Opis |
|---|---|
| **ifconfig eth**x **up** | **eth**x je ime mrežnog sučelja; za `eth0` to je naredba: `ifconfig eth0 up` |
| **ifconfig eth**x **down** | **eth**x je ime mrežnog sučelja; za `eth0` to je naredba: `ifconfig eth0 down` |

Odnosno pomoću novije naredbe `ip link` to isto postižemo sa sljedećim naredbama:

| Naredba | Opis |
|---|---|
| **ip link set eth**x **up** | **eth**x je ime mrežnog sučelja; za `eth0` to je naredba: `ip link set eth0 up` |
| **ip link set eth**x **down** | **eth**x je ime mrežnog sučelja; za `eth0` to je naredba: `ip link set eth0 down` |



Na mnogim drugim UNIX operativnim sustavima (**Solaris, FreeBSD, AIX, …**) i dalje se koristi naredba **`ifconfig`**!

**Slijede primjeri**
Aktivirajmo mrežno sučelje `eth0` upotrebom naredbe `ifconfig` na sljedeći način:
```
ifconfig eth0 up
```
Ili aktivirajmo mrežno sučelje s upotrebom novije i naprednije naredbe `ip` na sljedeći način:
```
ip link set eth0 up
```
Sada deaktivirajmo mrežno sučelje `eth0`
```
ifconfig eth0 down
```
Ili deaktivaciju možemo napraviti s novijom naredbom `ip`
```
ip link set eth0 down
```

**Konfiguriranje IP parametara**
U primjerima ćemo koristiti mrežno sučelje `eth0`. Definirajmo sljedeće IP parametre koje ćemo konfigurirati:

- *IP adresa*: `172.17.100.1`
- Maska mreže (*netmask*): `255.255.255.0`
- *Broadcast IP adresa*: `172.17.100.255`

**Konfiguracija IP adrese i maske mreže mrežnog sučelja**
Za konfiguraciju IP adrese željenog mrežnog sučelja (`<mrežno sučelje>`) sintaksa je:
```
ifconfig <mrežno sučelje> <IP adresa>
```
Odnosno sintaksa s novijom naredbom `ip` je sljedeća:
```
ip address add <IP adresa> dev <mrežno sučelje>
```

Pogledajmo kako ćemo konfigurirati naše mrežno sučelje `eth0` s parametrima navedenim na prethodnoj stranici:
```
ifconfig eth0 172.17.100.1 netmask 255.255.255.0
```

Ili, ako želimo, isto možemo postići s novijom naredbom `ip`
```
ip address add 172.17.100.1/24 dev eth0
```
Opis: `/24` je maska mreže (*netmask*) od 24 bita koja odgovara vrijednosti maske mreže: 255.255.255.0

**Konfiguracija *broadcast* adrese mrežnog sučelja**
Napomena: ako je pâr IP adresa i pripadajuća mu maska mreže (*Netmask*) pravilno konfiguriran i *broadcast* IP adresa će biti ispravno i automatski konfigurirana.

Sintaksa je pri tome:
```
ifconfig <mrežno sučelje> broadcast <broadcast adresa>
```

Odnosno sintaksa s novijom naredbom `ip` je sljedeća:
```
ip address add <IP adresa> broadcast <Broadcast adresa> dev <mrežno sučelje>
```

Pogledajmo kako ćemo konfigurirati *broadcast* adresu našeg mrežnog sučelja `eth0`

```
ifconfig eth0 broadcast 172.17.100.255
```

Ili kako postići isto s novijom naredbom `ip`

```
ip address add 172.17.100.1/24 broadcast 172.17.100.255 dev eth0
```

Na kraju je potrebno i podići (aktivirati) mrežno sučelje sa sljedećom naredbom:

```
ifconfig eth0 up
```

Ili s novijom naredbom `ip`

```
ip link set eth0 up
```

Sve gore navedeno moguće je konfigurirati i u jednom retku. Pogledajmo kako.

Pri tome je sintaksa:

**ifconfig** `<mrežno sučelje> <IP adresa>` **netmask** `<netmask adresa>` **broadcast** `<broadcast adresa>`

ili

**ip address add** `<IP/CIDR Netmask>` **broadcast** `<broadcast IP>` **dev** `<mrežno sučelje>`

Pogledajmo kako to napraviti u jednom retku:

```
ifconfig eth0 172.17.100.1 netmask 255.255.255.0 broadcast 172.17.100.255
```

Ili kako isto postići s novijom naredbom `ip`

```
ip address add 172.17.100.1/24 broadcast 172.17.100.255 dev eth0
```

U slučaju kada istovremeno konfiguriramo i IP adresu i masku mreže, *broadcast* IP adresa će biti automatski izračunata i dodana na sučelje.
Primjer ovakve konfiguracije slijedi:

```
ifconfig eth0 172.17.100.1 netmask 255.255.255.0
```

Ako želimo obrisati konfiguraciju mreže mrežnog sučelja, dovoljno je da ga isključimo:
Sintaksa je: **ifconfig** `<mrežno sučelje>` **down**

Pogledajmo primjer za isključivanje mrežnog sučelja (kartice) `eth0` i brisanje njene konfiguracije:

```
ifconfig eth0 down
```

Odnosno pogledajmo kako isključiti mrežno sučelje s novijom naredbom `ip`

```
ip link set eth0 down
```



Od **Red Hat 8.x** omogućena je konfiguracija mrežnih sučelja upotrebom **NetworkManager** servisa te dodatne naredbe `nmcli`!.

**Od Red Hat 9 Network Manger je jedini zadani servis za konfiguraciju mrežnih sučelja!**

**Izvor informacija: (K-14), `man ip`, `man ip address`, `man ifconfig`, `man ip link`, `man nmcli`.**

### 25.6.5.1.2. Trajna (permanentna) ručna konfiguracija mreže

U slučajevima kada želimo trajno ručno konfigurirati mrežne (IP) parametre našeg računala ili češće poslužitelja, potrebno je uređivanje konfiguracijskih datoteka za mrežu. U praksi: računala, mrežne pisače i druge sekundarne mrežne uređaje, najbolje je konfigurirati preko ***DHCP*** poslužitelja.

S druge strane sve primarne mrežne uređaje poput preklopnika (*switcheva*) i usmjerivača (*routera*) te poslužitelje, potrebno je ručno konfigurirati, jer nam je njihova dostupnost vrlo važna i u slučajevima kada je *DHCP* poslužitelj nedostupan ili neispravan. Konfiguracijske datoteke mrežnih sučelja odnosno kartica koriste se za svaku mrežnu karticu zasebno, prema principu: jedna datoteka=jedna mrežna kartica. U trenutku pokretanja operativnog sustava, čitaju se ove konfiguracijske datoteke u kojima se nalaze upute za rad sâmih mrežnih kartica odnosno mrežnih sučelja kao i konfiguraciju njihovih IP parametara rada.

## Osnovni IP parametri

Osnovna mrežna konfiguracijska datoteka za ***Red Hat*** do **v.9.x** (***CentOS***, ***Rocky***, ***Fedora*** i sl.) distribucije Linuxa, nalazi se unutar direktorija: `/etc/sysconfig/network-scripts/`.
Njeno ime je oblika `ifcfg`-**ime-mrežne-kartice**; tako se primjerice za `eth0` mrežnu karticu, datoteka unutar ovog direktorija zove: `ifcfg-eth0`.
Dakle datoteka za `eth0` mrežnu karticu s punom putanjom je: `/etc/sysconfig/network-scripts/ifcfg-eth0`.

Pogledajmo što sve može sadržavati ova datoteka (opisat ćemo samo nekoliko najvažnijih parametara):

| Parametar | Vrijednost | Opis |
|---|---|---|
| `DEVICE=` | `eth0` ili `eth1` ili `eth`**`X`** ili ... | Ovo je naziv mrežnog sučelja (kartice): `eth0` ili `eth1` ili `eth2`, .... |
| `BOOTPROTO=` | `none` ili `bootp` ili `dhcp` ili `static` | Vrsta protokola ili metoda koja se koristi za dodjeljivanje IP parametara mreže ili podizanja sustava s mreže. Opcija `static` se ne koristi u *RedHat/CentOS* v.7+ pa umjesto nje treba koristiti opciju `none`. |
| `IPADDR=` | ***IP** adresa* | ***IP*** adresa ove mrežne kartice, primjerice: `192.168.1.10`. |
| `NETMASK=` | ***Maska mreže*** *(Netmask)* | ***Maska mreže***, primjerice: `255.255.255.0`. |
| `ONBOOT=` | `yes` ili `no` | Treba li ova mrežna kartica biti aktivna tijekom podizanja (pokretanja) sustava (`yes`) ili ne treba (`no`). Standardno je `yes`. |
| `USERCTL=` | `yes` ili `no` | `yes` - svi korisnici imaju pravo kontroliranja rada mrežne kartice.<br>`no` - sâmo `root` korisnik ima sva prava konfiguracije sučelja. |
| `NM_CONTROLLED=` | `yes` ili `no` | Ima li *Network Manager* program/alat prava konfiguriranja mrežnih kartica. Za ***RedHat/CentOS 8+*** potrebno je uključiti ga (`yes`). |
| `NAME=` | `eth0` ili `eth1` ili `eth`**`X`** ili ... | Naziv mrežnog sučelja ili mrežne kartice: `eth0` ili `eth1` ili `eth2`, ....<br>Ovaj naziv će biti vidljiv iz mrežnih programa. |
| `TYPE=` | `Ethernet` ili `Bond` ili `Bridge` ili `Tap` ili `Veth` ili `vlan` ili `OVSBridge`, ... | Vrsta mrežnog sučelja; standardno je to: `Ethernet`.<br>Poglavlja: za ***Bridge***: **20.6.1.2.**, za ***Bond***: **20.6.6.2**. te za ***TAP***: **20.6.3**. odnosno za ***VETH*** *sučelje* je to poglavlje: **20.6.4.**,<br>Za **OpenvSwitch [OVS]** (***OVSBridge***), pogledajte poglavlje: **26.8.5**. |
| `ETHTOOL_OPT=` | ***Ethtool*** razne opcije | Razne opcije `ethtool` programa koje želimo aktivirati pri pokretanju računala, poput brzine rada (10/100/1000Mbps), vrste *duplexa* i slično.<br>*Pogledajte primjere dvije stranice niže, kao i u poglavlju:*<br>***10.5.2.2. Interrupt moderation.*** |
| `VLAN=` | `yes` ili `no` | Koristimo li **VLAN**ove (obično ne) i želimo li uključiti podršku za njih.<br>Pogledajte poglavlje: **20.6.2.3.** |
| `DEFROUTE=` | `yes` ili `no` | Kod računala s više mrežnih sučelja, za mrežno sučelje (mrežnu karticu) koja ima direktnu vezu na *Default Gateway* ovdje je potrebno postaviti `yes` kao i za računalo s jednom jedinom mrežnom karticom/sučeljem. |
| `BRIDGE=` | Ime mrežnog sučelja: (pr. `eth0`) | Sâmo ako mrežno sučelje (pr. `eth0`) pripada nekom mrežnom mostu (*bridge*-u) to se definira ovdje. Pogledajte poglavlje: **20.6.1.2.** |
| `MTU=` | Vrijednost **MTU**a | Ako želimo mijenjati ***MTU*** mrežnog sučelja to se radi ovdje[(inače ne dirati)].<br>Pogledajte poglavlje: **23.4.2. Maximum Transmission Unit (*MTU*).** |
| `DEVICETYPE=` | Za posebne uređaje; primjerice za *OpenvSwitch je* `ovs` | Za **OpenvSwitch [OVS]**, pogledajte poglavlje: **26.8.5**. |

Postoji još cijeli nîz parametara i njihovih pripadajućih vrijednosti, ali ovo su najbitnije odnosno najčešće korištene. Preporuka je za Red Hat do v.8.x postaviti: `NM_CONTROLLED=no` na poslužiteljima jer ne želimo prepustiti mrežnom servisu mogućnost da sâm rekonfigurira mrežna sučelja koja smo ručno konfigurirali s razlogom.

*Network Manager* je alat koji je zadužen za automatsku konfiguraciju mrežnih kartica primjerice pomoću ***DHCP***-a koji obično ne želimo na poslužiteljima, već eventualno samo na stolnim računalima ili laptopima.



***Od Red Hat 9.x. na dalje, Network Manager je nužan za rad mreže i bez njega je konfiguracija mreže nemoguća!***

***Međutim Network Manager više ne koristi tzv. ifcfg konfiguracijske datoteke u direktoriju:*** `/etc/sysconfig/network-scripts/`.

Za ***RedHat/CentOS*** 6.x. ako želimo trajno zaustaviti *Network Manager servis* pokrenimo sljedeće naredbe:

```
chkconfig NetworkManager off
service NetworkManager stop
```

Odnosno na ***RedHat/CentOS*** 7.x. (ali ne za ***RedHat/CentOS 8*** i novije) to možemo postići s:

```
systemctl disable NetworkManager
```



O radu sa servisima smo već govorili u poglavlju:
***7.3. Stanja rada Linuxa (Runlevels).***



Od v.8.x ***Network Manager*** zamjenjuje ***network*** servis zadužen za rad s mrežnim sučeljima (konfiguracija i pokretanje, spuštanje te restart) stoga ga **nemojte** zaustavljati. Dakle ovo vrijedi samo za ***Red Hat*** **v.8.x.** i novije.

**Slijede primjeri.**
Ručno konfigurirajmo mrežno sučelje: `eth0` sa sljedećim IP parametrima:

- IP adresa: `192.168.1.10`
- Maska mreže: `255.255.255.0`

S navedenim IP parametrima konfiguracijska datoteka: `/etc/sysconfig/network-scripts/ifcfg-eth0` će izgledati ovako:

```
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
IPADDR=192.168.1.10
NETMASK=255.255.255.0
NM_CONTROLLED=no                      #→ Za Red Hat 8+ mora biti postavljeno na =yes
USERCTL=yes
```

**Konfiguracija DNS poslužitelja**
Konfiguracijska datoteka u kojoj se definiraju *DNS* poslužitelji koje će naše računalo kontaktirati je, kako smo već govorili u poglavlju:

Konfiguracijska datoteka `/etc/resolv.conf` može sadržavati i druge parametre poput:

| Parametar | Vrijednost | Opis |
|---|---|---|
| `nameserver` | Slijedi IP adresa ***DNS*** poslužitelja | Primjerice: `195.29.150.3` |
| `domain` | Slijedi ime domene | Primjerice: `mojadomena.com`.<br>Dodaje se ako ime računala (*hostname*) upišemo bez domene (FQDN). |
| `search` | Slijedi ime dodatnih domena | Primjerice: `mojadomena.com tvojadomena.com` itd. (domene odvojene razmakom) proširuju parametar `domain` |

Unesimo sâmo naše *DNS* poslužitelje (od našeg ***ISP-a***) u konfiguracijsku datoteku `/etc/resolv.conf` :

```
nameserver 195.29.150.3
nameserver 195.29.150.4
```

**Konfiguracija podrazumijevanog usmjerivača (*Default Gateway*) i imena računala (*hostname*)**
Konfiguracija *Default Gateway-a* uz još nekoliko osnovnih parametara nalazi se u konfiguracijskoj datoteci: `/etc/sysconfig/network`.

Ovdje možemo pronaći još nekoliko parametara, pa pogledajmo i koje:

| Parametar | Vrijednost | Opis |
|---|---|---|
| `NETWORKING=` | `yes` ili `no` | Uključuje li se globalno (za cijelo računalo) mreža (`yes`) ili se **ISKLJUČUJE** (`no`). |
| `HOSTNAME=` | Ime računala (*Hostname*) | Ime ovog računala: primjerice `server1` |
| `GATEWAY=` | IP adresa *Default Gateway*-a | IP adresa podrazumijevanog usmjerivača; primjerice: `192.168.1.1` |
| `GATEWAYDEV=` | Ime mrežnog sučelja; pr. `eth0` | Mrežno sučelje kao *Gateway*. Koristi se sâmo, ako imamo više mrežnih sučelja (kartica) na istoj mreži (*subnetu*) te želimo forsirati baš jedno od njih. Inače se ova opcija ne koristi jer sustav sâm odabire ispravno mrežno sučelje na osnovu njegove IP adrese. |

**Primjer dodavanja sljedećih parametara:**

- Ime računala (engl. *Hostname*): `server1`
- Podrazumijevani usmjerivač (*Default Gateway)* s IP adresom: `192.168.1.1`

Sada će datoteka `etc/sysconfig/network` izgledati ovako:

```
NETWORKING=yes
HOSTNAME=server1
GATEWAY=192.168.1.1
```



*Restartajmo računalo i potom provjerimo je li se sve ispravno konfiguriralo* kako smo definirali.

**Provjera IP adrese i maske mreže (*Netmaska*)**

Provjeru IP adrese i maske mreže radimo s naredbom `ifconfig` ili s novijom naredbom: `ip addr show`

```
ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 12:22:73:22:28:11
          inet addr:192.168.1.10  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
          RX packets:536950 errors:0 dropped:0 overruns:0 frame:0
          TX packets:572702 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:53114213 (50.6 MiB)  TX bytes:76191038 (72.6 MiB)
```

Pogledajmo redak: `inet addr:192.168.1.10  Bcast:192.168.1.255  Mask:255.255.255.0`

Iz njega vidimo kako je sve (IP adresa i maska mreže) konfigurirano kako smo i zamislili.

**1. Provjera podrazumijevanog usmjerivača (*Default Gateway-a).***

Provjeru *Default Gateway-a* možemo vidjeti s naredbama: `netstat -rn` i `ip route`. Provjerimo sve sa `netstat -rn`

```
netstat -rn

Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG        0 0          0 eth0
```

Opis: `Destination 0.0.0.0` i `Genmask (Netmask): 0.0.0.0` označavaju podrazumijevanu rutu odnosno rutu za *Default Gateway* jer govori: za sve odredišne IP adrese (`0.0.0.0`) sa svim maskama mreže (`0.0.0.0`) koristiti podrazumijevan usmjerivač (*Default gateway*): `192.168.1.1`. Napravimo istu provjeru i s novijom naredbom: `ip route`

```
ip route

192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.10
default via 192.168.1.1 dev eth0
```

Ovdje je sve jasno: *podrazumijevana ruta* (`default`), pokazuje na *Default Gateway:* `192.168.1.1` kako smo i htjeli.

**2. Provjera DNS poslužitelja**

Pod pretpostavkom da smo u datoteci `/etc/resolv.conf` dodali ispravne DNS poslužitelje, njihov rad možemo provjeriti s dvije naredbe: `host` i `nslookup`.

Sada provjerimo domenu: **www.opensource-osijek.org** s naredbom `nslookup` na sljedeći način:

```
nslookup  www.opensource-osijek.org

Server:         195.29.150.3
Address:        195.29.150.3#53

Non-authoritative answer:
www.opensource-osijek.org       canonical name = opensource-osijek.org.
Name:   opensource-osijek.org
Address: 213.147.104.78
```

Vidimo da je kontaktiran ***DNS*** poslužitelj s IP adresom: `195.29.150.3` te da smo dobili odgovor kako je za domenu: `www.opensource-osijek.org` zadužena IP adresa: `213.147.104.78` *(u trenutku pisanja teksta).*



Ako koristimo ***Red Hat* 9+** tada se statička i dinamička konfiguracija rade pomoću **Network Manager** servisa i pripadajućih novih konfiguracijskih datoteka (u direktoriju: `/etc/NetworkManager/system-connections/`) ili ručno s naredbom `nmcli`. Za detalje pogledajte poglavlja:

**25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**
**26.1. Network Manager.**

**Kako postaviti permanentne (trajne) postavke rada mrežne kartice koje postavljamo s naredbom ethtool**
Kao što smo već vidjeti svaka pojedina mrežna kartica se konfigurira u zasebnoj datoteci unutar direktorija: /etc/sysconfig/network-scripts/ pa je tako konfiguracijska datoteka za eth0 mrežnu karticu, datoteka imena: ifcfg-eth0.

Pogledajmo konfiguraciju ove datoteke koja sadrži permanentne promjene vezane za mogućnosti konfiguracije koje inače dobivamo upotrebom naredbe ethtool, a koje se definiraju ovdje pod opcijom: ETHTOOL_OPTS="" odnosno unutar njenih navodnika.


Naime mnoge parametre rada mrežne kartice na niskoj razini, poput *brzine* i *dupleksa* moguće je postaviti jedino s naredbom ethtool.

Pogledajmo jedan takav primjer te ispišimo konfiguraciju za mrežno sučelje eth0:

```
cat /etc/sysconfig/network-scripts/ifcfg-eth0

DEVICE=enp0s2
BOOTPROTO=static
IPADDR=10.1.1.100
NETMASK=255.255.255.0
ETHTOOL_OPTS="-s eth0 speed 1000 duplex full autoneg off"
ETHTOOL_OPTS="$ETHTOOL_OPTS ; -G eth0 rx 1300 tx 4096"
```

Nas konkretno zanimaju sâmo zadnja dva retka koja smo dodali, a koji se odnose na postavke vezane za naredbu ethtool:

```
ETHTOOL_OPTS="speed 1000 duplex full autoneg off"
ETHTOOL_OPTS="$ETHTOOL_OPTS ; -G eth0 rx 1300 tx 4096"
```


Za rad s naredbom ethtool te primjere konfiguracije pogledajte poglavlje:
**25.7.10. Naredba ethtool.**

U prvoj opciji (retku) smo naložili mrežnoj kartici eth0 da postavi:

- Fiksno, ručno, brzinu na 1000 Mbps (1Gbps), potom *duplex* na *full* te da se isključi *autonegotiation*, a sve sa: speed 1000 duplex full autoneg off

U drugoj opciji smo rekli mrežnoj kartici eth0 da postavi:

- ***Rx*** *ring buffer* odnosno prijemnu međumemoriju mrežne kartice, na 1.300 (-G eth0 rx 1300).
- ***Tx*** *ring buffer* odnosno međumemoriju za slanje, mrežne kartice, na 4.096 (tx 4096).


Oprez: prvo proučite poglavlje:
**25.1.1 Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**

Sve opcije i parametre naredbe ethtool koje želimo da se aktiviraju tijekom svakog pokretanja sustava (ili mrežnog sučelja) moramo dodati na ovom mjestu. Ne zaboravimo da se svako mrežno sučelje odnosno mrežna kartica konfigurira u svojoj zasebnoj konfiguracijskoj (tzv. ***ifcfg***) datoteci: /etc/sysconfig/network-scripts/ifcfg-**XY** pri čemu je **XY** ime sučelja odnosno mrežne kartice.


Za **Red Hat 9+,** pošto se navedene **ifcfg** konfiguracijske datoteke (/etc/sysconfig/network-scripts/ifcfg-**XY**) više ne koriste, pogledajte kako se konfiguriraju mrežna sučelja:
**25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**
**26.1. Network Manager.**

**Izvori informacija: (1041),(K-14), man ethtool, man nmcli.**

### 25.6.5.2. Dinamička konfiguracija mreže korištenjem DHCP poslužitelja

Kao što smo već govorili, DHCP protokol se koristi za automatsku konfiguraciju mrežnih IP parametara mrežne kartice i sustava na strani klijenta. Dakle od ***DHCP*** poslužitelja dobivamo:

- IP adresu i masku mreže (*Netmask*).
- Podrazumijevani usmjerivač (*Default Gateway*) uređaj odnosno njegovu IP adresu.
- ***DNS*** poslužitelj(e) i mnogo drugih (opcionalnih) parametara (pr. ***NTP*** poslužitelj i slično).

*DHCP* parametri se definiraju na strani *DHCP poslužitelja* (*DHCP server*), a njih dohvaćamo na *DHCP* klijentu.
U ovom slučaju *DHCP* klijentu na našoj mrežnoj kartici (našeg računala). Drugim riječima sve IP parametre koje smo u prethodnom poglavlju ručno konfigurirali za svaku mrežnu karticu na računalu, poput: IP adrese i *Netmaska*, *Default Gatewaya*, DNS poslužitelja i slično, odnosno pojednostavljeno IP adresu i druge parametre našeg računala, moguće je konfigurirati na jednom centralnom mjestu, za sva računala na mreži. To mjesto je *DHCP* poslužitelj.
Nakon konfiguracije *DHCP* poslužitelja potrebno je pomoću *DHCP* klijenta "povući" te sve parametre s *DHCP* poslužitelja na svako računalo na mreži koje ima instaliran *DHCP* klijent.

**Kako to radi?**
Recimo da smo na strani *DHCP* poslužitelja definirali da će svi *DHCP* klijenti (računala) na mreži dobiti sljedeće parametre mreže:

- IP adresu iz opsega: 192.168.1.10 do 192.168.1.100 uz masku mreže (*Netmask*): 255.255.255.0.
- DNS Poslužitelje: 158.225.15.16 i 158.226.15.17.
- Kao podrazumijevani usmjerivač (*Default Gateway)* IP adresu: 192.168.1.1.

Kada prvo računalo na mreži pokrene *DHCP* klijent na svojoj mrežnoj kartici (pr. `eth0`) njemu će *DHCP* poslužitelj dodijeliti prvu slobodnu IP adresu iz opsega: 192.168.1.10 do 192.168.1.100 uz masku mreže (*Netmask*): 255.255.255.0 te će mu poslati i ostale parametre, koje će *DHCP* klijent postaviti na svoje računalo, za konkretno mrežno sučelje/karticu.

Ako je sve prošlo u redu, naše računalo sada ima ispravno konfigurirane IP parametre poput:

- *IP adresu računala* (pr. na `eth0` sučelju): 192.168.1.**10** i pripadajuću masku mreže: 255.255.255.0.
- Podrazumijevani usmjerivač (*Default Gateway*) je postavljen na IP adresu: 192.168.1.1.
- *DNS* poslužitelji koje možemo koristiti su postavljeni na: 158.225.15.16 i 158.226.15.17.

Ako neko drugo računalo na mreži također zatraži IP parametre od DHCP poslužitelja, on mu dodjeljuje sljedeću slobodnu IP adresu iz definiranog opsega (u našem slučaju bi to bila IP: 192.168.1.**11**). DHCP poslužitelj čuva ove rezervacije IP adresa za DHCP klijente određeno vrijeme, definirano na DHCP poslužitelju. Ovo vrijeme se zove "Engl. ***DHCP Lease Time***".
Ako je ovo vrijeme isteklo, a računalo kojemu je dodijeljena ova IP adresa nije ugašeno i DHCP klijent ispravno radi, dolazi do procesa osvježavanja (Engl. *Renew*) te isto računalo kontaktira DHCP poslužitelj i ponovno (osvježava) tj. dobiva tu istu IP adresu, ponovno u istom vremenu trajanja (Engl. *Lease Time*). Odnosno brojač počinje od početka.

Ovaj proces se stalno ponavlja, dokle god je računalo (DHCP klijent) aktivno odnosno uključeno. U slučaju kada je to vrijeme isteklo, a računalo (DHCP klijent) je ugašeno ili mu ne rade DHCP klijent ili mreža, oslobađa se i rezervacija IP adrese za to određeno računalo te postojeća IP adresa postaje slobodna za dodjeljivanje prvom sljedećem računalu. U bilo kojem trenutku naše računalo (DHCP klijent) može javiti DHCP poslužitelju da oslobađa svoju IP adresu (Engl. *Release*) koja potom na DHCP poslužitelju postaje dostupna bilo kojem drugom računalu (DHCP klijentu) za dodjeljivanje.



Za više detalja o radu ***DHCP*** protokola, pogledajte poglavlje:
**25.8.3. DHCP protokol.**

**Izvori informacija: (1041),(K-14), (719), RFC903 (RARP), RFC951 (BOOTP), DHCP: RFC1531, RFC1541, RFC2131.**

#### 25.6.5.2.1. Privremena konfiguracija korištenjem DHCP poslužitelja

U slučajevima kada privremeno (do *restarta* računala) želimo dobiti konfiguracijske (IP) parametre mreže s DHCP poslužitelja, možemo koristiti naredbu `dhclient`. Najčešći parametri naredbe `dhclient` vidljivi su u tablici:

| Parametar | Opis |
|---|---|
| `-r` | Oslobodi dodijeljenu IP adresu; zahtjev prema DHCP poslužitelju (engl. *Release*). |
| `-R` | Pošalji listu opcija koje DHCP klijent traži od DHCP poslužitelja. |
| `-H` | Pošalji DHCP poslužitelju *Hostname* ovog računala (klijenta). |
| `-F` | Pošalji DHCP poslužitelju puno ima domene (FQDN) ovog računala (klijenta). |
| `-v` | Detaljan ispis poruka (engl. *Verbose*). |

Standardan klijentov *DHCP* upit odnosno zahtjev za dodjelu IP parametara, prema *DHCP* poslužitelju ima sintaksu:
*`dhclient` mrežno-sučelje(interface)*
Primjerice ovako: `dhclient eth0`. Ako želimo dobiti IP parametre mreže od DHCP poslužitelja na `eth0` mrežnoj kartici našeg računala, dovoljno je pokrenuti sljedeću naredbu:

```
dhclient eth0
```

Nakon nekoliko trenutaka, naša mrežna kartica `eth0` dobiva sve parametre mreže, poput:

- IP adrese i maske mreže (*Netmaska*).
- Podrazumijevanog usmjerivača (*Default Gateway-a)* te listu DNS poslužitelja, kao i drugih parametara.

Osim ovog jednostavnog načina upotrebe DHCP klijenta, moguće je sve željene parametre i opcije DHCP klijenta spremiti u konfiguracijsku datoteku: `/etc/dhclient.conf`.

**Izvori informacija: (K-12),(K-14), `man dhclient`, `man dhclient.conf`.**

#### 25.6.5.2.2. Trajna konfiguracija mreže korištenjem DHCP poslužitelja

U prethodnim primjerima vidjeli smo kako se može privremeno konfigurirati DHCP klijent na našem računalu.
Sada ćemo naučiti kako konfigurirati DHCP klijenta (naše računalo) trajno. Dakle uređivanjem konfiguracijskih datoteka koje su zadužene za trajnu (i nakon restarta računala) konfiguraciju DHCP klijenta. Prvo provjerimo je li mreža uopće uključena odnosno aktivirana u datoteci: `/etc/sysconfig/network`. Dakle provjerimo imamo li ovdje sljedeći redak konfiguracije:

```
NETWORKING=yes
```

Ako je sve u redu kao u primjeru, idemo dalje. Uredimo konfiguracijsku datoteku za našu mrežnu karticu. U našem slučaju je to `eth0` pa je njena pripadajuća konfiguracijska datoteka: `/etc/sysconfig/network-scripts/ifcfg-eth0`.
Sve što je potrebno da bi naše računalo na mrežnoj kartici `eth0` radilo kao *DHCP* klijent su sljedeći redci konfiguracije:

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
```

Opis ispisa:

- `DEVICE` i `ONBOOT` smo već opisali.
- s `BOOTPROTO` naše računalo prebacujemo u *DHCP* klijent način rada i to konkretno s `BOOTPROTO=dhcp`. Dakle `BOOTPROTO=dhcp` aktivira DHCP klijenta na ovoj mrežnoj kartici (`eth0`) te on automatski kod pokretanja sustava, traži od DHCP poslužitelja IP parametre za ovu mrežnu karticu.

Ovo je sve što je potrebno da naše računalo i nakon restarta radi kako DHCP klijent odnosno da svaki puta tijekom restarta računala dobije sve IP parametre mreže s DHCP poslužitelja.
Za provjeru, možemo restartati računalo i vidjeti jesmo li dobili sve IP parametre potrebne za rad, s DHCP poslužitelja.
Provjerimo IP adresu i masku mreže naše `eth0` mrežne kartice sa naredbom:

```
ifconfig eth0
```

Ili isto možemo dobiti s novijom naredbom `ip` na sljedeći način:

```
ip address show eth0
```

Provjerimo i rute koje smo dobili; što možemo napraviti na nekoliko načina:

```
ip route
```

Ili pozovimo stariju naredbu:

```
netstat -rn
```

U slučajevima kada želimo definirati dodatne opcije našeg DHCP klijenta, što u većini slučajeva nije potrebno, moramo urediti datoteku: `/etc/dhcp/dhclient-eth`**`x`**`.conf` odnosno za mrežnu karticu `eth0` će to biti datoteka imena: `/etc/dhcp/dhclient-eth0.conf`.

**Izvori informacija: (K-14), `man ip route`, `man ifconfig`, `man ip address`, `man 5 dhclient.conf`.**

### 25.6.6. Rad s mrežnim servisom

U slučajevima kada smo nešto mijenjali na mreži i želimo ju restartati, bez restarta cijelog računala, to možemo napraviti tako da restartamo centralni mrežni servis, koji će u pozadini restartati i sva mrežna sučelja te ih ponovno rekonfigurirati, prema definiranim pravilima navedenim u njihovim konfiguracijskim datotekama (o kojima smo već govorili).
*Za **RedHat/CentOS*** **6.x** (u *RedHat/CentOS* **6.x** mrežni servis je zapravo skripta `/etc/init.d/network`)

```
service network restart
```

Naime u starijim inačicama *Red Hat Enterprise Linuxa* tj. *CentOS* Linuxa (do v. **6.x**) koristile su se *init* skripte (**SystemV** skripte) koje se nalaze u direktoriju `/etc/rc.d/init.d/`. Ove *init* skripte obično su napisane u *Bash* ljusci i dopuštale su administratoru sustava da kontrolira stanje servisa (Engl. *demona*) na sustavu, odnosno da ih pokreće i/ili zaustavlja.
Na ***RedHat/CentOS*** **7.x**, ove **SystemV** *init* skripte zamijenjene su s takozvanim *Systemd unit* skriptama, koje koristi vršni servis ***Systemd***. Stoga na ***RedHat/CentOS*** **7.x** mrežni servis restartamo na sljedeći način:

```
systemctl restart network
```



Iako već i od ***RedHat/CentOS*** **7.x** za cjelokupnu mrežnu funkcionalnost postoji servis ***Network Manager***, njegova puna funkcionalnost je dovršena tek od inačice ***8.x RedHat/CentOS Linuxa***.

## RedHat/CentOS 8.x+

Naime u ***RedHat/CentOS 8.x*** je uvedena nova varijanta servisa *Network Manager*, koji u potpunosti mijenja prijašnji servis `network`, ali i način rada sustava, po pitanju mreže. Štoviše, od ***RedHat/CentOS 8.x*** više ne postoji niti *Systemd unit* skripta `network`. To možemo vrlo lako provjeriti; pokretanjem sljedeće naredbe:

```
systemctl restart network

Failed to restart network.service: Unit network.service not found.
```

Vidimo da smo dobili poruku kako *Systemd unit* skripta imena `network`, više ne postoji!.



*Od **RedHat/CentOS 8.x** više ne postoji niti Systemd unit skripta* `network`*, već je za cjelokupni rad mreže, sada zadužen Systemd servis imena:* ***`NetworkManager`***.

Ako ga želimo pokrenuti (*Network Manager*), to možemo učiti sa sljedećom naredbom (iako je on standardno pokrenut):

```
systemctl start NetworkManager
```

Odnosno ako ga trajno želimo aktivirati (što bi isto tako trebala biti standardna postavka), to možemo postići sa:

```
systemctl enable NetworkManager
```

Ako smo mijenjali parametre mreže za to više nećemo trebati restartati niti novi servis *NetworkManager* već sa sljedećim naredbama možemo, spustiti (ugasiti) mrežu te ju ponovno podići. To moramo napraviti stoga jer će se prilikom pokretanja mreže (ne i mrežnog servisa), pročitati i postaviti njena nova konfiguracija.

Sve to postiženo s novom naredbom `nmcli` koja komunicira sa servisom *NetworkManager* koji to sve u pozadini i odrađuje.

```
nmcli networking off
nmcli networking on
```

Važno je ne zaboraviti da u ***ifcfg*** datoteci moramo imati postavljeno: `NM_CONTROLLED=yes`.



Vezano za servis *NetworkManager*, pogledajte sljedeće poglavlje, ali i poglavlje:
**26.1. Network Manager.**

**Izvori informacija: (1042),(1043),(K-14)**, **`man service`**, **`man systemctl`**, **`man nmcli`**.

#### 25.6.6.1. Konfiguracija mreže upotrebom naredbe `nmcli` i *NetworkManager* servisa

Nova značajka ***Red Hat Enterprise Linuxa 7*** i srodnih distribucija Linuxa, jest da je za mrežne usluge sada zadužen servis ***NetworkManager***. Njegov osnovni zadatak je dinamička kontrola mreže i mrežne konfiguracije, tako da on pokušava održati mrežna sučelja i veze aktivnima, kada su dostupne, a da i dalje podržava tradicionalne konfiguracijske datoteke tipa ***ifcfg*** (pogledajte prijašnja poglavlja). ***NetworkManager*** može se koristiti s vrstama mrežnih sučelja poput: ***Ethernet***, ***VLAN***, ***Bridge***, ***Bond/Team***, ***Wi-Fi*** i drugih. Za ove vrste mrežnih sučelja *NetworkManager* može konfigurirati: IP adrese i parametre veze, statičke rute, DNS informacije te VPN veze, kao i mnoge parametre specifične za svaku mrežnu vezu odnosno mrežno sučelje. Za rad odnosno konfiguraciju mrežnih sučelja i veza se koristi naredba: `nmcli`, koja se za rad oslanja na servis *NetworkManager*. *NetworkManager* pruža i D-Bus ***API*** (aplikacijsko programsko sučelje), koji možete koristiti i ako razvijate svoje aplikacije.
Provjerimo je li servis *NetworkManager* uredno pokrenut, sa sljedećom naredbom:

```
systemctl status NetworkManager

● NetworkManager.service - Network Manager
   Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled; vendor
preset: enabled)
   Active: active (running) since Tue 2021-10-12 11:37:57 CEST; 10min ago
```

Vidimo da je servis *NetworkManager* pokrenut (`loaded`) i aktiviran (`enabled`) tijekom svakog pokretanja sustava.
U slučaju kada se servis *NetworkManager* nije pokrenuo ili ga je potrebno restartati, to možemo napraviti sa:

```
systemctl restart NetworkManager
```



Za sve ostale radnje (ručno i trajno pokretanje, zaustavljanje i sl.) prema ovom ili bilo kojem drugom servisu čiji rad kontrolira ***Systemd***, pogledajte primjere u poglavlju: **7.3.2.2. *Systemd* - rad sa servisima (*daemonima*).**



Jedna od zanimljivosti naredbe **`nmcli`** je i njena mogućnost dopunjavanja parametara pomoću tipke ***TAB***, u slučaju kada ste zaboravili sintaksu njenih parametara i opcija ili želite vidjeti koji su sve parametri i opcije dostupni za upotrebu.



Naredba **`nmcli`** odnosno poznavanje njenog rada je korisno i stoga jer se ona sada (2021.g.) koristi na većini drugih distribucija Linuxa, poput **Debian** baziranih distribucija (pr. **Ubuntu**, **Armbian**, **Proxmox Virtual Environment**, …).

Prvo pogledajmo status naših mrežnih sučelja ili mrežnih kartica. Skratili smo ispis na samo jedno mrežno sučelje:

```
nmcli device status
DEVICE  TYPE      STATE      CONNECTION
enp0s3  ethernet  connected  enp0s3
```

Ovdje vidimo da je naše mrežno sučelje `enp0s3` aktivno, odnosno da je u stanju (`connected`),„*spojeno*“, dakle u upotrebi je.

Potom pogledajmo koje sve aktivne mrežne profile s pripadajućim mrežnim sučeljima imamo kreirane na sustavu, a da se za njih brine servis *NetworkManager*:

```
nmcli connection show
NAME    UUID                                  TYPE      DEVICE
enp0s3  3c36b8c2-334b-57c7-91b6-4401f3489c69  ethernet  enp0s3
```

Ovdje vidimo da je naše mrežno sučelje (`DEVICE`) `enp0s3` vrste: `ethernet` te vidimo njegovu jedinstvenu ***UUID*** oznaku. Pod `NAME` se navodi ime mrežnog profila (konfiguracije) te na koje mrežno sučelje (pod `DEVICE`) se primjenjuje.



***Moguće je imati i više mrežnih profila, čak i za isto mrežno sučelje, sve prema vašim potrebama!***



Važno je razumjeti da se konfiguracija definira unutar mrežnog profila, za željeno mrežno sučelje.

Pogledajmo i kako odspojiti naše mrežno sučelje (mrežnu karticu): `enp0s3`

```
nmcli device disconnect enp0s3
```

Odnosno pogledajmo kako ga ponovno spojiti:

```
nmcli device connect enp0s3
```

## *Network Manager* i *nmcli*: trajna (perzistentna) konfiguracija mrežnog sučelja.

U ovom primjeru smo dodali novu mrežnu karticu na računalo (`enp0s3`) te ćemo nju i konfigurirati.
Kreirajmo mrežni profil imena `NAT` te mu dodajmo i konfigurirajmo naše mrežno sučelje: `enp0s3`. Potom mu dodijelimo IP adresu (`ip4`) i podrazumijevani usmjerivač odnosno „*Default Gaetway*“ (`gw4`). To sve ćemo postići sa sljedećom naredbom:

```
nmcli connection add type ethernet con-name NAT ifname enp0s3 ip4 192.168.10.10/24
gw4 192.168.10.1
```

Sada za ovaj mrežni profil imena (`NAT`), za ovo mrežno sučelje (`enp0s8`), možemo konfigurirati DNS poslužitelje:

```
nmcli connection modify NAT ipv4.dns "8.8.8.8 8.8.4.4 1.1.1.1"
```



Čim smo to napravili, `nmcli`, odnosno ***NetworkManager*** će kreirati mrežnu ***ifcfg*** skriptu za inicijalizaciju mreže, sljedećeg imena*: `/etc/sysconfig/network-scripts/ifcfg-NAT`. Sâmim time ona će biti aktivna kod sljedećeg restarta računala. Dakle ovdje je rad ***NetworkManager-a*** prilagođen našoj distribuciji Linuxa (***RedHat/CentOS***).



Druge distribucije Linuxa mogu imati izvorno ponašanje ***NetworkManager-a***; pa će se konfiguracije snimati u datoteke unutar direktorija: `/etc/NetworkManager/system-connections/` ili prilagođeno ponašanje poput našeg za **RedHat/CentOS** distribucije Linuxa, koje sve snimaju u takozvane *ifcfg* datoteke u direktorij: `/etc/sysconfig/network-scripts/`.



Za sve ostale detalje o zadanom ponašanju *NetwokManager* servisa i mrežnih konfiguracijskih datoteka, pogledajte: **26.1. Network Manager.**

Na kraju podignimo ovo mrežno sučelje; koje je definirano u mrežnom profilu `NAT`:

```
nmcli connection up NAT ifname enp0s3
```

Ako želimo vidjeti što je sada sve konfigurirano (i dostupno za konfiguraciju), za ovaj mrežni profil (`NAT`) i njegovo sučelje:

```
nmcli connection show NAT
```

I vidjet ćemo desetke opcija koje su konfigurirane za naše mrežno sučelje.

Pogledajmo sâmo neke od korisnih konfiguracijskih opcija koje možemo koristiti:

| Konfiguracijska opcija i trenutna `vrijednost` | Opis opcije |
|---|---|
| **connection.id** `enp0s3` | Ime mrežnog sučelja (veze). |
| **connection.autoconnect** `yes` | Automatsko spajanje kod detektiranja dostupne (aktivne) veze.<br>Vrijednosti su `yes` i `no`. Ova *Nmcli* opcija je `autoconnect` |
| **802-3-ethernet.speed** `0` | Brzina rada mrežnog sučelja **0**=nije striktno definirano.<br>Brzine su: 10, 100, 1000, 10000, … |
| **802-3-ethernet.duplex** | Definicija *duplex* načina rada.<br>Vrijednosti su: `full` ili `half`. |
| **802-3-ethernet.auto-negotiate** | Definicija koristi li se automatsko dogovaranje parametara veze (*Auto-negotiate*).<br>Vrijednosti su `yes` i `no` (ili nedefinirano). |
| **ipv4.addresses:** `192.168.10.10/24` | Ovdje se definira IP (IPv4) adresa mrežnog sučelja.<br>Ova *Nmcli* opcija je: `ip4` |
| **ipv4.gateway:** `192.168.10.1` | Ovdje se definira IP (IPv4) adresa podrazumijevanog usmjerivača (***Default Gateway***). Ova *Nmcli* opcija je: `gw4` |

Pogledajmo kako za naše mrežno sučelje promijeniti parametre brzine rada na: *1000Mbps*, *full duplex*, *auto-negotiate*:

```
nmcli con modify LAN 802-3-ethernet.speed 1000 802-3-ethernet.duplex full 802-3-
ethernet.auto-negotiate yes
```

Potom moramo isključiti pa podići ovo mrežno sučelje; kako bi se promjene parametara brzine rada aktivirale:

```
nmcli con down LAN
nmcli con up LAN
```

U slučaju kada smo ručno mijenjali konfiguracijske datoteke*, potrebno ih je osvježiti s `nmcli` alatom, na sljedeći način:

```
nmcli connection reload
```

## Upoznat ćemo se s detaljima konfiguracije VLAN, BRIDGE i BOND sučelja

**Primjeri koji slijede odnose se na nativno ponašanje *NetworManager* servisa to jest *NetworManager* konfiguracijske datoteke odnosno ne na takozvane *ifcfg* datoteke (iako se mogu koristiti i one).**

**Nmcli i VLAN sučelja**
Pomoću *NetworkManager* servisa i pripadajuće naredbe `nmcli` moguće je direktno kreirati i VLAN mrežna sučelja.
Pogledajmo kako na mrežnoj kartici `enp0s3` kreirati VLAN sučelje koje pripada VLAN 30 mreži.

```
nmcli con add type vlan con-name VLAN30 dev enp0s3 id 30
```

Ovim korakom smo kreirali profil naziva `VLAN30`, za novo VLAN sučelje, kako smo objasnili. Nakon toga pojavit će se datoteka (za ***Red Hat 9+***): `/etc/NetworkManager/system-connections/VLAN30.nmconnection` koja će sadržavati sljedeće sekcije konfiguracije:

```
[connection]
id=VLAN30
uuid=3194d94f-a88f-466c-914c-25a7b469bcb6
type=vlan

[ethernet]

[vlan]
flags=1
id=30
parent=enp0s3

[ipv4]
method=auto
```

U ispisu vidimo da se radi o VLAN sučelju (`type=vlan`) koje pripada VLANu broj 30 (`id=30`) te da je ono vezano za mrežnu karticu `enp0s3` (`parent=enp0s3`). Pogledajmo koja sve mrežna sučelja sada imamo aktivna:

```
nmcli connection show
NAME                    UUID                                  TYPE      DEVICE
VLAN30                  aa2c3986-9011-42db-83e6-70fc9695c267  vlan      enp0s8.30
enp0s3                  e11206ee-a40c-4241-a6cc-ba2cd3777beb  ethernet  enp0s3
```

← Vidimo `VLAN30` mrežni profil, koji je u pozadini kreirao sistemsko mrežno sučelje naziva: `enp0s8.30` što je očekivano najnižoj razini Linuxa.



Za detalje oko **VLAN** mreža i mrežnih sučelja, pogledajte poglavlje:
**20.6.2. VLANovi odnosno virtualne lokalne mreže.**

**Nmcli i Bridge sučelja**
Pomoću *NetworkManager* servisa i pripadajuće naredbe `nmcli` moguće je direktno kreirati i ***bridge*** mrežna sučelja.
Pogledajmo kako kreirati *bridge* sučelje `br0` i na njega spojiti mrežnu karticu `enp0s3`:

```
nmcli con add type bridge ifname br0 con-name br0
nmcli con add type bridge-slave ifname enp0s3 master br0
```

Dakle ovim korakom smo kreirali dva profila naziva `br0` i `bridge-slave-enp0s3` koji će se pojaviti kao datoteke (za ***Red Hat 9+***) unutar direktorija: `/etc/NetworkManager/system-connections/`.

Ove konfiguracijske datoteke će sadržavati:

<table>
<tr><th>br0.nmconnection</th><th>bridge-slave-enp0s3.nmconnection</th></tr>
<tr><td><pre>[connection]
id=bridge-br0
uuid=ccde003b-273f-4835-a80b-
992d88c1fd9e
type=bridge
interface-name=br0



[ethernet]

[bridge]

[ipv4]
method=auto</pre></td><td><pre>[connection]
id=bridge-slave-enp0s3
uuid=8dd37c7a-eff1-46d6-8a45-
bd378fd9f506
type=ethernet
interface-name=enp0s3
master=br0
slave-type=bridge

[ethernet]

[bridge-port]</pre></td></tr>
</table>

Pogledajmo *bridge* `br0` sučelje i vezu sa sučeljem `enp0s3`:

```
nmcli connection show
NAME                 UUID                                  TYPE      DEVICE
br0                  f2638395-867a-41b3-8f22-4a01b599bd07  bridge    br0
bridge-slave-enp0s8  42fe3ec1-9dd9-4fd3-8e62-20033db1e62e  ethernet  enp0s3
```

← Vidimo dva mrežna profila: `br0` i `bridge-slave-enp0s3` koji su u pozadini kreirali sistemsko mrežno sučelja naziva: `br0` i povezali ga s mrežnom karticom `enp0s3`:, što možemo na razini sustava vidjeti i ovako:

```
bridge link show br0
3: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding
priority 32 cost 100
```

← Ovdje vidimo tu vezu `br0` i mrežnog sučelja (kartice) `enp0s3`, to jest pripadnost mrežne kartice mrežnom sučelju `br0`.



Za detalje oko **Bridge** mrežnih sučelja, pogledajte poglavlje:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**

**Nmcli i Bond sučelja**

Pomoću *NetworkManager* servisa i pripadajuće naredbe `nmcli` moguće je direktno kreirati i ***bond*** mrežna sučelja.
Pogledajmo kako kreirati *bond* sučelje `bond0` i unutar tog logičkog sučelja spojiti fizičku mrežnu karticu `enp0s3`:

```
nmcli con add type bond ifname bond0 con-name bond0 bond.options "mode=balance-rr,miimon=100"
nmcli con add type ethernet ifname enp0s3 master bond0
```

Dakle ovim korakom smo kreirali dva profila naziva `bond0` i `bond-slave-enp0s3` koji će se pojaviti kao datoteke (za ***Red Hat 9+***) unutar direktorija: `/etc/NetworkManager/system-connections/` koje će sadržavati:

| bond0.nmconnection | bond-slave-enp0s3.nmconnection |
|---|---|
| [connection]<br>id=bond0<br>uuid=ac07b997-5cda-4762-b9b1-6e79a2988b49<br>type=bond<br>interface-name=bond0<br><br>[bond]<br>miimon=100<br>mode=balance-rr<br><br>[ipv4]<br>method=auto | [connection]<br>id=bond-slave-enp0s3<br>uuid=5d99efb8-ff7e-4fd9-9583-41b786823af0<br>type=ethernet<br>interface-name=enp0s3<br>master=bond0<br>slave-type=bond<br><br>[ethernet]<br><br>[bond-port] |

Pogledajmo stanje novih veza

```
nmcli connection
NAME               UUID                                  TYPE      DEVICE
bond0              6cb411e6-82c7-4975-8c44-7e5230ea9b48  bond      bond0
bond-slave-enp0s3  8d843bb5-b69c-4739-a212-498716be1a8c  ethernet  enp0s3
```

← vidimo `bond0` logičko mrežno sučelje te fiziku mrežnu karticu `enp0s3` unutar njega.

Za dodavanje druge mrežne kartice (pr. `enp0s9`) u `bond0`, možemo pokrenuti naredbu:

```
nmcli con add type ethernet ifname enp0s9 master bond0
```

Za dodatnu aktivaciju mrežnih kartica `enp0s3` i `enp0s9` te `bond0` sučelja možemo pokrenuti

```
nmcli con up bond-slave-enp0s3
nmcli con up bond-slave-enp0s9
nmcli con up bond0
```



Za detalje oko ***Bond*** mrežnih sučelja, pogledajte poglavlje:
**20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).**



Za detalje rada *NetworkManager* servisa te postavke mrežnih konfiguracijskih datoteka pogledajte poglavlje:
**26.1. Network Manager.**

**Izvori informacija: (944),(945),(946),(1042),(1043),(1343),(K-14),** **`man nmcli`**, **`man systemctl`**.

## 25.6.7. Konfiguracija pravila usmjeravanja (*ruta*)

### 25.6.7.1. Privremena konfiguracija statičkih ruta s naredbama *route* i *ip route*

U slučajevima kada je privremeno potrebno dodavati nova pravila usmjeravanja odnosno *rute (Engl. route)* koristimo se naredbom `route` ili `ip route`. Ove promijene nisu trajne i traju sâmo do restarta računala. Iste naredbe se koriste i za ispis trenutnih *ruta* na sustavu. Statičke *rute* su sve *rute* koje smo dodali ručno pomoću neke naredbe ili preko konfiguracijske datoteke. Dakle sve one koje nismo dobili pomoću nekog (*ruting)* protokola za usmjeravanje, poput: *RIP*,*OSPF*, *BGP*, … .
Slijede primjeri. Ispišimo sve *rute* na sustavu, u brojčanom obliku (`-n` prekidač) :

```
route -n

Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.17.100.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
0.0.0.0         172.17.100.254  0.0.0.0         UG    0      0        0 eth0
```

Vidimo da je za podrazumijevanu rutu (*default route*), koju predstavljaju: `Destination: 0.0.0.0` i `Netmask: 0.0.0.0`, zadani pristupnik (*Default Gateway)* uređaj s IP adresom: 172.17.100.254 do kojeg dolazimo preko našeg mrežnog sučelja: `eth0`. Pogledajmo i metodu ispisa istih informacija pomoću naredbe `ip route`

```
ip route

172.17.100.0/24 dev eth0  proto kernel  scope link  src 172.17.100.10
default via 172.17.100.254 dev eth0
```

## Kako dodati zadani pristupnik odnosno podrazumijevani usmjerivač (*Default Gateway**)

Podrazumijevani usmjerivač *(Default gateway)* koji će biti trajno (*permanentno*) konfiguriran, konfigurira se u datoteci: `/etc/sysconfig/network` u varijabli `GATEWAY=`
Privremena konfiguracija podrazumijevanog usmjerivača odrađuje se sa sljedećom sintaksom s naredbom `route`:
`route add default gw <IP adresa Default Gateway-a>`
Odnosno isto možemo postići pomoću novije naredbe `ip route` sa sljedećom sintaksom:
`ip route add default via <IP adresa Default Gateway-a>`

Pogledajmo primjer ovakve konfiguracije kada želimo dodati *default gateway* s IP adresom: `172.17.100.254`

```
route add default gw 172.17.100.254
```

Odnosno isto možemo postići s novijom naredbom `ip route` na sljedeći način:

```
ip route add default via 172.17.100.254
```

## Dodavanje i brisanje *rute*

Kod dodavanja *rute*, mogu se dodavati *rute* prema mrežama (engl. *Network*) ili pojedinim računalima (engl. *Host*).
Sintaksa za dodavanje (Engl. *Add*) *rute* prema **mrežama** je sljedeća:
`route add -net <IP adresa mreže> netmask <Netmask od te mreže> gw <IP adresa Gatewaya te mreže>`
Odnosno isto možemo dobiti s novijom naredbom:
`ip route add <IP adresa mreže>/<CIDR oblik netmaska> via <IP adresa Gatewaya te mreže>`

Sintaksa za brisanje (engl. *Delete*) *rute* prema mrežama je sljedeća:
`route del -net <IP adresa mreže> netmask <Netmask od te mreže> gw <IP adresa Gateway-a te mreže>`

**Primjeri**
Dodajmo rutu za mrežu: 192.168.100.0 /255.255.255.0, da ide preko *Gateway-a* s IP adresom*: 172.17.100.252.

```
route add -net 192.168.100.0 netmask 255.255.255.0 gw 172.17.100.252
```

Odnosno napravimo isto s novijom naredbom:

```
ip route add 192.168.100.0/24 via 172.17.100.252
```

U slučaju kada nam više ne treba ruta koju smo upravo dodali, možemo ju obrisati na sljedeći način:

```
route del -net 192.168.100.0 netmask 255.255.255.0 gw 172.17.100.252
```

Odnosno napravimo isto s novijom naredbom:

```
ip route del 192.168.100.0/24 via 172.17.100.252
```

## Testiranje rute

Ako imamo veći broj *ruta* i više nismo sigurni preko koje *rute* će naš mrežni paket prolaziti, na raspolaganju nam je dodatni alat: `ip route get`. Ovaj alat će nam simulirati prolazak paketa kroz našu *ruting* tablicu odnosno tablicu usmjeravanja i prikazati odabranu rutu.

**Primjer:** Provjerimo preko koje rute će se s našeg računala doći do mreže 192.168.100.0/24:

```
ip -s route get 192.168.100.0/24

192.168.100.0 via 172.17.100.252 dev eth0  src 172.17.100.10
```

Opis: do mreže `192.168.100.0/24` se dolazi preko rute koja prolazi preko usmjerivača* s IP adresom `172.17.100.252`.

**Izvori informacija:** **`man route`**, **`man ip route`**.

### 25.6.7.2. Trajna konfiguracija statičkih ruta

Trajnu konfiguraciju statičkih ruta postižemo dodavanjem rute na razini svake mrežne kartice (engl. *interface*) u posebnu datoteku, specifičnu za određenu mrežnu karticu/sučelje. Tako ćemo trajne statičke rute za `eth0` mrežnu karticu snimati u datoteku `/etc/sysconfig/network-scripts/route-eth0`. Sadržaj ove datoteke prati sintaksu naredbe `ip route`. Sintaksa je: `Mreža/CIDR netmask` **`via`** `<router/gateway za tu mrežu>` **`dev`** `<Mrežna kartica>`



**Napomena**: statičke rute, privremene ili trajne, dodaju se samo za specifične mreže do kojih se ne može doći preko podrazumijevanog usmjerivača (pristupnika) odnosno takozvanog *Default Gatewaya*.

**Primjer**
Dodajemo statičku trajnu rutu za mrežu 10.10.10.0/24 do koje dolazimo preko IP adrese (*routera*/usmjerivača*) 172.17.100.222, preko mrežne kartice `eth0`.
Stoga kreirajmo i uredimo datoteku `/etc/sysconfig/network-scripts/route-eth0` te dodajmo sljedeći redak:

```
10.10.10.0/24 via 172.17.100.222 dev eth0
```

Sâmo mrežno sučelje (`eth0`) se podrazumijeva jer uređujemo datoteku specifičnu za ovo mrežno sučelje, pa je dovoljno i:

```
10.10.10.0/24 via 172.17.100.222
```

Nakon novog unosa, dovoljno je pokrenuti proceduru za restart mreže (***RedHat/CentOS 6***), kako bi se dodane *rute* aktivirale: `service network restart` odnosno za ***RedHat/CentOS 7.x***: `systemctl restart network`

Za ***RedHat/CentOS 8.x***, osvježiti ih možemo s `nmcli` alatom, na sljedeći način (bez restarta mrežnog servisa):

```
nmcli connection reload
```

Isto tako spuštanjem (gašenjem) i podizanjem (pokretanjem/paljenjem) ovog mrežnog sučelja (`eth0`) novo dodana *ruta* će se pročitati i aktivirati za konkretno mrežno sučelje. Sada prvo spustimo, pa podignimo mrežno sučelje `eth0` sa:

```
ifdown eth0
ifup eth0
```



I nakon restarta računala sve rute dodane u datoteke: `/etc/sysconfig/network-scripts/route-`**`XY`** će također ostati aktivne.

Sada, nakon restarta mrežnog sučelja, mreže ili računala, rute možemo provjeriti sa slijedećim naredbama:

```
netstat -rn
```

Ili napravimo isto, ali s novijom naredbom:

```
ip route show
```

****Gateway** označava pristupni uređaj do odredišne mreže.*

## RedHat/CentOS 8+

***Na RedHat/CentOS 8.x***, pošto se većina skripti unutar direktorija: `/etc/sysconfig/network-scripts/` više ne koristi, za rad s pravilima usmjeravanja jer se koristi novi ***Network Manger***, preporuča se koristiti novi program: `nmcli`.
Osnovni trajni način dodavanja rute bi sada bio primjerice za sučelje `enp0s17` ako za mrežu 192.168.122.0/24 želimo definirati IP adresu usmjerivača: 192.18.1.222; odnosno da se preko njega pristupa toj mreži:

```
nmcli connection edit enp0s17
set ipv4.routes 192.168.122.0/24 192.168.1.222
save persistent
quit
nmcli connection up enp0s17
```

Navedena konfiguracija će biti snimljena u novu datoteku: `/etc/sysconfig/network-scripts/route-enp0s17`**.** Međutim na starim sustavima (***RedHat/CentOS 6/7***), zapis unutar navedene datoteke će biti malo drugačiji od gore navedenog, ako koristimo `nmcli` za snimanje. Međutim i na novom sustavu možemo koristiti staru sintaksu u gore navedenoj datoteci*, ako ju ručno mijenjamo, kao što smo to radili na starijim inačicama (***RedHat/CentOS 6/7***).

**Izvori informacija: (910), `man nmcli`,`man nm-settings`, `man nmcli-examples`.**

### 25.6.8. Ruting i optimizacija ruting parametara

Vratimo se malo unatrag; bilo koje računalo ili uređaj koji je u stanju prihvatiti i prosljeđivati pakete (na OSI sloju tri) između dvije mreže, smatramo usmjerivačem (*routerom*). Svaki usmjerivač mora imati barem dva mrežna sučelja; jedno sučelje se spaja na jednu mrežu, a drugo sučelje spaja se na drugu mrežu. Pod mrežnim sučeljem obično podrazumijevamo zasebne mrežne kartice, iako to mogu biti i virtualna sučelja, kao što je primjerice *VLAN* sučelje. Računala ili mrežni uređaji povezani s bilo kojom mrežom, informacije o susjednim mrežama dobivaju pomoću ruta, a njih mogu dobiti pomoću protokola usmjeravanja (Engl. *Routing protocol*) ili statičkom konfiguracijom odnosno ručnim dodavanjem ruta. Naime svako računalo pa tako i usmjerivač na osnovu tablice usmjeravanja (Engl. *Routing Table*) odlučuje gdje, koji mrežni paket mora proslijediti (poslati). Kada Linux radi kao usmjerivač to mu omogućuje da prihvaća mrežne pakete na jednom sučelju i prenese ih na drugo mrežno sučelje. Proces prihvaćanja paketa s jedne strane (s jednog mrežnog sučelja) i slanja paketa na drugu stranu (na drugo mrežno sučelje) poznat je kao prosljeđivanje (Engl. ***Forwarding***).
Mnoge mrežne funkcionalnosti za svoj rad traže ovakvu mogućnost prosljeđivanja mrežnih paketa.



Standardno Linux računala s više mrežnih sučelja (kartica) nemaju uključeno prosljeđivanje odnosno ***forwarding*** mrežnih paketa s jednog sučelja na drugo, pa se stoga standardno ne mogu ponašati kao usmjerivači.

Mogućnost prosljeđivanje mrežnih paketa odnosno ponašanje kao usmjerivača se definira u *sysctl* varijabli `net.ipv4.ip_forward` odnosno u */proc* datoteci: `/proc/sys/net/ipv4/ip_forward`.

Ako želimo privremeno uključiti ovu funkcionalnost, kako bi Linux postao usmjerivač, to možemo napraviti sa:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Za trajnu aktivaciju ove konfiguracije možemo napraviti sljedeći redak unosa u datoteci: `/etc/sysctl.conf`:

```
net.ipv4.ip_forward = 1
```

Ovime smo pretvorili naše linux računalo u usmjerivač. Nakon ovog koraka obično je potrebno napraviti optimizacije koje smo objasnili u prijašnjim cjelinama unutar vršnog poglavlja: **24 Transportni protokoli (OSI sloj 4)**.

**Vratimo se rutama**
Pogledajmo kako vidjeti ispis ruta odnosno tablice usmjeravanja na našem računalu s naredbom `route`. Mada smo isto mogli vidjeti i s naredbama: `ip route` i `netstat -r`, ali naredba `route` to ipak prikazuje malo ljepše:

```
route -n

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.17.100.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
0.0.0.0         172.17.100.254  0.0.0.0         UG    0      0        0 eth0
```

Iz ovog primjera vidimo kako svaki sustav mora imati barem nekoliko ruta, koje govore sustavu gdje slati pakete, ovisno o odredištu. Naime svaki puta kada mrežni sloj mora poslati svaki pojedini paket na neku odredišnu IP adresu, on prethodno mora pogledati u svoju tablicu usmjeravanja, kako bi znao na koje mrežno sučelje poslati taj paket (pr. na `eth0`).
Kako bi se ovaj proces ubrzao, uvedena je brza međumemorija (*route cache*) u koju se pohranjuju tablice usmjeravanja, kako bi ih se moglo iznimno brzo dohvaćati. Ovo je iznimno važno pogotovo kada koristite Linux kao usmjerivač, jer neučinkovito dohvaćanje ruta znatno usporava cijeli mrežni podsustav. Upotrebom *route cache* međumemorije odnosno mehanizma, tablica usmjeravanja sa svojim unosima se praktično pohranjuje u posebnu tablicu u (RAM) memoriji koja se često općenito zove i ***forwarding information base*** (*FIB*) tablica, o kojoj se direktno brine sâm kernel.



*Osnovni dio **route cache** međumemorije je takozvani Protokol Independent Destination Cache, koji se naziva **DST**.*

Glavni zadatak ove *međumemorije* je pohranjivanje informacija koje omogućuju sustavu usmjeravanja (*routing subsystem*) pronalaženje odredišta za pakete kao i pružanje tih informacija kroz skup funkcija prema višim mrežnim slojevima.
Ova komponenta također nudi neke funkcije za upravljanje periodičkim čišćenjem navedene *međumemorije*.
Sâmi unosi su na najnižoj razini pohranjeni kao *hash* vrijednosti u *hash* tablici prema kojima se uspoređuje vrijednost za svaki paket koji se treba poslati na mrežu, prema odredištu. Naime svaki puta kada se neki mrežni paket mora poslati na mrežu, kernel prvo pretražuje *route cache* tablicu, a tek ako u njoj nije pronađen potreban unos, pretražuje se klasična tablica usmjeravanja, iz koje će se tada podaci i ponovno osvježiti u *route cache* tablicu. Ovakav način rada osigurava drastično bržu obradu i prosljeđivanje mrežnih paketa nego li je to bilo bez ovakve *hash* tablice.
Zapravo kernel ima nekoliko ovakvih tablica:

- Jedna je takozvana lokalna (***local***) o kojoj se brine sâm kernel i korisnici je nemaju potrebe dirati.
  U njoj se pohranjuju sve lokalne adrese, a pomoću nje kernel zna hoće li mrežni paket biti isporučen lokalno (na lokalnom računalu) ili, ako se mora slati na drugo računalo. Odnosno da će biti usmjeren dalje, ako je to dopušteno. To je prva stvarna tablica usmjeravanja koju koristi kernel kada obavlja pretraživanje. Ona se upotrebljava po redoslijedu, odmah nakon *route međumemorije*.
- Druga je takozvana glavna (***main***) tablica usmjeravanja koja se prema zadanim postavkama koristi za sve ostale adrese. Ovo je tablica usmjeravanja koju koristi naredba `route`, a to je i tablica koja se koristi kada ne navedete naziv tablice s naredbom `ip route`.

- Na kraju imamo i posebnu treću tablicu (o kojoj ćemo detaljnije govoriti malo kasnije), a to je takozvana ***custom*** tablica koja zapravo predstavlja sve ostale tablice (jer ih ovdje može biti više). Te će se tablice koristiti kada mrežni paket odgovara naprednim pravilima usmjeravanja navedenima s naredbom: `ip rule XY`. Te tablice usmjeravanja su poput normalnih tablica usmjeravanja, a mogu imati i zadanu standardnu krajnju rutu (*default route*).

Tablicu usmjeravanja imena ***local*** odnosno onu označenu kao lokalna, ne možemo mijenjati već samo gledati; s naredbom:

```
ip route show table local

local 192.168.100.1       dev eth0  proto kernel  scope host  src 192.168.100.1
broadcast 127.255.255.255 dev lo    proto kernel  scope link  src 127.0.0.1
broadcast 192.168.100.0   dev eth0  proto kernel  scope link  src 192.168.100.1
broadcast 192.168.100.255 dev eth0  proto kernel  scope link  src 192.168.100.1
broadcast 127.0.0.0       dev lo    proto kernel  scope link  src 127.0.0.1
local 127.0.0.1           dev lo    proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8         dev lo    proto kernel  scope host  src 127.0.0.1
```

U ovoj tablici vidimo sljedeće:

- Radi li se o *broadcast* (`broadcast`) adresi (pr. 192.168.100.255) ili adresi mreže (pr. 192.168.100.0) koja se isto označava kao (`broadcast`) ili o lokalnoj (`local`) adresi (pr. 192.168.100.1) ili o opsegu adresa koji se nalazi lokalno na našem računalu.
- Kroz dalje stupce je definirano preko kojih mrežnih sučelja (pr. `eth0`, `lo`, ...) se dolazi do odredišne IP adrese te kako se o tome brine kernel (`proto kernel`).

Tablicu usmjeravanja „***main***“ možemo vidjeti s naredbom:

```
ip route show table main
```

Takozvanu „***route cache***“ tablicu možemo vidjeti sa sljedećom naredbom:

```
ip route show cache
```

Međutim, od novijih Linux kernela (3.6+) *route cache* tablice odnosno ovakva implementacija ispisa (ali ne i rada) je napuštena zbog sigurnosnih rizika, pa ćemo pozivanjem naredbe: `ip route show cache` na novijim Linux kernelima dobiti prazan ispis. Ipak u bilo kojem trenutku možemo provjeriti je li određeni unos u ovoj međumemoriji ili nije.
Provjerimo nalazi li se unos za adresu 192.168.1.100 u ovoj memoriji; a ako se nalazi, bit će označen kao `cache`:

```
ip route get 192.168.1.100

192.168.1.100 dev eth0 src 172.17.100.1
    cache
```

U svakom slučaju, sve tablice odnosno njihove rezervirane vrijednosti vidljive su u posebnoj datoteci: `/etc/iproute2/rt_tables`.
Mi ovdje nećemo ulaziti u detalje već ćemo pregledati mogućnosti koje možemo optimizirati pomoću *sysctl* varijabli, koje vrijede i za novije i starije kernele Linuxa. Dakle pogledajmo *sysctl* varijable koje se odnose na usmjeravanje:

- `net.ipv4.route.max_size = X` - ovdje se postavlja vrijednost (**X**) kojom se definira maksimalan broj unosa u brzoj *ruting međumemoriji;* za sve unose u svim *route cache* tablicama. Ova vrijednost se rijetko prelazi jer je postavljena prilično visoko, negdje oko: 2.147.483.647. Svaki puta kada se tablice popune do ove razine, aktivirat će se proces čišćenja zvan „*Garbage Collector*“ (***GC***) pri kojemu će se početi čistiti zastarjeli unosi, prema parametrima definiranim u sljedećim varijablama.
- `net.ipv4.route.gc_elasticity = X` - ovdje definiramo prosječnu dubinu nîza u koji se spremaju rute, na vrijednost (**X**) kojom se definira i ponašanje ***GC*** procedure čišćenja *routing cache* međumemorije. Standardno je postavljena na dubinu osam. Kada se *routing cache* međumemorija prepuni odnosno kada su svi nîzovi popunjeni do zadnje dubine, pokreće se ***GC*** procedura čišćenja.
- `net.ipv4.route.max_size = X` - ovdje definiramo maksimalan moguć broj unosa u *routing cache međumemoriji.* Postavljena vrijednost je u većini slučajeva više nego dovoljna jer se automatski postavlja ovisno o količini dostupne RAM memorije u trenutku inicijalizacije sustava.
- `net.ipv4.route.gc_min_interval_ms = X` - ova opcija definira minimalno svakih koliko milisekundi će se pokretati ***GC*** i provjeravati stanje *route cache* međumemorije, bez obzira bila popunjena ili ne. Standardno je postavljeno na 500 (ms) što znači da se pokreće dva puta u sekundi.
- `net.ipv4.route.gc_interval = X` – ova opcija je prvo izbačena u kernelima nakon 2.6.32 pa je vraćena u kernelima 3.2+. Ona definira vrijeme u sekundama, svakih koliko sekundi će *se pokretati **GC*** zbog čišćenja *route cache* međumemorije. Standardno je postavljeno na 60 (sekundi) što znači kako se mehanizam provjere čišćenja pokreće svakih 60 sekundi. Dalje djelovanje mehanizma stvarnog čišćenja ovisi o sljedećim vrijednostima.
- `net.ipv4.route.gc_timeout = X` - ovdje se definira koliko dugo će se čuvati unosi u *route cache* međumemoriji. Standardno je postavljeno na 300. To znači kako će mehanizam čišćenja, (***GC***) prvo provjeravati je li neki unos stariji od 300 sekundi i ako je, tek tada će ga obrisati.
- `net.ipv4.route.gc_tresh = X` - standardno je postavljeno na **-1**.

Sve navedene varijable su dostupne i kao posebne datoteke unutar direktorija: `/proc/sys/net/ipv4/route/`.
Dodatno imamo i veličinu *hash* tablice koja se uopće može koristiti za sve unose u *routing cache* međumemorijama, a definira se na razini kernela, u trenutku pokretanja sustava. Ova varijabla se zove: `rhash_entries`. Njenu vrijednost postavlja sâm kernel na osnovu dostupne RAM memorije u trenutku pokretanja sustava, ali ju se može mijenjati modifikacijom *boot loadera.* Ona će biti aktivna tijekom sljedećeg restarta računala. Ovu vrijednost u većini slučajeva nećete mijenjati.

Maksimalna vrijednost koju mi možemo postaviti je definirana u *sysctl* varijabli `net.ipv4.route.max_size`, a koja ne može biti veća od one definirane u kernel ***boot*** opciji odnosno varijabli `dhash_entries`.
Ova *kernel boot* opcija se definira varijablom `dhash_entries` unutar konfiguracije *boot loadera* (**poglavlje: 11.1.1.5.**).
U slučaju ***RedHat/CentOS*** distribucije Linuxa govorimo o ***GRUB*** *boot loaderu* čije inicijalne postavke se nalazi u datoteci `/etc/default/grub`.
Ovdje nećemo objašnjavati detalja kako mijenjati ovu datoteku i *GRUB boot loader,* a dovoljno je znati kako se varijabla `dhash_entries` i njena nova vrijednost postavljaju unutar reda `GRUB_CMDLINE_LINUX="  "` i to postavljanjem ove varijable i njene vrijednosti unutar navodnika. Trenutno korištene *kernel boot* parametre odnosno *boot* parametre s kojima je *kernel* učitan možemo vidjeti sa sljedećom naredbom:

```
cat /proc/cmdline
```

Odnosno ljepše formatirane parametre možemo vidjeti sa sljedećim nîzom naredbi:

```
tr ' ' '\n' < /proc/cmdline
```

**Vratimo se na *ruting* međumemoriju i brisanje unosa iz *route cache* memorije.**
U nekim slučajevima kada mijenjamo rute, možemo se dovesti u situaciju kada se u određenom trenutku nova promjena još nije propagirala u *ruting* međumemoriju odnosno u *hash* tablicu na osnovu koje se zapravo odrađuje usmjeravanje (*routing*).
Tada je najčešće potrebno obrisati trenutnu *ruting* međumemoriju kako bi se sve zajedno osvježilo.
To možemo postići s naredbom (na starijim kernelima do inačice 3.x):

```
ip route flush cache
```


Na novijim kernelima (3.x +) više nije moguće vidjeti sadržaj ove međumemorije na način kao prije, osim s posebnim programima i alatima, sve iz sigurnosnih razloga.

## *Praćenje stanja route cache memorije*

Linux i dalje održava neke statistike o korištenju *ruting* međumemorije (*route cache*). Možete ih pronaći u datoteci `/proc/net/stat/rt_cache`. Međutim s naredbom `lnstat` ih možemo i lijepo ispisati. Pogledajmo i kako:

```
lnstat -s1 -i1 -c-1 -f rt_cache
```

```
rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|rt_cache|
 entries|  in_hit|in_slow_|in_slow_|in_no_ro|  in_brd|in_marti|in_marti| out_hit|out_slow|out_slow|gc_total|gc_ignor|gc_goal_|gc_dst_o|in_hlist|out_hlis|
        |        |     tot|      mc|     ute|        |  an_dst|  an_src|        |    _tot|     _mc|        |      ed|    miss| verflow| _search|t_search|
.     11|       0|       1|       1|       1|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|
.     11|       0|       1|       0|       1|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|
.     11|       0|       1|       0|       1|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|       0|
```

Objasnit ćemo neke od vidljivih statistika odnosno stupaca:

- `rt_cache_entries` je broj unosa u *ruting* međumemoriji. Trebali biste ga usporediti sa *sysctl* vrijednosti `net.ipv4.route.max_size` i osigurati da se ova rezervirana predmemorija nikad ne napuni kako bi se izbjeglo prekomjerno pokretanje procesa čišćenja memorije (***GC***).
- `rt_cache_in_hit` i `rt_cache_out_hit` su brojevi izbjegnutih regularnih pretraživanja iz klasične tablice usmjeravanja jer je rezultat pronađen u *ruting* međumemoriji (*pred memoriji*) za dolazne i odlazne pakete. Kada se radi o usmjerivaču (na kojem pratimo ove statistike), najčešće će se većina ovih događaja odnosno statistika događati sâmo na dolaznoj strani. Dodatno biste trebali usporediti ovu vrijednost s `rt_cache_in_slow_tot` i `rt_cache_in_out_slow_tot` koji predstavljaju broj pretraživanja u tablicama ruta. Na ovom sustavu, učinkovitost *route cache* memorije je više od 90% što je prilično dobro.
- `rt_cache_gc_total` označava koliko je često proces čišćenja memorije (***GC***) zatražio da se aktivira (naš uređaj je tek pokrenut pa nije bilo pokretanja *GC*-a). Dok `rt_cache_gc_ignored` odgovara tome koliko često **GC** nije pokrenut jer je već pokrenut prije toga, a ovisno o drugim vrijednostima. Razlika između ove dvije vrijednosti bi trebala biti vrlo mala kako bi se osiguralo da se **GC** proces ne pokreće nepotrebno veliki broj puta u sekundi.
- `rt_cache_gc_goal_miss` nam govori koliko često **GC** nije mogao odraditi čišćenje.
  Ova vrijednost treba težiti nuli (0).
- `rt_cache_gc_dst_overflow` govori koliko često je *ruting* međumemorija postala veća od dopuštene maksimalne veličine. Ovo se nikad ne smije dogoditi osim kada u radu pokušate smanjiti veličinu ove međumemorije.
- `rt_cache_in_hlist_search` i `rt_cache_out_hlist_search` označavaju koliko često je traženje u *ruting* međumemoriji trebalo pogledati sljedeći unos u nizu; svaki put kada Linux treba slijediti sljedeći pokazivač u predmemoriju, povećava se jedan od tih brojača. Ovo je pojam o prosječnoj duljini lanaca u *hash* tablici *cache* rute.

**Sumirajmo:**

- Ako se broj unosa u *route cache* međumemoriji popuni preko razine definirane u `net.ipv4.route.max_size` pokrenut će se proces čišćenja ove memorije (*garbage collector/**GC***).
- Svakih **n** sekundi definiranih u varijabli `net.ipv4.route.gc_min_interval_ms` pokretat će se mehanizam provjere (***GC***) treba li se ova međumemorija čistiti.
- Svi unosi koji su stariji od vrijednosti koja je definirana u varijabli `net.ipv4.route.gc_timeout` bit će obrisani od strane **GC**, kada se on aktivira (u gornjim slučajevima).

**Izvori informacija: (642),** **`man 5 proc`,`man ip route`,`man lnstat`.**

### 25.6.8.1. *Routing policy* (*Policy based routing*)

**Slijedi napredna cjelina!**
U slučajevima kada imamo potrebe za dodatnim rutama baziranim prema specifičnim potrebama korisnika ili sustava, i to je moguće odraditi unutar Linuxa. Jedan od primjera je potreba kada se mrežni paketi koji dolaze s točno određene IP adrese ili opsega IP adresa trebaju proslijediti na točno određenu rutu. Drugi primjer bi bio zabrana dolaska mrežnih paketa s određenih mreža i slično. Ovakva funkcionalnost se zove ***Routing policy*** ili ***Policy based routing*** odnosno usmjeravanje bazirano na pravilnicima. To se odrađuje tako da nam sustav daje mogućnost upotrebe više tablica usmjeravanja koje se pohranjuju unutar dijela ***custom*** tablica. Pri tome se prvo prolaze sve tablice unutar ***custom*** dijela tablica, a tek potom tablice unutar ***main*** i ***default*** tablice. Kako bi to radilo, koristi se interna mala baza podataka o usmjeravanju (***routing policy database*/RPDB**) koja kontrolira redoslijed kojim kernel pretražuje tablice usmjeravanja. Svako pravilo ima svoj prioritet, a pravila se ispituju sekvencijalno počevši od pravila **0** do pravila **32.767**. Kada novi paket stigne na obradu za usmjeravanje, uz pretpostavku da je *ruting* međumemorija prazna, kernel počinje s pravilom najvišeg prioriteta odnosno u ***RPDB*** pravilu **0**. Kernel prolazi svako pravilo u ovoj bâzi, sve dok paket koji se usmjerava ne odgovara pravilu. Kada se to dogodi kernel slijedi upute za slanje paketa definirane u tom pravilu. Obično to uzrokuje da kernel obavlja traženje rute u određenoj tablici usmjeravanja.
Ako se odgovarajuća ruta nalazi u tablici usmjeravanja kernel koristi tu rutu. Ako se takva ruta ne pronađe, kernel se vraća na početak kako bi ponovno prešao ***RPDB*** nazu sve dok se svaka opcija ne iscrpi. Međutim, ako želite koristiti ovu funkcionalnost provjerite je li vaš kernel kompiliran sa sljedeće dvije opcije: `IP_ADVANCED_ROUTER` i `IP_MULTIPLE_TABLES`.
Pogledajmo je li naš kernel kompiliran s njima:

```
egrep "IP_ADVANCED_ROUTER|IP_MULTIPLE_TABLES" /boot/config-`uname -r`

CONFIG_IP_ADVANCED_ROUTER=y
CONFIG_IP_MULTIPLE_TABLES=y
```

Vidimo kako imamo potvrdu odnosno `=y` što znači kako ova podrška postoji u našem kernelu.

Pogledajmo i osnovne naredbe koje možemo definirati unutar pravila (*rules*) usmjeravanja. To su (postoji ih i više ali ih nećemo sve navoditi):

- `unicast` - *unicast* naredba odnosno pravilo je najčešća vrsta pravila. Ova jednostavna vrsta pravila uzrokuje da se kernel odnosi na određenu tablicu usmjeravanja u potrazi za rutom. Ako nijedna vrsta pravila nije navedena u naredbenom retku, pretpostavlja se da je pravilo upravo *unicast*. Pogledajmo nekoliko primjera:

```
ip rule add unicast from 192.168.100.100 table 5
ip rule add unicast iif eth2 table 5
ip rule add unicast fwmark 4 table 4
```

- `broadcast` - ovo je *broadcast* pravilo slične upotrebe kao i *unicast*, ali za *broadcast* IP adrese.
- `multicast` - ovo je *multicast* pravilo slične upotrebe kao i *unicast*, ali za *multidcast* IP adrese.
- `nat` - ovo je vrsta naredbe koja se koristi za ispravan rad *stateless **NAT***-a. Pogledajmo još nekoliko primjera:

```
ip rule add nat 192.168.100.200 from 10.17.247.11
ip rule add nat 10.20.11.0 from 172.60.20.0/16
```

- `unreachable` - svako pretraživanje rute koja treba odgovarati unosu pravila u kojem trebamo dobiti poruku o nedostupnosti koja treba uzrokovati da kernel generira ***ICMP*** poruku nedostupan (*ICMP unreachable*) prema izvornoj adresi paketa (pošiljatelju). Pogledajmo i ovdje nekoliko primjera:

```
ip rule add unreachable iif eth2 tos 0xc0
ip rule add unreachable iif wan0 fwmark 5
ip rule add unreachable from 192.168.200.0/24
```

- `prohibit` - svako pretraživanje rute koja treba odgovarati unosu pravila u kojemu trebamo dobiti poruku o nedostupnosti koja treba uzrokovati da kernel generira ***ICMP*** poruku zabranjen pristup (*ICMP prohibited*) prema izvornoj adresi paketa (pošiljatelju).

Pogledajmo nekoliko primjera za `prohibit`:

```
ip rule add prohibit from 192.168.200.100
ip rule add prohibit to 192.168.150.0/24
ip rule add prohibit fwmark 7
```

- `blackhole` - dok paket prolazi ***RPDB*** bazu i sve njene tablice usmjeravanja, bilo koji način traženja koji odgovara pravilu s ovom vrstom, uzrokuje da se paket odbaci. Pri tome nijedna vrsta ***ICMP*** poruke neće biti poslana i neće se proslijediti nijedan paket. Pogledajmo i ove primjere:

```
ip rule add blackhole from 192.168.200.100
ip rule add blackhole from 192.168.200.0/24
ip rule add blackhole to 10.17.247.11/28
```

**Kreiranje vlastite tablice usmjeravanja**
Zamislimo Linux usmjerivač odnosno računalo s dvije mrežne kartice:

- `eth0` s IP adresom: 192.168.1.1 na koju imamo spojeni *default gateway (192.168.1.254)* odnosno primarni podrazumijevani izlazni usmjerivač prema internetu, spojen na *optiku prema telekomu (ISP-u)*.
- `eth1` s IP adresom: 192.168.2.1 na koju imamo spojen sekundarni izlazni usmjerivač (*192.168.2.254*) prema internetu (spojen na ***ADSL*** prema telekomu (*ISP-u*)).

Trenutno sva računala iz svih mreža prolaze kroz Linux i prema internetu idu preko njegove `eth0` mrežne kartice, preko *default gatewaya: 192.168.1.254.* To želimo promijeniti na način da sva računala iz mreže na kojoj su nam poslužitelji, a nalaze se na adresama 192.168.2.0/24 i spojena su na našu mrežnu karticu `eth1` ne koriste standardni *default gateway: 192.168.1.254* koji je spojen s optikom, već drugi, koji je spojen preko ***ADSL*** linije i čija IP adresa je *192.168.2.254.*

Stoga ćemo kreirati svoju zasebnu tablicu usmjeravanja imena `serveri-1` koja će imati identifikator **200**.

```
echo 200 serveri-1 >> /etc/iproute2/rt_tables
```

I sada kada imamo svoju novu tablicu usmjeravanja (*ruting* tablicu) dodajemo joj svoje pravilo usmjeravanja (*rule*) za sve koji dolaze s mreže 192.168.2.0/24. Mi želimo da svi koji dolaze s mreže 192.168.2.0/24 upadaju u našu tablicu usmjeravanja imena: `serveri-1`. Pogledajmo kako to izgleda:

```
ip rule add from 192.168.2.0/24 table serveri-1
```

Dakle rad s pravilima usmjeravanja (Engl. *Routing rules*) definira se prema sljedećoj sintaksi:
`ip rule` **`NAREDBA SELEKTOR AKCIJA TABLICA`**

Pri tome:

- **NAREDBA** može biti:
  - `list` - za ispis tablica usmjeravanja.
  - `add` - za dodavanje pravila usmjeravanja.
  - `del` - za brisanje pravila usmjeravanja.
  - `flush` - za brisanje apsolutno **<u>svih</u>** pravila (oprez).
- **SELEKTOR** može biti:
  - `not` - definira negaciju izraza.
  - `from` - definira da se prati s koje mreže dolazi mrežni promet.
  - `to` - definira da se prati na koju mrežu dolazi mrežni promet.
  - `tos` - "*type of service*" odnosno vrsta usluge za ***QoS***: pr. `0x00`=normalno,`0x02`=minimize cost,`0x04`=maximize reliability, ...
  - `fwmark` - ovime definiramo treba li se provjeravati oznaka na mrežnim paketima koja je administrativno dodana od strane sustava za filtriranje paketa (`iptables`) i to sâmo, ako smo to tako napravili. Oznaka `fwmark` nije dio normalnog IP paketa i postoji samo kao dio strukture podataka koja se nalazi u memoriji na uređaju za usmjeravanje, kako bi se označio određeni paket prema željenom kriteriju. Ako pak nema oznake (*fwmark*) na paketu, ovaj ključ za pretraživanje se ne upotrebljava. Kada je on ipak prisutan, kernel će tražiti podudaranje, a ako se ne pronađe odgovarajući unos, kernel će nastaviti traženje rute prolazom kroz ***RPDB*** bazu.
  - `dev` - ovdje definiramo mrežno sučelje na koje mrežni paket dolazi; u novijim inačicama ove naredbe imamo malo drugačiji odabir:
    - `iif` - označava ulazno (*i=input*) mrežno sučelje.
    - `oif` - označava izlazno (*o=output*) mrežno sučelje.
  - `prio` - označava prioritet pravila (*rule-a).*

- **AKCIJA** može (i ne mora) biti:
  - `table` - tablica
    - **TABLICA** može biti:
      - `local` - označava tablicu ***local***.
      - `main` - označava tablicu ***main***.
      - `default` - označava tablicu ***default***.
      - `XY` - broj ili ime naše tablice.

Sada pogledajmo kako izgledaju naše tablice usmjeravanja s pravilima usmjeravanja:

```
ip rule ls

0:      from all lookup local
32765:  from 192.168.2.0/24 lookup serveri-1
32766:  from all lookup main
32767:  from all lookup default
```

A sada možemo dodavati i nove rute odnosno pravila (*rule*) unutar naše tablice usmjeravanja imena `serveri-1`.
Vidimo kako naše novo dodano pravilo ima identifikator (prioritet) broj `32765`. Dodatno vidimo i kako se ova ruta prema identifikatoru (`32765`) izvršava prije ***main*** i ***default*** tablica usmjeravanja.

Sada ćemo dodati i novo pravilo unutar tablice usmjeravanja `serveri-1`, a u kojem definiramo da je podrazumijevani usmjerivač (*default gateway)* za mrežu 192.168.2.0/24, uređaj s IP adresom: 192.168.2.254 jer imamo drugu mrežnu karticu `eth1` u toj mreži, koja je spojena na njega. Dodajmo ovo novo pravilo:

```
ip route add default via 192.168.2.254 table serveri-1
```

Pogledajmo sada našu novu tablicu usmjeravanja naziva `serveri-1`

```
ip route list table serveri-1

default via 192.168.2.254 dev eth1
```

Vidimo kako je u našoj novoj tablici `serveri-1` *default gateway* postavljen na `192.168.2.254` kako smo i htjeli. To znači kako će svi koji dolaze s mreže 192.168.2.0/24 koristiti *default gateway* 192.168.2.254 do kojeg dolazimo preko naše druge mrežne kartice `eth1`, a ne na naš standardni podrazumijevani usmjerivač (*default gateway):* 192.168.1.254 do kojeg smo do sada dolazili s mrežne kartice `eth0` odnosno s mreže 192.168.1.0/24.
Za sve one koji dolaze s mreže 192.168.1.0/24 i dalje će se koristiti podrazumijevani usmjerivač*: 192.168.1.254* što dokazuje naša standardna tablica usmjeravanja koja nam daje naš standardni podrazumijevani usmjerivač i druge rute.

Pogledajmo našu standardnu tablicu usmjeravanja:

```
ip route list table main

192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.1
default via 192.168.1.254 dev eth0
```

Dodajmo drugu rutu u našu novu tablicu usmjeravanja: `serveri-1` u kojoj definiramo da će svi koji žele doći do mreže: 192.168.100.0/24 biti proslijeđeni na usmjerivač s IP adresom: 192.168.1.118:

```
ip route add 192.168.100.0/24 via 192.168.1.118 table serveri-1
```

Pogledajmo ponovno našu tablicu usmjeravanja:`serveri-1`

```
ip route list table serveri-1

default via 192.168.2.254 dev eth1
192.168.100.0/24 via 192.168.1.118 dev eth0
```

Sada vidimo i podrazumijevanu (*default)* rutu odnosno podrazumijevani usmjerivač (`192.168.2.254`) te novi unos koji smo upravo dodali (`192.168.100.0/24 via 192.168.1.118 dev eth0`).

**Izvori informacija: `man ip rule`, `man ip route`.**

## Označavanje važnih paketa pomoću naredbe `iptables`

Zamislimo slučaj u kojem želimo označiti mrežne pakete prema određenom kriteriju te tako označene pakete koristiti za primjenu određenog pravila usmjeravanja. U našem primjeru želimo sve mrežne pakete kojima je odredišni ***TCP*** port ***25*** (dakle prema *e-mail* poslužitelju) označiti s oznakom (*fwmark/mark*) broj **10**, na sljedeći način:

```
iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 25  -j MARK --set-mark 10
```

I sada možemo ove pakete koji nose oznaku (*fwmark*) **10** prosljeđivati samo na našu novu tablicu usmjeravanja: `serveri-1`

```
ip rule add fwmark 10 table serveri-1
```

Pogledajmo što smo sada dobili. Dakle samo ćemo ispisati dio koji nas ovdje zanima:

```
ip rule list

32762:  from all fwmark 0xa lookup serveri-1
```

To znači kako sa bilo koje IP adrese (`from all`) označeni paketi koji nose oznaku **10** (`fwmark 0xa`) moraju biti proslijeđeni na tablicu usmjeravanja imena: `serveri-1` što je povezano pomoću poveznice: `lookup serveri-1`.

**Izvori informacija: `man ip rule`, `man iptables`.**

### Brisanje (ruting) pravila usmjeravanje unutar specifične tablice usmjeravanja.

Pravila unutar naše tablice brišemo tako da ga moramo identificirati s identifikatorom koji predstavlja njen prioritet.

Tako bi da imamo novo pravilo s prioritetom 32.764, to pravilo obrisali sa sljedećom naredbom:

```
ip rule del pri 32764
```

Isto tako možemo dodavati nova pravila s definiranim prioritetom, pa na taj način dodajemo određeno pravilo ispred ili iza nekog već postojećeg. Mi ćemo dodati još jedno pravilo ispred našega; čisto za probu:

```
ip rule add pri 32764 from 192.168.3.0/24 table serveri-1
```

Pogledajmo sada naša pravila usmjeravanja:

```
ip rule ls
0:      from all lookup local
32764:  from 192.168.3.0/24 lookup serveri-1
32765:  from 192.168.2.0/24 lookup serveri-1
32766:  from all lookup main
32767:  from all lookup default
```

Vidimo novo pravilo (`PRIO 32764`) uredno dodano ispred našega već postojećeg pravila (*PRIO* `32765`).

Važno je razumjeti kako sve ove promijene nisu trajne, a da bi bile trajne možemo ih dodati u datoteku specifičnu za mrežno sučelje na koje se odnose. Pa bi tako u primjeru dodavanja novog pravila poput:

```
ip rule add from 192.168.2.0/24 table serveri-1
```

To isto pravilo morali dodatni u posebnu datoteku mrežnog sučelja; primjerice za mrežno sučelje `eth0` iz koje se čitaju *policy based rute*, bi bila datoteka imena: `/etc/sysconfig/network-scripts/rule-eth0`.

Navedena datoteka bi tada sadržavala sljedeći redak odnosno unos:

```
from 192.168.2.0/24 table serveri-1
```

Na ***RedHat/CentOS 8.x***, pošto se većina skripti unutar direktorija: `/etc/sysconfig/network-scripts/` više ne koriste, za rad s pravilima usmjeravanja (koristi se tzv. novi *Network Manger*), potrebno je koristiti novi program: `nmcli`.
Način njegove konfiguracije i rada sada nećemo posebno objašnjavati.



Za više detalja o ***Network Manager*** servisu i `nmcli` naredbi, pogledajte poglavlje:
**25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**

Za ***RedHat/CentOS 8.x***:
Ako se ipak želite vratiti na stari način rada morate instalirati paket koji se u ovoj inačici više ne koristi:

```
yum install network-scripts
```

## Veza *PBR*-a i *VRF* (engl. *Virtual routing and forwarding*)



Za dodatne primjere upotrebe i proširenja ***PBR*** sustava, pogledajte i poglavlje:
**25.7.9. Naredba ip** i to cjelinu u kojoj govorimo o ***VRF***-ovima, to jest naredbu **`ip vrf`**.

Naime ***VRF*** (engl. *Virtual routing and forwarding*) tehnologija iako se najviše koristi na naprednim usmjerivačima, koristi se i za izolaciju mreža preko Linuxa kao *hipervizora* za virtualna računala, u posebnim primjenama kod većih sustava za virtualizaciju (pr. *Cloud sustavi)* za izolaciju to jest upotrebu virtualnih računala unutar virtualnih mreža u oblaku (engl. *virtual cloud network [***VCN***])* to jest privatnih mreža u oblaku (engl. *private cloud network [***PCN***])*. VRF osigurava izolaciju prometa na OSI sloju tri za usmjeravanje, slično kako se VLAN koristi za izolaciju prometa na OSI sloju dva. U osnovi, VRF čine zasebne (vršne) tablice usmjeravanja.
Mrežni uređaji (fizički, virtualni ili logički) povezani su s VRF-om tako što se uređaj spaja na određeni VRF. U tom trenutku mrežne adrese dodijeljene uređaju su lokalne za VRF s uređajem i povezanim tablicama usmjeravanja premještenim u tablicu usmjeravanje povezanu s konkretnim VRF-om. To znači da je unutar svakog VRF-a moguće koristiti izolirani IP adresni prostor, kreirati podmreže i tablice usmjeravanja te koristiti vatrozid(e).

**Izvor informacija:(536),(537),(538),(539),(540),(541),(542),(543),(544),(545),(546),(547),(548),(549),(550),(551),(552), (553),(554),(555),(910),`man ip`,`man ip rule`,`man nmcli`,`man nm-settings`,`man ip vrf`.**

### 25.6.9. Skripte i funkcijske datoteke sustava zadužene za konfiguraciju mreže

**Slijedi napredno poglavlje!**

***Red Hat/CentOS* 6.x i 7.x** (i sâmo djelomično ***RedHat/CentOS 8.x***) te kompatibilne distribucije Linuxa koriste nekoliko datoteka koje sadrže važne zajedničke funkcije koje se koriste za podizanje i spuštanje mrežnih sučelja. Drugim riječima ove skripte se koriste tijekom svakog gašenja ili uključivanja mrežnih sučelja, ali i prilikom restarta mrežnog servisa odnosno *daemona.* Naime umjesto da se unutar svake mrežne skripte (datoteke) nalaze sve ove funkcije, one se grupiraju u nekoliko datoteka koje se pozivaju kada je to potrebno. Jedna od ovih skripti s funkcijama je i datoteka: `/etc/sysconfig/network-scripts/network-functions`. Ona sadrži najčešće korištene IPv4 funkcije koje su korisne za mnoge druge mrežne skripte koje konfiguriraju mrežna sučelja odnosno mrežne kartice.
Te funkcije uključuju kontaktiranje pokrenutih programa koji su zatražili informacije o promjenama statusa sučelja te:

- Informacije o mrežnom sučelju kao i postavljanje imena računala (*hostname*).
- Pronalaženje uređaja za pristup odnosno *default gateway-a,* definiranog u *datoteci* `/etc/sysconfig/network`.
- Provjeru je li određeni uređaj isključen ili nije, kao i provjeru koristi li se *bonding* (*agregacija*).
- Provjeru je li se mijenjala konfiguracija DNS poslužitelja u datoteci: `/etc/resolv.conf`.
- Dodavanje ruta, ali i logiranje poruka vezanih za mrežu, te druge značajke.

Budući da se funkcije potrebne za IPv6 sučelja razlikuju od IPv4 sučelja, za tu potrebu imamo funkcijsku datoteku: `/etc/sysconfig/network-scripts/network-functions-ipv6`.
Postoje i dvije kontrolne skripte koje su zadužene za aktiviranje i deaktiviranje mrežnih sučelja. Radi se o skriptama:

- Za aktiviranje (pokretanje) mrežnih sučelja: `/etc/sysconfig/network-scripts/ifup`, a ona je simbolička poveznica (*link*) na: `/usr/sbin/ifup`.
- Za deaktiviranje (gašenje) mrežnih sučelja: `/etc/sysconfig/network-scripts/ifdown`, a ona je simbolička poveznica na: `/usr/sbin/ifdown`.

Ove skripte pozivaju lokalne funkcije ili pozivaju skripte specifične za vrste ili kategorije mrežnih sučelja poput *VLAN* (**802.1Q**) sučelja, a one (skripte) se nalaze u direktoriju `/etc/sysconfig/network-scripts/`.
Druge skripte u navedenom direktoriju (mapi) su namijenjene za izvođenje različitih zadataka inicijalizacije mreže tijekom procesa podizanja mrežnih sučelja, za koje koriste i dodatne funkcijske skripte (skripte s funkcijama) poput: `/etc/rc.d/init.d/functions` i već navedene `/etc/sysconfig/network-scripts/network-functions`.
Naime na početku izvršavanja jedne od ove dvije skripte (kod aktivacije ili deaktivacije mrežnog sučelja) nakon svih provjera, pokreću se konačne skripte zadužene za konkretnu vrstu mrežnog sučelja, pa tako imamo sljedeće skripte prema vrsti ili kategoriji mrežnih sučelja:

- Za njihovu **aktivaciju (**sve do ***RedHat/CentOS 8.x.*** koji sada isključivo koristi ***Network Manager*** servis**):**
  - `/etc/sysconfig/network-scripts/ifup-eth` - za ***ethernet*** vrstu mrežnih sučelja; poput klasičnih: ***ethernet***, ***bonding***, ***bridge, VLAN, sub interface*** i sličnih mrežnih sučelja.
    Pri tome primjerice skripta/datoteka (`ifup-eth`) provjerava:
    - Učitane kernel module (ako su potrebni).
    - Provjerava sve potrebno za ***bridge*** mrežna sučelja te sve potrebno za ***tap*** mrežna sučelja.
    - Provjerava sve potrebno za ***bond*** mrežna sučelja.
    - Provjerava sve potrebno za ***wireless*** mrežna sučelja (koja onda poziva: `ifup-wireless`).
    - Provjerava sve potrebno za klasična ***ethernet*** (*eth, eno, enp…*) mrežna sučelja.
    - Provjerava stanje rada mrežnih sučelja i na kraju ih konfigurira, postavljenjem pomoću konfiguracijskih datoteka specifičnih za mrežna sučelja:
      - Konfiguracijske datoteke ovisne o mrežnim sučeljima su datoteke imena: `/etc/sysconfig/network-scripts/ifcfg-*` (**koriste se i na *RedHat/CentOS* 8.x**)
        Pri tome je `*=ethXY,enoXY,bondXY,….` U ovim datotekama su definirani IP i drugi parametri rada svakog pojedino definiranog mrežnog sučelja.
  - `/etc/sysconfig/network-scripts/ifup-ppp` - za **`PPP`** (***point to point***) vrstu mrežnih sučelja; čije konfiguracijske datoteke su imena: `/etc/sysconfig/network-scripts/ifcfg-ppp*` pri čemu je `*=0,1,2,3,4,` …
- Za njihovu **deaktivaciju** imamo **(**sve do ***RedHat/CentOS 8.x.*** koji sada isključivo koristi ***Network Manager*** servis**):**
  - `/etc/sysconfig/network-scripts/ifdown-eth` - za ***ethernet*** vrstu mrežnih sučelja; klasičnih ***ethernet***, ***bonding***, ***bridge, VLAN, sub interface*** i sličnih mrežnih sučelja.
    Pri tome skripta/datoteka (`ifdown-eth`) također provjerava sve isto kao i `ifup-eth` skripta.
    - Te ih na kraju i *dekonfigurira* (briše konfiguraciju) pomoću konfiguracijskih datoteka specifičnih za mrežna sučelja:
    - Konfiguracijske datoteke ovisne o mrežnim sučeljima su datoteke imena: `/etc/sysconfig/network-scripts/ifcfg-*` pri tome je `*=ethXY,enoXY,bondXY,….` U ovim datotekama su definirani IP i drugi parametri rada svakog pojedino definiranog mrežnog sučelja (**ove datoteke se koriste i za RedHat/CentOS 8.x**).
  - `/etc/sysconfig/network-scripts/ifdown-ppp` – za **`PPP`** (***point to point***) vrstu mrežnih sučelja; čije konfiguracijske datoteke su imena poput:
    `/etc/sysconfig/network-scripts/ifcfg-ppp*` pri tome je `*=0,1,2,3,4,` …

Pred kraj imamo i skripte koje se pokreću kada su se već sva mreža sučelja inicirala, te je potrebno odraditi dodatne radnje. I ovdje imamo dvije skripte: jednu za uključivanje (inicijalizaciju) i drugu za gašenje (brisanje). Navedene skripte su:

- `/etc/sysconfig/network-scripts/ifup-post` - za aktivaciju dodatnih opcija i parametara.
- `/etc/sysconfig/network-scripts/ifdown-post` - za deaktivaciju dodatnih opcija i parametara.

Opcije i parametri koji se ovim skriptama ili aktiviraju ili deaktiviraju su:

- Za pozivanje skripti za rad s rutama, odnosno učitavaju se sljedeće datoteke:
  - `/etc/sysconfig/network-scripts/ifup-routes` - za dodavanje ruta.
  - `/etc/sysconfig/network-scripts/ifdown-routes` - za brisanje ruta.
- Za osvježavanje unosa DNS poslužitelja i DNS opcija prema potrebi i za sve druge stvari.

I na sâmom kraju imamo i skripte koje su zadužene za same ***rute*** koje smo maloprije spomenuli, a koje se nalaze unutar direktorija `/etc/sysconfig/network-scripts/` i to njih dvije, od kojih je jedna:

- Za njihovu **aktivaciju** odnosno dodavanje ruta (**sve do *RedHat/CentOS 8.x.* koji sada isključivo koristi *Network Manager* servis**):
  - `/etc/sysconfig/network-scripts/ifup-routes` - koja čita datoteke koje sadrže rute koje se primjenjuju za specifična mrežna sučelja a pri tome imamo:
    - Rute za pojedina mrežna sučelja: `/etc/sysconfig/network-scripts/route-`**`X`** pri čemu je **`X`**`=eth*,eno*,bond*,br*,…` (**ove datoteke se koriste i za RedHat/CentOS 8.x**)
    - Rute za *policy (based) routing* koji se definira isto za svako mrežno sučelje, a nalaze se u datoteci: `/etc/sysconfig/network-scripts/rule-`**`X`** pri čemu je **`X`**`=eth*,eno*,bond*,br*,…` (**ove datoteke se koriste i za RedHat/CentOS 8.x**)
- Za njihovu **deaktivaciju** odnosno brisanje (**sve do *RedHat/CentOS 8.x.* koji sada isključivo koristi *Network Manager* servis**):
  - `/etc/sysconfig/network-scripts/ifdown-routes` - koja čita datoteke za rute za specifična mrežna sučelja isto kao i za njihovu aktivaciju.

Zapravo na sâmom početku odnosno u trenutku pokretanja sustava, sustav kada inicijalizira mrežu, prvo pokreće skriptu: `/etc/init.d/network` s parametrom `start`. Isto tako ova skripta se poziva kada pokrećemo naredbu `service network start/stop/restart` ili `systemctl` *`start/stop/restart`* `network`.

Ova skripta poziva sve potrebne funkcijske datoteka i skripte koje smo do sada spomenuli, sve ovisno o tome koja mrežna sučelja konfigurira ili briše odnosno poziva li se opcijom `start`, `stop` ili `restart`. Naime ona prvo:

- Traži sva dostupna mrežna sučelja definirana u direktoriju: `/etc/sysconfig/network-scripts/`.
- Potom ih inicijalizira ili deaktivira (briše), pomoću skripti specifičnih za svako od njih.
- Zatim dodaje statičke rute i *policy rute* (ako ih ima).

Dakle sve ovisno o potrebi, odnosno ovisno kako se poziva; sa: `start`, `stop` ili `restart`.

Definicija opcija konfiguracijskih datoteka ovisna je o inačici *init* skripti a nalazi se u datoteci: `/usr/share/doc/initscripts-`***`inačica`***`/sysconfig.txt` pri čemu ***`inačica`*** označava inačicu *initscripts* paketa, čiju inačicu možemo vidjeti sa sljedećim nîzom naredbi:

```
rpm -qa | grep initscripts | awk -F- '{print $1 "-" $2}'
```

Ako želimo čitati navedenu datoteku `sysconfig.txt` s detaljnim uputama to možemo postići na sljedeći način (nîz naredbi):

```
less /usr/share/doc/`rpm -qa|grep initscripts|awk -F- '{print $1 "-"
$2}'`/sysconfig.txt
```



Slijede **Red Hat/ 8.x specifičnosti i važne novosti odnosno promijene!**

Od ***RedHat/CentOS*** v.8.x napuštena je upotreba većine skripti unutar vršnog direktorija: `/etc/sysconfig/network-scripts/`. Jedino što se sada ovdje nalazi je konfiguracijska datoteka za naše(a) mrežno sučelje, primjerice za `eth0` bi to bila stara konfiguracijska datoteka imena: `/etc/sysconfig/network-scripts/ifcfg-eth0` i još par skripti za usmjeravanje. Dakle sve gore navedene funkcionalnosti sada se odrađuju upotrebom takozvanog ***Network Manager*** servisa.

Njegov status možemo vidjeti s naredbom:

```
systemctl status NetworkManager
```

Odnosno možemo ga restartati, zaustavljati ili pokretati uobičajeno s metodama: `start/stop/restart`.

Vezano za mrežna sučelja, ako to trebamo, možemo ih restartati tako da ih prvo spustimo pa podignemo (odjednom):

```
ifdown eth0;ifup eth0
```

Ove dvije skripte (`ifdown` i `ifup`) su poveznice na *Network Manager* skripte: `nm-ifdown` i `nm-ifup`.

Novija metoda restarta mrežnih sučelja radi se upotrebom naredbe `nmcli`, koja će u slučaju kada smo promijenili konfiguraciju mreže (pr. IP adresu i slično) učitati novu konfiguraciju i primijeniti ju.

To možemo postići s dvije naredbe u nîzu; odnosno pomoću naredbe `nmcli` na sljedeći način:

```
nmcli networking off; nmcli networking on
```

Pri tome servis ***Network Manager*** mora biti aktivan jer zapravo on odrađuje sve ove zadatke.

---

Međutim, ako se ipak želite vratiti na stari način rada, morate instalirati paket koji se u ovoj inačici (**v.8.x**) više ne koristi:

```
yum install network-scripts
```

**Izvori informacija: (612),(613),(614),** **`man nmcli`**, **`man ifdown`**, **`man ifup`**, **`man nm-settings`**.

---

# 25.7. Osnovni mrežni alati

U ovoj cjelini ćemo objasnit neke od osnovnih alata odnosno programa za provjeru dostupnosti i ispravnosti rada mreže i mrežnih servisa.

## 25.7.1. *Naredba ping*

Naredba `ping` se koristi za testiranje dostupnosti računala (poslužitelja, mrežnog uređaja, …) na drugoj strani mreže, a koji koristi TCP/IP protokol. Osim sâme dostupnosti mjeri se i vrijeme koje je potrebno da mrežni paket dođe do odredišta.
Ime naredbe `ping` je nastalo na osnovu podvodnog *sonara* koji šalje impulse zvuka u određenom smjeru, te detektira jeku (engl. *Echo*) odnosno odbijeni zvuk, koji se odbio od određenog predmeta ispod vode. Zvuk koji se pri tome čuje podsjeća na riječ "*ping*". Naredba `ping` šalje poseban paket, korištenjem **ICMP** protokola (*Internet Control Message Protocol*) konkretno se radi o "*ICMP Echo Request*" poruci poslanoj na odredišnu IP adresu (onoga čiju dostupnost testiramo) te se čeka na odgovor. S odredišne IP adrese se šalje odgovor (*ICMP Echo Reply*) te se mjeri prosječno vrijeme između slanja i primanja (engl. *Round-trip*). Dakle ovo vrijeme je vrijeme koje je trebalo da paket dođe do odredišta uz vrijeme potrebno da dobijemo odgovor, podijeljeno s dva.
Tako se dobio prosjek odlaznog i dolaznog vremena, što nije savršeno jer u praksi put kojim je paket putovao do odredišta, ne mora biti isti kao u povratu nazad, pa stoga odlazno i dolazno vrijeme mogu biti različiti[960]. Nadalje, prati se i koliko se paketa potencijalno izgubilo (jer ih se obično šalje više). Vrijeme između slanja i primanja se zove latencija ili tromost odnosno kašnjenje. Konačan rezultat je ispis statistike o primljenim paketima, i to: koliko je paketa poslano, a koliko primljeno (koliki je postotak uspješnosti).
Osim toga vidimo i koja su prosječna vremena od slanja do primanja paketa: minimalno, srednje i maksimalno vrijeme. *Ping* omogućava i korištenje dodatnih opcija (prekidača) od kojih ćemo koristiti sâmo prekidač `-c` koji označava koliko paketa želimo poslati. Pogledajmo i primjere. Provjerimo dostupnost IP adrese 192.168.200.100 na našoj lokalnoj mreži, slanjem tri paketa upotrebom naredbe `ping` na sljedeći način:

```
ping -c 3 192.168.200.100

PING 192.168.200.100 (192.168.200.100) 56(84) bytes of data.
64 bytes from 192.168.200.100: icmp_req=1 ttl=64 time=0.297 ms
64 bytes from 192.168.200.100: icmp_req=2 ttl=64 time=0.125 ms
64 bytes from 192.168.200.100: icmp_req=3 ttl=64 time=0.199 ms
--- 192.168.200.100 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3997ms
rtt min/avg/max/mdev = 0.125/0.238/0.325/0.072 ms
```

Ovdje vidimo da je druga strana dostupna i da je bilo `0%` gubitaka. Važno je i da nije bilo većih odstupanja u vremenima za svaki paket te vidimo da je prosječno vrijeme (*avg* : engl. *Average*): 0,238 ms što je za lokalnu mrežu solidno prolazno vrijeme.

Moguće je tijekom ispisa zapisivati i vrijeme kada su ICMP paketi poslani, za svaki zaseban paket, ako naredbi `ping` dodamo prekidač `-D`, a pri tome se vremenska oznaka zapisuje u UNIX vremenu (pr. `1580474459.1178=31.1.2020,12:40:59.117`).
Još jedan koristan prekidač je `-A` koji prvo mjeri vrijeme prolaska paketa do odredišta (*Round-trip-time*), a potom šalje pakete s korištenjem tog vremena kao vremena razmaka između slanja novih paketa prema odredištu. S time se paketi šalju brže nego obično (jer je standardno vremenski odmak između slanja paketa 1s). Još jedna od korisnih opcija je `-W` nakon koje navodimo vrijeme u sekundama potrebno za odgovor (standardno se čeka ***2 x RTT*** vrijeme), ako očekujemo duže vrijeme za odgovor.
Još jedan od korisnih prekidača je i `-t` za testiranje promjene TTL vremena života paketa [*u IP zaglavlju*]).
Dodatni korisni prekidač je `-b` nakon kojega navodimo *broadcast* adresu mreže, ako želimo *pingati* cijelu mrežu, što je korisno u slučajevima potrebe za otkrivanjem računala u lokalnom segmentu mreže. I to ako nemaju postavljenu ***sysctl*** varijablu
`net.ipv4.icmp_echo_ignore_broadcasts = 1`.

Za testiranje mogućnosti procesiranja velike količine paketa imamo i opciju `-f` s kojom šaljemo maksimalan broj *ICMP* paketa koliko naše računalo može kreirati u datom trenutku, što potencijalno može zagušiti računalo (ili mrežu), ali je dobar test.



*Koja vremena odziva očekivati za **LAN** mreže: sve unutar nekoliko milisekundi (ms). Dok za **WAN** mreže unutar države to vrijeme iznosi do oko ~100 ms odnosno do ~50ms, ako imate zakupljene veze (između dva grada unutar tvrtke ili sl.). Vremena do drugih susjednih država i dalje od toga iznose prosječno oko 200 ms, a sve puno dalje je nešto ili malo više od 200 ms.*



Za druge napredne primjere upotrebe naredbe **`ping`**, pogledajte i poglavlja:
**23.2. IP fragmentacija.**
**23.4.1. Time to live (*TTL*).**
**23.4.2.1. MTU primjeri.**

Jedna od navedenih **ICMP** poruka odnosno paket koji šalje naredba `ping` izgleda ovako (prema vrsti ICMP poruke ovo je *ICMP* upit):

```
(1)Frame 48: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
(2)Ethernet II, Src: 40:8d:5c:57:26:63, Dst: d4:76:ea:2e:70:7f
(3)Data Protocol Version 4, Src: 192.168.1.103, Dst: 192.168.200.100
Data Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d5a [correct]     [Checksum Status: Good]
```

Ispis smo malo skratili (za dio koji nosi podatke). Dakle na OSI sloju dva `(2)` vidimo našu MAC adresu `40:8d:5c:57:26:63` te da se paket šalje na MAC adresu od usmjerivača: `d4:76:ea:2e:70:7f` jer je konačna odredišna IP adresa na nekoj drugoj mreži do koje se mora ići preko usmjerivača. Potom na OSI 3 sloju `(3)` vidimo našu IP adresu `192.168.1.103` i odredišnu IP adresu `192.168.200.100`. I konačno na sljedećem sloju vidimo da se radi o ***ICMP*** protokolu, na kojem je vidljivo da se radi o vrsti poruke broj `8` što odgovara: `Echo (ping) request` poruci. Nakon ove poruke slijedi odgovor od druge strane koji je vrlo sličan ovome, s time da mu je u polju: `Type:` broj `0` koji označava odgovor `Echo(ping)reply`, ako je sve u redu.

**Izvor informacija: (960), `man ping` te izvor informacija o ICMP paketu: (803) i RFC 792 te poglavlje: *25.7.1.1*.**

## 25.7.1.1. ICMP poruka

Naredbe `ping`, `ping6` (za IPv6) `tracepath`, `tracepath6` (za IPv6), `traceroute` i `traceroute6` (za IPv6) kao i mnoge druge, oslanjaju se na ***ICMP*** poruke za rad. ***ICMP*** protokol (Engl. *Internet Control Message Protocol*) definiran je u **RFC 792**. Koriste ga razni mrežni uređaji, uključujući usmjerivače i klasična računala, i za slanje poruka o pogreškama kao i operativnim informacijama koje ukazuju na uspjeh ili neuspjeh prilikom komunikacije s odredišnom IP adresom. Na primjer, poruka o pogreški se šalje kada tražena usluga nije dostupna ili kada nije moguće doći do odredišnog računala ili usmjerivača odnosno onoga koga testiramo. ***ICMP*** poruke ne koriste transportne protokole kao što su TCP i UDP za razmjenu *ICMP* poruka između sustava. Dakle *ICMP* u pravilu ne koristi niti TCP niti UDP protokol za transport već se on oslanja sâmo na IP protokol. Dakle *ICMP* zaglavlje se nastavlja na IP zaglavlje (uz Ethernet dio naravno), pa tako cjelokupni paket na OSI sloju jedan (OSI 1) izgleda ovako:

| Ethernet zaglavlje | IP zaglavlje | ICMP zaglavlje | Ethernet zaglavlje |
|---|---|---|---|
| `Preambula, SFD, Izvorišna MAC, Odredišna MAC, ETHER TYPE, …` | `Oznaka protokola, Izvorišna IP, Odredišna IP,…` | **Type** \| **Code** \| **Checksum**<br>**Sadržaj** | `FCS, Interframe Gap, …` |

Vratimo se na *ICMP* poruku odnosno pogledajmo oblik *ICMP* poruke u tablici:

| **Type** | **Code** | **Checksum** |
|---|---|---|
| **Content** | | |

Pri tome svako polje ima određeno značenje (navest ćemo sâmo njih nekoliko)**:**

- **`Type`** - označava vrstu ***ICMP*** poruke uz pripadajući kôd (**`Code`**), a koja može biti (navesti ćemo ih samo nekoliko):
  **`Type 0`** - uz **`Code 0`** je tzv. *Echo Reply* (odgovor na *ping*).
  **`Type 3`** - označava nedostupna odredišta uz sljedeće kôdove:
  - **`Code 0`** - označava nedostupnu mrežu.
  - **`Code 1`** - označava nedostupno računalo.
  - **`Code 2`** - označava nedostupan odredišni protokol.
  - **`Code 3`** - označava nedostupan odredišni port.
  - **`Code 4`** - označava da je potrebna fragmentacija, ali je postavljen ***DF*** bit (***Don't Fragment***).
- **`Type 5`** - označava poruke o preusmjeravanju (Engl. *Redirect*):
  - **`Code 0`** - označava preusmjeravanje za mrežu.
  - **`Code 1`** - označava preusmjeravanje za računalo.
- **`Type 8`** uz **`Code 0`** - označava tzv. *Echo Request* (upit za *ping*).
- **`Type 11`** uz **`Code 0`** - označava istek ***TTL*** vremena paketa tzv. *Time Exceeded*, …
- Postoji cijeli nîz ovih kôdova; i to njih ukupno **255**, koje možete vidjeti popisane na sljedećoj adresi: https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#Control_messages

Što je moguće optimizirati na Linuxu, a vezano je za *ICMP* poruke, pogledajmo pobrojano u sljedećoj tablici (sâmo osnovno):

| Sysctl varijabla za IP v.4 (postoje i za IP v.6.) | Stand vrijed | Opis |
|---|---|---|
| `net.ipv4.icmp_echo_ignore_broadcasts` | 1 | Uklj/Isklj: Zabranjeno odgovaranje na *ICMP multicast* ili *broadcast* poruke (**1**=uključena zabrana). |
| `net.ipv4.icmp_ratelimit` | 1000 | Ograničenje maksimalne brzine slanja *ICMP* poruka čija vrsta je definirana u varijabli: `icmp_ratemask` |
| `net.ipv4.icmp_msgs_per_sec` | 1000 | Maksimalan broj ICMP poruka u sekundi poslan s ovog računala, na mrežu. |
| `net.ipv4.icmp_msgs_burst` | 50 | Maksimalan broj ICMP poruka poslanih odjednom, s ovog računala, na mrežu |

| Sysctl varijabla za IP v.4 (postoje i za IP v.6.) | Stand. vrijed. | Opis |
|---|---|---|
| `net.ipv4.icmp_ratemask` | `6168` | Bit maska koja označava; pogled na bitove:<br>**0** Echo Reply<br>**3** Destination Unreachable<br>**4** Source Quench<br>**5** Redirect<br>**8** Echo Request<br>**B** Time Exceeded<br>**C** Parameter Problem<br>**D** Timestamp Request<br>**E** Timestamp Reply<br>**F** Info Request<br>**G** Info Reply<br>**H** Address Mask Request<br>**I** Address Mask Reply<br>Značajni bitovi: `IHGFEDCBA9876543210`<br>Maska 6168: `0000001100000011000`<br>Označava iz binarnog u decimalno: **`6168`** |

*U slučaju kada ne koristimo **IPv4** već **IPv6** protokol, tada se koriste **ICMP v6** poruke, koje su uvele i nekoliko novih polja u odnosu na **IPv4 ICMP**; poput **Type=135 (neighbor solicitation)** te **TYPE=136 (neighbor advertisement)** i nekoliko drugih, kako je vidljivo i u **RFC4861** koji definira rad **NDP** protokola, koji koristi ova nova polja odnosno funkcionalnosti.*



Pogledajte i poglavlje:
**20.7. ARP protokol.**

Pogledajmo i kako izgledaju *ICMP* paketi na mreži, ako koristimo naredbu: `ping`. Dakle mi ćemo s našeg računala s IP adresom 192.168.1.162 *pingati* DNS poslužitelj na IP adresi 1.1.1.1 i to sâmo s tri paketa. Pogledajmo kako smo to uradili:

```
ping -c 3 1.1.1.1
```

Sada nećemo puno pažnje posvetiti sâmom *pinganju* već *ICMP* paketima koji se u pozadini razmjenjuju.
Istovremeno dok smo slali *ICMP* pakete tj. *pingali* udaljeni poslužitelj: 1.1.1.1, snimali smo samo *ICMP* pakete koji su potom otvoreni u programu ***Wireshark***. Pakete smo snimali na mrežnoj kartici: `enp0s3`, s programom `tcpdump` na sljedeći način:

```
tcpdump -i enp0s3 -s0 icmp -w /tmp/ping.pcap
```

**Slika 236.A. Ping odnosno ICMP poruke, koje smo slali prema odredištu: pogled na snimljene ICMP pakete iz programa Wireshark.**

ping.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000 | 192.168.1.162 | 1.1.1.1 | ICMP | 98 | Echo (ping) request id=0x0443, seq=1/256, ttl=64 (reply in 2) |
| 2 | 0.019128 | 1.1.1.1 | 192.168.1.162 | ICMP | 98 | Echo (ping) reply id=0x0443, seq=1/256, ttl=57 (request in 1) |
| 3 | 1.001662 | 192.168.1.162 | 1.1.1.1 | ICMP | 98 | Echo (ping) request id=0x0443, seq=2/512, ttl=64 (reply in 4) |
| 4 | 1.021457 | 1.1.1.1 | 192.168.1.162 | ICMP | 98 | Echo (ping) reply id=0x0443, seq=2/512, ttl=57 (request in 3) |
| 5 | 2.004263 | 192.168.1.162 | 1.1.1.1 | ICMP | 98 | Echo (ping) request id=0x0443, seq=3/768, ttl=64 (reply in 6) |
| 6 | 2.023430 | 1.1.1.1 | 192.168.1.162 | ICMP | 98 | Echo (ping) reply id=0x0443, seq=3/768, ttl=57 (request in 5) |

Sada pogledajmo prvi paket na slici (naš *ping* s IP adrese: 192.168.1.162 prema IP adresi: 1.1.1.1). Označili smo **(OSI)** slojeve:

```
(1)Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
(2)Ethernet II, Src: 08:00:27:ec:36:32, Dst: d4:76:ea:2e:70:7f
(3)Internet Protocol Version 4, Src: 192.168.1.162, Dst: 1.1.1.1
(4)Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xb414 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1091 (0x0443)
    Identifier (LE): 17156 (0x4304)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
    Timestamp from icmp data: Jul 20, 2020 09:19:10.000000000 Central European Daylight Time
    [Timestamp from icmp data (relative): 0.208901000 seconds]
```

***OSI*** slojeve smo označili sa (**1**)(**2**)(**3**) i (**4**). Prvo pogledajmo OSI sloj tri **`(3)`** odnosno IP protokol u kojem vidimo kako se paket šalje s IP adrese: 192.168.1.162 na odredišnu IP adresu: 1.1.1.1. Nakon trećeg OSI sloja u četvrtom OSI sloju **`(4)`** ne koriste se TCP ili UDP kao transportni protokoli, već se ovdje koristi ***ICMP*** praktično kao transportni protokol. Prvo što nam je vidljivo na ovoj razini je da se radi o poruci (`Echo (ping) request`) koja je tako dekodirana jer se radi o: **`Type 8`** uz **`Code 0`**. Slijedi provjera provjernog zbroja (*checksum*) `[Checksum Status: Good]`.

Identifikatori ove komunikacije su brojevi vidljivi kao `Identifier (BE)` i `Identifier (LE)` pa se oni ne mijenjaju za vrijeme slanje ping-a (*ICMP*-a) odnosno unutar komunikacijskog kanala. Ovo je zapravo ista vrijednost zapisana na dva način: „*Little Endian* (***LE***)" i „*Big Endian* (***BE***)", pri čemu ovaj naziv ***Endian*** označava način zapisivanja nîza bitova podataka. Dakle mi ćemo promatrati sâmo ***BE*** vrijednost.
Ono što se mijenja unutar komunikacijskog kanala je broj sekvence (ponovno ćemo gledati samo ***BE***) odnosno redni broj poruke, vidljiv kao: `Sequence number (BE)`. Dakle konkretno je ovo prva *ICMP* poruka u nîzu (*ping request*) pa ona ima broj `1`. Na kraju slijede vremenske oznake: `Timestamp from icmp data`, pomoću kojih se izračunava vrijeme odziva.

A sada pogledajmo odgovor na našu poruku, koji nam dolazi s IP adrese: 1.1.1.1 (skratili smo ispis samo na **OSI** sloj četiri):

```
(4)Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xbc14 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1091 (0x0443)
    Identifier (LE): 17156 (0x4304)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Request frame: 1]
    [Response time: 19,128 ms]
    Timestamp from icmp data: Jul 20, 2020 09:19:10.000000000 Central European Daylight Time
    [Timestamp from icmp data (relative): 0.228029000 seconds]
```

Ovdje u odgovoru vidimo da je to odgovor, jer imamo **`Type 0`** uz **`Code 0`** što označava: `Echo (ping) reply`.
Zatim vidimo kako je identifikator komunikacije ostao isti: `Identifier (BE): 1091` te da je ostao isti i broj sekvence: `Sequence number (BE): 1`, jer se ovdje radi o odgovoru na prvu poslanu sekvencu. Naime sljedeća *ping* poruka s naše strane (*ping request*) će imati uvećan broj sekvence tj. bit će postavljena na **2**, kao i njen odgovor. Zatim će se stalno povećavati tijekom svakog sljedećeg upita, pri čemu i upit i odgovor imaju broj sekvence. Naime odgovor se referencira na točno određeni redni broj upita. Dakle odgovor nam dolazi za točno određeni upit, odnosno njegov redni broj, što je zapravo oznaka: `Sequence number`.
I na kraju imamo izračunato vrijeme prolaska ICMP paketa od odredišta (IP:1.1.1.1) do nas: `Response time: 19,128 ms`.

**Pogledajmo i drugu malo složeniju primjenu *ICMP* poruka**. Jedan od primjera je upotreba `traceroute` naredbe koju ćemo upoznati u sljedećoj cjelini. Ona osim *ICMP* protokola iskorištava IP polje koje se zove ***TTL*** (Engl. *Time to Live*) koje označava vrijeme života paketa. Naime prolaskom mrežnog paketa kroz svaki pojedini usmjerivač, on čita ***TTL*** polje, smanjuje ga za jedan te kreira novi mrežni paket koji prosljeđuje dalje. Svaki novi usmjerivač radi isto, sve dok ***TTL*** polje ne dođe do vrijednosti jedan (1), a tada ga zadnju usmjerivač smanjuje na nulu (0) i odbacuje, te na izvorišnu adresu šalje poruku o tome.
Upravo to i iskorištava naredba `traceroute`. Dakle ona prvo šalje nîzove po tri ***UDP*** paketa prema odredišnoj IP adresi; u našem slučaju je to IP adresa 1.1.1.1. i to na odredišne portove koji su obično u opsegu od ***33434*** na dalje (svaki novi paket na slijedeći port po redu). S time da prva tri paketa u nîzu imaju postavljen *TTL* na 1, druga tri paketa u nîzu imaju postavljen *TTL* na 2, i tako dalje. Kada prvi nîz od tri UDP paketa koji imaju *TTL* jedan (1), preuzme prvi usmjerivač na koji smo spojeni (obično naš usmjerivač za izlaz na internet), on će *TTL* polje smanjiti za jedan, i pošto će ono sada biti nula (0), on će odbaciti svaki od ta tri paketa, te poslati *ICMP* poruke izvorišnoj IP adresi tj. nama koji smo poslali te pakete, za svaki paket zasebno; dakle tri zasebna ICMP paketa. Naredba `traceroute` će potom poslati novi nîz od tri UDP paketa na odredišnu IP adresu: 1.1.1.1, ali sada s *TTL* poljem koje će imati vrijednost dva (2), pa će ovi paketi proći kroz prvi usmjerivač, koji će smanjiti vrijednost na jedan i paket proslijediti na prvi sljedeći usmjerivač.

Prvi sljedeći usmjerivač će primiti ove pakete i smanjiti im *TTL* za jedan, pa će on biti nula. Zbog toga (*TTL*=0) će ih on odbaciti i poslati nam nazad ICMP poruke (za svaki poslani UDP paket). Naredba `traceroute` će potom poslati novi nîz od tri UDP paketa na odredišnu IP adresu: 1.1.1.1, ali sada s *TTL* poljem koje će imati vrijednost tri (3) i tako dalje, sve dok paketi ne dođu do odredišta.

Pogledajmo kako izgleda prvi od nîza paketa poslanih na odredišnu IP adresu: 1.1.1.1, s označenim **OSI** slojevima: **`(1)`**, **`(2)`**, **`(3)`** i **`(4)`**:

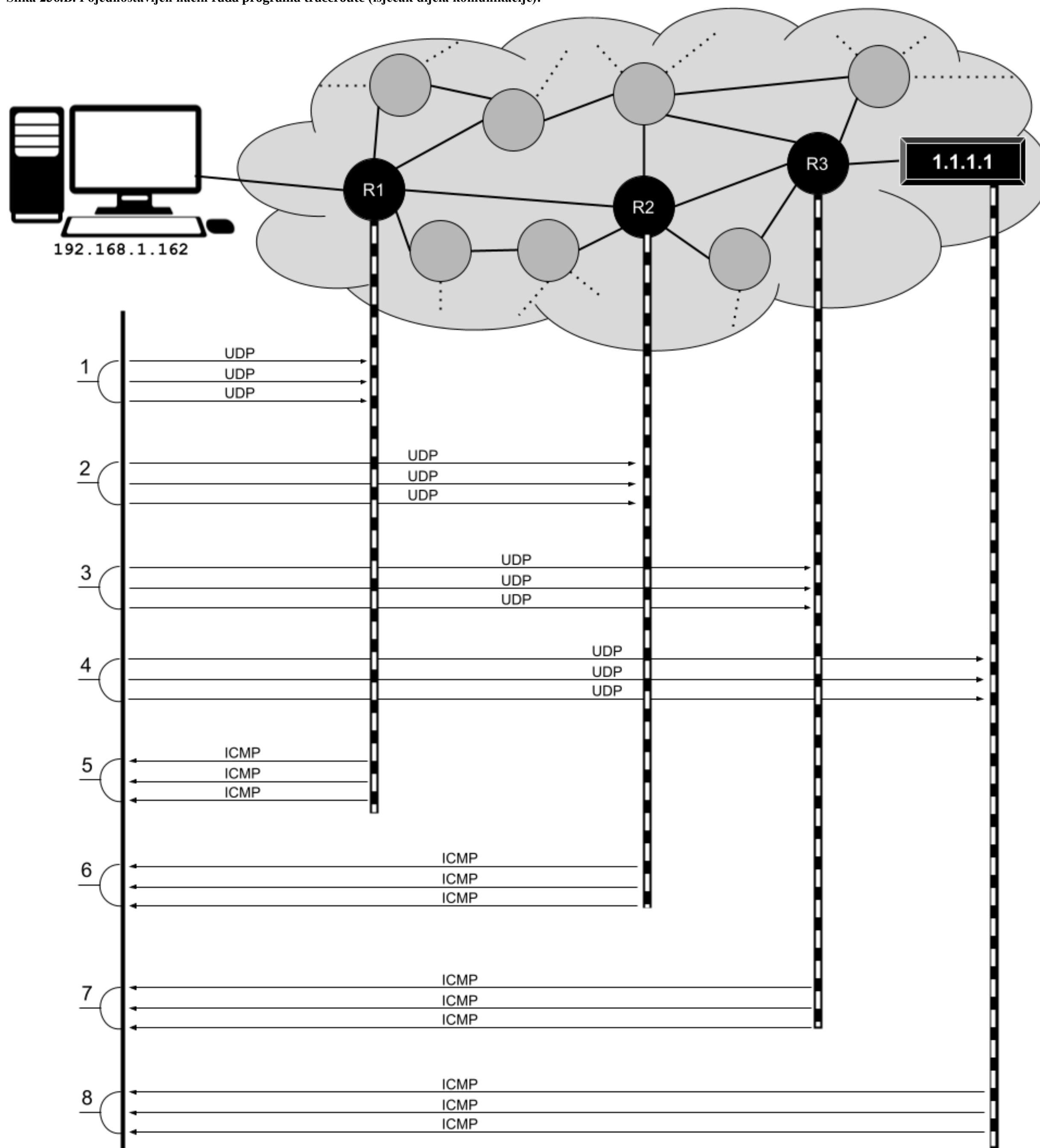
```
(1)Frame 10: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
(2)Ethernet II, Src: 08:00:27:ec:36:32, Dst: d4:76:ea:2e:70:7f
(3)Internet Protocol Version 4, Src: 192.168.1.162, Dst: 1.1.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xd2fb (54011)
    Flags: 0x0000
    Fragment offset: 0
    Time to live: 1
    Protocol: UDP (17)
    Header checksum: 0x226a [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.1.162
    Destination: 1.1.1.1
(4)User Datagram Protocol, Src Port: 39800, Dst Port: 33434
```

I sada pogledajmo prvi od ***ICMP*** odgovora, na prvi poslani *UDP* paket sa postavljenom vrijednosti: `Time to live: 1`.

```
(1)Frame 28: 102 bytes on wire (816 bits), 102 bytes captured (816 bits)
(2)Ethernet II, Src: d4:76:ea:2e:70:7f, Dst: 08:00:27:ec:36:32
(3)Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.162
(4)Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0xb985 [correct]
    [Checksum Status: Good]
  → Internet Protocol Version 4, Src: 192.168.1.162, Dst: 1.1.1.1
  → User Datagram Protocol, Src Port: 39800, Dst Port: 33434
```

Dakle vidimo *ICMP* paket koji nam šalje prvi usmjerivač: 192.168.1.1, kao odgovor, a čiji **`Type 11`** uz **`Code 0`** govori kako se radi o poruci `Time to live exceeded in transit`. Nadalje, na *ICMP* sloju paketa sadržani su ugniježđeni i ***IP*** i ***UDP*** slojevi (→) kao dio odgovora. Pogledajmo i skraćenu pojednostavljenu logičku shemu rada programa `traceroute` (sl. **236.B**).

**Slika 236.B. Pojednostavljen način rada programa traceroute (isječak dijela komunikacije):**




Statistike ICMP poruka (prema vrstama i kategorijama) možemo pratiti s naredbom `nstat` primjerice ovako:

```
nstat -az | grep -i icmp
```

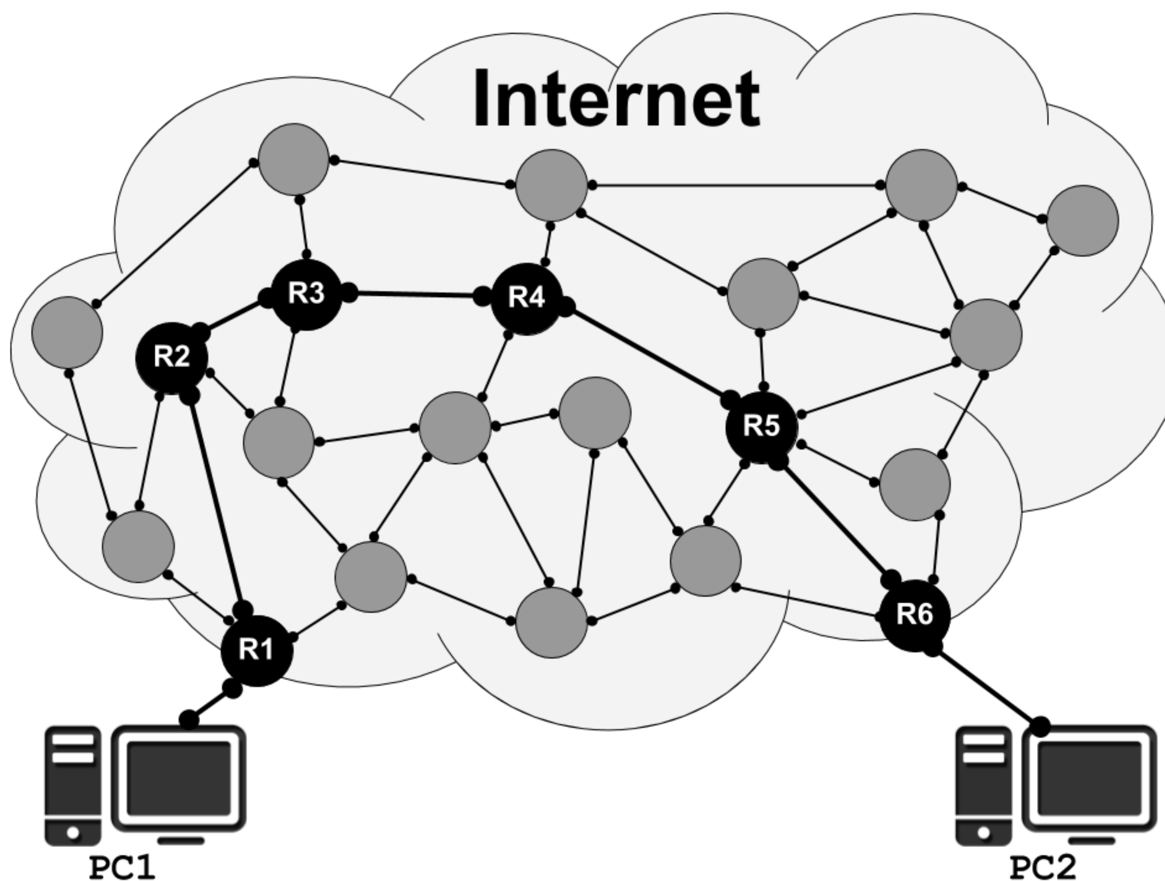
**Izvor informacija: (642),(1650), `man 7 icmp`, `man ping` te izvori informacija o ICMP paketu: (IPv4:803) i RFC 792.**

## 25.7.2. *Naredbe traceroute* i *tracepath*

Za razliku od naredbe `ping` koja provjerava dostupnost krajnje točke u komunikaciji, naredba `traceroute` prikazuje nam sve točke (*route*) kroz koje prolazi paket do odredišta uz mjerenje kašnjenja do svake od tranzitnih točaka odnosno usmjerivača (*routera*) tj. uređaja koji rade na OSI sloju tri. Na nekim sustavim ne postoji naredba `traceroute` koja dolazi u softverskom paketu `traceroute` već samo naredba `tracepath` koja dolazi u novijem softverskom paketu `iputils`.

Zamislimo sljedeću pojednostavljenu topologiju mreže na internetu, vidljivu na slici 236.

**Slika 236.1 Topologija mreže interneta (pojednostavljeno).**




Ako naše računalo (**PC1**) želi napraviti `traceroute` na drugo računalo **PC2** koje je negdje na internetu, a paket mora proći kroz 6 usmjerivača (***R1 – R6***), mjerit će se vrijeme koje je potrebno od našeg računala koje inicira mjerenje, do svakog od njih pojedinačno i na kraju do odredišta prema sljedećem principu:

Računalo 1 (**PC1**) ←→ Usmjerivač/*Router* 1 (**R1**)

Računalo 1 (**PC1**) ←→ Usmjerivač/*Router* 2 (**R2**)

Računalo 1 (**PC1**) ←→ Usmjerivač/*Router* 3 (**R3**)

Računalo 1 (**PC1**) ←→ Usmjerivač/*Router* 4 (**R4**)

Računalo 1 (**PC1**) ←→ Usmjerivač/*Router* 5 (**R5**)

Računalo 1 (**PC1**) ←→ Usmjerivač/*Router* 6 (**R6**)

Računalo 1 (**PC1**) ←→ Računalo 2 (**PC2**)

Zapravo se na svaku točku u prijenosu odnosno usmjerivač šalju tri paketa. Ovisno o implementaciji, obično se ispisuju sva tri mjerenja za svaku točku. U slučaju da sva tri mjerenja po mjernoj točki (usmjerivaču) ne budu obrađena, označavaju se tri znaka o neuspjeloj konekciji (`*`) na problematičnoj mjernoj točki. `Traceroute` na Linuxu šalje slične poruke kao i naredba `ping` za koje se koristi isti protokol (*ICMP*) uz drugu podvrstu poruka, ali i uz eventualnu mogućnost odabira: UDP poruke ili TCP poruke. Ovaj odabir postoji zbog toga što je moguće da neki usmjerivači u nîzu, zbog sigurnosnih i drugih razloga imaju isključen rad s klasičnim *ICMP* porukama.

Tada će u odgovorima biti samo tri zvjezdice (`* * *`) koje označavaju grešku za svaki usmjerivač koji nije odgovorio. Upotrebom UDP ili TCP poruka taj problem se može riješiti jer se više ne šalje *ICMP* poruka (***ICMP ECHO***) nego eventualno posebni TCP odnosno UDP paketi.

Pogledajmo i primjer u kojem ćemo probati napraviti `traceroute` na odredišnu adresu `google.com`
Napomena: prekidač `-n` znači da ne želimo da se IP adrese prevode (engl. *Resolving*) u imena (*FQDN*) usmjerivača, već da imamo sâmo prikazane IP adrese usmjerivača kroz koje paketi prolaze.

```
traceroute -n google.com

traceroute to google.com (212.92.207.155), 30 hops max, 60 byte packets
 1  192.168.100.1  0.051 ms  0.014 ms  0.012 ms
 2  192.168.200.1  3.014 ms  4.342 ms  4.511 ms
 3  10.51.192.1  2.824 ms  2.813 ms  2.798 ms
 4  10.10.4.17  5.841 ms  5.833 ms  5.865 ms
 5  10.10.0.29  5.795 ms  5.915 ms  5.943 ms
 6  10.50.0.73  6.374 ms  6.332 ms  6.362 ms
 7  10.50.0.74  6.201 ms  5.642 ms  5.602 ms
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

Vidljivo je da sve do usmjerivača broj 7 imamo mjerenja, a dalje je sve zablokirano (za *ICMP*). Inače se kod svih mjerenja pokušava koristiti maksimalno do 30 točaka (usmjerivača) do odredišta (tkzv. *Hop*-ovi). Zbog toga engl. naziv "*Hop count*". Ovo ograničenje je po potrebi moguće i proširiti. Ograničenje je povezano s poljem ***TTL*** (Engl. *Time To Live*) koje označava koliko maksimalno usmjerivača neki mrežni paket može proći prije nego bude odbačen od strane prvog sljedećeg usmjerivača. Stoga se s tim parametrom prema potrebi može proširiti broj usmjerivača, a parametar je `-m` **broj**.

Naime svaki paket na mreži ima postavljen ***TTL*** (*Time To Live*) broj odnosno vrijeme dozvoljenog života paketa. Prolaskom paketa kroz svaki usmjerivač na putu, ovaj broj usmjerivač smanji za jedan, sve dok se ne dođe do nule. Ako i kada se došlo do nule, trenutni usmjerivač odbacuje paket. Drugim riječima, ako je nekom paketu trebalo do odredišta preveliki broj usmjerivača on će biti odbačen. Naredbe `traceroute` i `tracepath` rade malo drugačije, pa mogu davati nešto drugačije rezultate.



Vezano za ***TTL*** polje unutar **IP** zaglavlja paketa, pogledajte poglavlje:
**23.4.1 Time to *live (TTL)*.**

Probajmo sada isti test, ali korištenjem ***TCP*** protokola, pomoću prekidača `-T`.
U ovom slučaju šalje se *TCP SYN* paket za otvaranje TCP konekcije (**opisano u poglavlju: 24.2.2.**).

```
traceroute -n -T google.com

traceroute to google.com (212.92.207.152), 30 hops max, 60 byte packets
 1  192.168.100.1  0.050 ms  0.008 ms  0.007 ms
 2  192.168.200.1  3.444 ms  4.108 ms  4.283 ms
 3  10.51.192.1  2.213 ms  2.255 ms  2.391 ms
 4  10.10.4.17  5.729 ms  5.767 ms  5.803 ms
 5  10.10.0.29  6.256 ms  6.209 ms  5.895 ms
 6  10.50.0.73  6.982 ms  6.796 ms  6.821 ms
 7  10.50.0.74  6.653 ms  5.713 ms  5.626 ms
 8  212.92.207.152  5.661 ms  5.348 ms  5.280 ms
```

Sada vidimo kako imamo mjerenja do krajnje točke (točka `8`) što znači i kako nam do odredišta treba osam usmjerivača.

**Izvor informacija: (1044), `man 7 icmp`, `man tracepath`, `man traceroute`.**

## 25.7.3. *Naredba nslookup*

Naredba `nslookup` (engl. *name server lookup*) koristi se za provjeru ispravnosti ili dostupnosti unosa na *DNS* (engl. *Domain Name System*) poslužitelju. Pomoću ove naredbe kontaktiramo *DNS* poslužitelj da nam razriješi IP adresu na osnovi imena računala ili ***FQDN*** imena tj. imena računala s njegovom pripadajućom domenom. U slučajevima kada sumnjamo da nam *DNS* poslužitelji nisu dobro konfigurirani ili nam zbog nekog drugog razloga ***DNS*** razlučivanje (*resolving)* ne radi, ovo je alat koji nam može pomoći. Pogledajmo primjer u kojemu želimo pronaći IP adresu poslužitelja: **`opensource-osijek.org`**

```
nslookup opensource-osijek.org

Server:         195.29.166.116
Address:        195.29.166.116#53

Non-authoritative answer:
Name:   opensource-osijek.org
Address: 213.147.104.78
```

Ovdje vidimo da je `nslookup` pronašao prvi ***DNS*** poslužitelj konfiguriran na sustavu (`195.29.166.116`) te da je njemu poslao upit da mu da točnu IP adresu od FQDN imena: `opensource-osijek.org` računala.

Dobili smo odgovor da je od računala `opensource-osijek.org`, njegova pripadajuća IP adresa: `213.147.104.78` što je točno (*u trenutku pisanja*). Iz ovoga možemo reći da nam je ***DNS*** poslužitelj dobro konfiguriran i da nam odgovara, te da smo dobili traženi odgovor.

Napomena:

- `Non-authoritative answer` znači da smo dobili odgovor od *DNS* poslužitelja koji nije zadužen za tu domenu (domenu **`.org`** u ovom slučaju) te da je on morao pitati viši *DNS* poslužitelj (viši u *DNS* hijerarhiji)(**Poglavlje: 25.8.5.**).
- `Authoritative answer` značilo bi da smo dobili odgovor od *DNS* poslužitelja koji je zadužen za tu domenu za koju smo slali upit te da on nije morao pitati viši *DNS* poslužitelj u hijerarhiji(**Također opisano u poglavlju: 25.8.5.**).

U drugom primjeru promijenimo *DNS* poslužitelj koji želimo da se kontaktira, u novi DNS poslužitelj čija je IP: `1.1.1.1`.

Potom mu pošaljimo isti upit pozivajući naredbu `nslookup` na sljedeći način (upisujemo naredbe nakon znaka >):

```
nslookup

> server 1.1.1.1
Default server: 1.1.1.1
Address: 1.1.1.1#53

>  opensource-osijek.org
Server:         1.1.1.1
Address:        1.1.1.1#53

Non-authoritative answer:
Name:   opensource-osijek.org
Address: 213.147.104.78
> exit
```

Naredba `nslookup` može se pokrenuti i bez parametara, a tada ulazimo u njeno naredbeno sučelje (***CLI***).
*DNS* poslužitelj smo tada promijenili naredbom `server` iza koje slijedi IP adresa *DNS* poslužitelja (`1.1.1.1`).

Nakon toga očekuje se upit punog imena računala (***FQDN***) koje provjeravamo, pa stoga unosimo: `opensource-osijek.org`.
U slučaju kada vaš Linux nema instaliranu naredbu `nslookup` morate instalirati *bind-utils* paket, sa sljedećom naredbom:

```
yum -y install bind-utils
```



**Dodatni primjeri upotrebe su vidljivi u poglavlju:**
**25.8.5.5. DNS zapisi (*DNS records*).**

**Izvor informacija:** **`man nslookup`**.

### 25.7.4. *Naredba* netstat

Naredba `netstat` (engl. *Network Statistics*) prikazuje nam razne parametre rada mreže te mrežne statistike na OSI slojevima 3 i 4 (mrežni i transportni sloj). Mada smo se s njom već susreli, proći ćemo neke njene osnove. Dakle ona nam daje:

- IP konfiguraciju mrežnih sučelja (kartica).
- Tablice usmjeravanja (*routing tablice*).
- Status mrežnih konekcija.
- Statistike vezane za mrežne protokole, pakete, kao i druge podatke, od trenutka pokretanja sustava.

Postoji i cijeli nîz parametara od kojih ćemo spomenuti sâmo neke:

| Parametar (prekidač) | Opis |
|---|---|
| `-a` | Prikazuje sve mrežne veze (konekcije), te TCP i UDP portove. |
| `-e` | Prikazuje "*Ethernet*" parametre mreže te statistiku o količini primljenih/poslanih podataka. |
| `-g` | Prikazuje statistiku za *Multicast*. |
| `-l` | Prikazuje samo portove (servise) koji su u stanju slušanja to jest otvoreni su (engl. *Listen*). |
| `-r` | Prikazuje tablicu usmjeravanja (*Routing* tablicu). |
| `-i` | Prikazuje statistiku za mrežna sučelja i mrežne kartice. |
| `-n` | Sve konekcije koje prikazuje, prikazuje s IP adresama, a ne s imenima računala (*hostname*). |
| `-p` | Prikazuje statistike prema protokolu uz ispis servisa (*daemona*) zaduženog za konekciju. |
| `-s` | Prikazuje detaljne statistike mrežnog prometa tj. paketa (pr. za TCP/UDP/IP: ispravno, neispravno, greške, ... ). |
| `-t` | Prikazuje samo TCP pakete. |
| `-u` | Prikazuje samo UDP pakete. |
| `-W` | Ne skraćuje prikaz ispisa IPv6 adresa (može biti korisno ako koristite IPv6 adrese). |
| `-o` | Prikazuje informacija o brojačima. Pogledajte poglavlje: **24.2.15. TCP keepalives.** |

Opis svih opcija i parametara naredbe `netstat`, dostupan je na izvoru informacija: **(732).**

**Primjeri**

**1.** Prikaži sve mrežne konekcije (`-a`), ali u brojčanom formatu s IP adresama (`-n`), upotrebom naredbe `netstat`:

```
netstat -an

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:389             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.101:80        123.125.71.35:6851      SYN_RECV
tcp        0      0 127.0.0.1:38771         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.101:443       144.76.137.226:43556    TIME_WAIT
tcp        0      0 192.168.1.101:48388     192.168.1.131:514       ESTABLISHED
```

**2.** Prikaži sve mrežne konekcije (`-a`), ali u brojčanom formatu (`-n`) s time da vidimo koji servis (*daemon*) je otvorio koji TCP/UDP port (`-p`) što je vidljivo u zadnjem stupcu (`PID/Program name`):

```
netstat -anp

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:199           0.0.0.0:*               LISTEN      610/snmpd
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      776/mysqld
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      858/httpd
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      639/vsftpd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      620/sshd
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      821/sendmail
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      858/httpd
tcp        0      0 192.168.1.101:443       144.76.137.226:48515    TIME_WAIT   -
tcp        0      0 192.168.1.101:48388     192.168.1.131:514       ESTABLISHED 493/rsyslogd
tcp        0      0 127.0.0.1:389           127.0.0.1:42370         ESTABLISHED 596/slapd
```

Iz prethodnog ispisa, vidljivo je primjerice, da je **mysql** baza podataka (`mysqld` servis/*daemon*) na IP adresi: `0.0.0.0` (što znači ovo računalo: na svim IP adresama svih mrežnih sučelja) i to pokrenuto na portu broj `3306`. Osim toga vidimo kako je Web poslužitelj `httpd` (*Apache*), također na IP adresi: `0.0.0.0` (što isto označava ovo računalo), ali na portu broj `80`.
Osim toga vidimo i u kojem su stanju te konekcije, poput onih u stanju slušanja (`LISTEN`) te druge konekcije u stanju spojeno (`ESTABLISHED`) kao i konekcije u posebnom stanju čekanja (`TIME_WAIT`).

**Slijedi napredni primjer!**
Ako mrežne statistike gledamo kao u primjeru broj **2**, možemo uočiti dva stupca: `Recv-Q` i `Send-Q`. Sada ćemo pogledati dio statistike, samo za naše računalo (`192.168.1.101`), u trenutku kada se ono spaja (s programom `wget`) na udaljeni web poslužitelj (`199.232.18.132`) te s njega dohvaća sadržaj s web stranice. Stoga ćemo napraviti dodatno filtriranje:

```
netstat -anp | grep wget

Proto Recv-Q  Send-Q Local Address        Foreign Address     State        PID/Program
tcp   1709176 0      192.168.1.101:37288 199.232.18.132:80   ESTABLISHED  1866/wget
```

Pogledajmo o čemu se ovdje radi:

- `Recv-Q` - prikazuje bajtove podataka koji su zaprimljeni i čekaju u redu za slanje (*queue*) prema korisničkom programu koji je uspostavio vezu. Dakle ovo znači kako program spojen na taj mrežni *socket* još nije dohvatio ove podatke. Ova vrijednost treba biti što bliže 0. Na zauzetim poslužiteljima ta će vrijednost biti veća od 0, ali ne bi trebala biti vrlo visoka. Veći brojevi ovdje možda ne znači mnogo, osim ako vidite veliki broj u dolje opisanom stupcu `Send-Q`. Međutim u gornjem konkretnom primjeru to znači da je kernel na portu `37288` primio `1709176` bajta podataka, ali da ih proces (program) [`wget` s ***PID*** brojem `1866`] koji pristupa web stranci (na adresi `199.232.18.132`), još nije kopirao u svoju memoriju i počeo s obradom. To može indicirati da aplikacija (u ovom slučaju navedeni program `wget`) trenutno ne može dohvatiti podatke jer je prezauzet ili je računalo zagušeno, što je i bio slučaj u našem primjeru. U ovakvom slučaju, ako prijemna strana nije u stanju obraditi te pakete, oni će biti vidljivi kao odbačeni i s naredbom `nstat` (opisanom u sljedećem poglavlju):
  ```
  nstat -az TcpExtTCPRcvQDrop
  ```
- `Send-Q` - označava bajtove podataka koji se trenutno nalaze u redu za slanje udaljenom programu, jer vjerojatno udaljeni program još nije potvrdio njihov primitak (s ***TCP ACK*** porukom). Ova vrijednost bi trebala biti blizu 0. Veliki brojevi ovdje mogu ukazivati na probleme u mreži to jest najčešće u propusnosti mreže.



Za analizu uzroka potencijalnih problema u radu mreže ili mrežnih aplikacija, pogledajte poglavlje:
**25.3. Statistike, analiza i praćenje mrežnih paketa.**

Pogledajte i parametre kernela u poglavlju:
**24.2.8.2. Skaliranje TCP prozora (*Window Scaling*).**

**3.** Prikažimo ukupnu statistiku mrežnog prometa za sva mrežna sučelja (`-i`):

```
netstat -i

Kernel Interface table

Iface       MTU Met    RX-OK  RX-ERR RX-DRP RX-OVR   TX-OK     TX-ERR  TX-DRP  TX-OVR   Flg

eth0       1500   0 678732758      0      0      0   663110829      0       0       0   BMRU
lo        16436   0 194960446      0      0      0   194960446      0       0       0   LRU
```

Vidimo kako na sustavu postoje dva mrežna sučelja: fizičko `eth0` i logičko/softversko `lo` (*loopback*), za koja vidimo i statistike, koliko mrežnog prometa je primljeno i poslano s njih i na njih te je li bilo nekih grešaka (pr. `RX-ERR`, `RX-DRP`, …).

**4.** Ispišimo tablicu usmjeravanja (`-r`) za cijelo računalo:

```
netstat -rn

Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG        0 0          0 eth0
```

Vidimo kako je *Default Gateway* odnosno podrazumijevani usmjerivač (*Destination IP:* `0.0.0.0` i *Genmask:* `0.0.0.0`) postavljen na IP adresu `192.168.1.1` preko `eth0` mrežne kartice, kao i dodatni unos u tablici za mrežu 192.168.1.0 / 255.255.255.0 koja isto pokazuje na `eth0` mrežnu karticu.
Ovo je mreža na kojoj se nalazi i ovo računalo, pa je normalno da se njoj pristupa preko mrežnog sučelja `eth0`.



Za još neke od primjera upotrebe pogledajte sljedeće poglavlje, ali i poglavlja:
**9.4.3.1. Unix socketi.**
**24.2.13. Mehanizam TCP retransmisije.**
**24.2.16.1. Timeri i stanja veze.**
**25.3 Statistike, analiza i praćenje mrežnih paketa.**
**25.6.5.2.1. Privremena konfiguracija korištenjem DHCP poslužitelja.**

**Izvor informacija:** **`man netstat`**,**`man nstat`**,**`man 7 tcp`**,**`man 7 udp`**,**`man 7 ip`**.

### 25.7.5. Naredba *nstat*

Naredba `nstat` je novija varijanta dijela naredbe `netstat` (engl. *Network Statistics*) i prikazuje nam mrežne statistike na OSI slojevima 3 i 4 (mrežni i transportni sloj) te poruke kernela vezane za mrežu, od trenutka pokretanja sustava, na vrlo detaljan način. Ova naredba dolazi u paketu `iproute2` (zajedno s naredbama poput: `ip`, `bridge`, `lnstat` i drugima). Za nju postoji nekoliko parametara rada. Pogledajmo primjere njene upotrebe.

**1. Prikaži sve mrežne statistike** (`-az`), čak i prazne brojače (s **0** paketa), upotrebom naredbe `nstat`:

```
nstat -az

IpInReceives                    10566              0.0
IpInDelivers                    10566              0.0
IpOutRequests                   10616              0.0
IpOutNoRoutes                   32                 0.0
IcmpInErrors                    60                 0.0
IcmpOutErrors                   151                0.0
TcpActiveOpens                  633                0.0
TcpPassiveOpens                 616                0.0
TcpEstabResets                  278                0.0
TcpInSegs                       8062               0.0
TcpOutSegs                      7989               0.0
TcpOutRsts                      1178               0.0
UdpInDatagrams                  3117               0.0
UdpNoPorts                      124                0.0
UdpOutDatagrams                 2466               0.0
TcpExtDelayedACKs               25                 0.0
TcpExtTCPHPHits                 416                0.0
TcpExtTCPPureAcks               1925               0.0
IpExtInOctets                   3248600            0.0
IpExtOutOctets                  2884655            0.0
IpExtInMcastOctets              92855              0.0
IpExtOutMcastOctets             46335              0.0
IpExtInBcastOctets              150899             0.0
```

Kao što je vidljivo, statistike su prilično detaljne (i slične kao i statistike naredbe `netstat -s`), a određeni dio od njih opisali smo u prethodnim poglavljima.



Za neke od detalja ispisa statistike mrežnih paketa pogledajte i poglavlje:
**25.3 Statistike, analiza i praćenje mrežnih paketa.**
Detaljan opis svih statistika, pogledajte na izvoru informacija: **(732)**

Pogledajte i:
**25.5.6. Optimizacije mrežnih sučelja propusnosti veće od 10Gbps.**

S obzirom da ova naredba s kernelom komunicira na nižoj razini od naredbe `netstat`; ona je u mogućnosti dohvaćati i dodatne statistike (sve vidljive u gore vidljivom ispisu) i to znatno brže.

**2. Prikaži samo određene mrežne statistike**
`nstat` nam daje mnoge detalje o konekcijama, tako se primjerice može provjeriti ima li odbacivanja paketa u prijemnom nizu (vidljivom kao **Recv-Q**):

```
nstat -az TcpExtTCPRcvQDrop
```

Drugi napredni primjer bi bila provjera ima li odbačenih paketa zbog prepunjavanja tzv. *backlog* međumemorije (definirane u *sysctl* varijabli: `net.core.netdev_max_backlog`):

```
nstat -az TcpExtTCPBacklogDrop
```



Za detalje oko *backlog* međumemorije, pogledajte poglavlje:
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

**Izvor informacija: (732), `man nstat`,`man 7 tcp`,`man 7 udp`,`man 7 ip`.**

## 25.7.6. Naredba *list open files* (*lsof*)

Iako `lsof` nije naredba koja ima direktnu vezu s mrežom, ali pošto se na UNIX/Linux sustavima sve svodi na pristup nekim (vrstama) datoteka, tako su i mrežne konekcije i otvoreni portovi vidljivi kao posebne (otvorene) datoteke zvane *file deskriptori*. Drugim riječima naredba **Li**S**t** **O**pen **F**iles (***LSOF***) nam daje mogućnost da vidimo sve otvorene mrežne konekcije i datoteke koje su vezane za te konekcije. Nadalje možemo vidjeti koji program je otvorio koji mrežni port odnosno koji servis (*daemon*) je ostvario mrežnu konekciju i slično.
Za provjeru, koju inačicu naredbe `lsof` imamo i s kojim parametrima je kompilirana, pokrenite sljedeću naredbu:

```
lsof -v

lsof version information:
    revision: 4.82
    latest revision: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/
    latest FAQ: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/FAQ
    latest man page: ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/lsof_man
    constructed: Thu Jul 23 17:38:32 UTC 2015
    constructed by and on: mockbuild@c6b8.bsys.dev.centos.org
    compiler: cc
    compiler version: 4.4.7 20120313 (Red Hat 4.4.7-16) (GCC)
    compiler   flags:   -DLINUXV=26016   -DGLIBCV=212   -DHASIPv6   -DHASSELINUX   -
D_FILE_OFFSET_BITS=64 -D_LARGEFILE64_SOURCE -DHAS_STRFTIME -DLSOF_VSTR="2.6.16" -O2 -
g  -pipe  -Wall  -Wp,-D_FORTIFY_SOURCE=2  -fexceptions  -fstack-protector  --param=ssp-
buffer-size=4 -m64 -mtune=generic -fno-strict-aliasing

loader flags: -L./lib -llsof  -lselinux
    system info: Linux c6b8.bsys.dev.centos.org 2.6.32-220.el6.x86_64 #1 SMP Tue Dec 6
19:48:22 GMT 2011 x86_64 x86_64 x86_64 GNU/Linux. Anyone can list all files.
/dev warnings are disabled.
Kernel ID check is disabled.
```

Nadalje, pokretanjem naredbe `lsof` s prekidačem `-i` dobivamo popis odnosno listu svih otvorenih datoteka koje su vezane za *Internet* konekcije odnosno mrežu. Dodatno ćemo koristiti i prekidač `-n` kako bismo sva računala vidjeli s numeričkim oznakama odnosno s IP adresama, a ne imenima računala s kojih ili od kojih je mrežna konekcija otvorena. Ako koristimo i prekidač `-P` bit će prikazani i brojevi portova, a neće se prikazivati ime servisa koji je odgovoran za njih; primjerice: port ***22*** ili oznaka ***ssh***.

Pogledajmo primjer u kojem na našem poslužitelju na kojem pokrećemo ovu naredbu imamo IP adresu 192.168.100.1 i želimo vidjeti sve otvorene datoteke koje su vezane za IP protokol odnosno komunikaciju prema i od nas, odnosno naše IP adrese:

```
lsof -i -n -P

COMMAND    PID   USER FD TYPE  DEVICE SIZE/OF NODE NAME
rsyslogd   485   root 2u IPv4  420783  0t0    TCP  192.168.100.1:49743->192.168.100.100:514 (ESTABLISHED)
nginx     8886   root 6u IPv4 1687513  0t0    TCP  192.168.100.1:80 (LISTEN)
nginx     8888  nginx 6u IPv4 1687513  0t0    TCP  192.168.100.1:80 (LISTEN)
sshd     13414   root 3r IPv4 1696814  0t0    TCP  192.168.100.1:22->93.137.225.169:60010 (ESTABLISHED)
sshd     22746   root 3u IPv4 8937142  0t0    TCP  *:22 (LISTEN)
```

Ispis smo skratili te pogledajmo opis onoga što vidimo:

- Pogledajmo prvi red. Prva vidljiva naredba (u stupcu `COMMAND`) je `rsyslogd`. To je servis zadužen na primanje (ili slanje) tzv. "*Syslog*" poruka. U stupcu `NODE` vidimo da se ovdje radi o TCP konekciji. U stupcu `NAME` vidimo s koje izvorišne IP adrese se spajamo na koju udaljenu IP adresu. Dakle naš server: `192.168.100.1`, lokalni port `49743`, se spojio na udaljeni server: `192.168.100.100`, port `514`, pri tome je vrsta konekcije `ESTABLISHED` dakle radi se o ostvarenoj i trenutno aktivnoj konekciji na udaljeni poslužitelj (server).
- U drugom i trećem redu, pod stupcem `COMMAND` je `nginx`, a to je servis koji je naš lokalni Web poslužitelj i koji koristi TCP transportni protokol. Za njega vidimo da je pokrenut na IP adresi `192.168.100.1` i to na portu `80` (to je *HTTP* protokol) i da je u stanju `LISTEN` što znači da sluša i prihvaća nove konekcije.
- Sljedeći pod stupcem `COMMAND` su *ssh* servisi (`sshd`) i to:
    - Prvi po redu (onaj s `PID` brojem `13414`) je ostvarena konekcija (u stanju je `ESTABLISHED`) između našeg servera: `192.168.100.1`, a port je `22` što je *ssh* servis. Druga strana *ssh* tunela/konekcije je s IP adrese: `93.137.225.169`, na udaljenom (*remote*) portu `60010`.
    - Sljedeći je (***PID*** `22746`) sâm *SSH servis* (`sshd`) za koji vidimo da sluša na svim IP adresama lokalnog servera (`*:22` označava sve lokalne IP adrese, a port je *ssh* tj. `22`). Stanje `LISTEN` znači da osluškuje nove *ssh* veze.
- Stupac `USER` označava korisničko ime s kojim je pokrenut proces odnosno servis.

**Stupac FD**
Stupac `FD` nam daje detalje o "*File Deskriptorima*"



Podsjetite se ***file deskriptora*** u poglavlju:
**4.5.4. Opisnici datoteke (*File deskriptori)*.**

Moguće vrijednosti su :

- `cwd` - predstavlja trenutni radni direktorij.
- `rtd` - predstavlja *root* direktorij.
- `txt` - predstavlja ***text*** dio programa (dio programa koji sadrži programski kôd i podatke izvršnog programa).
- `mem` - predstavlja "*memory-mapped*" odnosno memorijski povezanu datoteku.

Osim navedenoga, ovaj stupac koji označava *file deskriptore* može imati i druga stanja poput `1u` koje znači da se radi o *file deskriptoru* brojčane oznake `1`. Brojčana oznaka odnosno broj *file deskriptora* je jedinstven na razini svakog pojedinog procesa po ***PID*** broju (vidljivo u /proc/***PID***/fd/ direktoriju). Iza ove brojke slijedi njihovo stanje rada: **`u`**, **`r`** ili **`w`** što znači:

- `r` - za prava čitanja (*read access*); što znači kako je taj *file deskriptor* u stanju čitanja.
- `w` - za prava pisanja (*write acc*ess); što znači kako je taj *file deskriptor* u stanju zapisivanja.
- `u` - za prava čitanja i pisanja (*read and write*); što znači kako je taj *file deskriptor* u stanju čitanja i zapisivanja.
- `W` - datoteka je otvorena s ovlastima za zapisivanje, ali je zaključana (*Write Lock on entire file*).

**Stupac TYPE**
Stupac `TYPE` definira vrstu datoteke i njenu identifikaciju. Moguće vrijednosti su:

- `DIR` - predstavlja direktorij.
- `REG` - predstavlja običnu (standardnu) datoteku.
- `CHR` - predstavlja "*Character*" vrstu posebne datoteke.
- `FIFO` - predstavlja "*First In First Out*" vrstu posebne datoteke.

**Stupac DEVICE**
Stupac `DEVICE` definira vrstu uređaja s ***MAJOR*** i ***MINOR*** identifikatorima.



Pogledajte i poglavlje:
**11.1.2 Uređaji (devices) ukratko** .

**Stupac SIZE/OFF**
Stupac `SIZE/OFF` govori kolika je veličina datoteke ili njen *offset*. Za normalne datoteke je to određena vrijednost, dok je za mrežne konekcije vrijednost uvijek nula (`0t0`). Točnije, ako je u pitanju veličina datoteke ona je normalno iskazana u bajtima. Ako se pak radi o *offsetu* tada obično postoje dvije mogućnosti:

- `0t` - uz vrijednost *offseta* je oznaka za decimalnu oznaku *offseta* (obično za *offset* od 8 ili manje brojeva).
- `0x` - uz vrijednost *offseta* je oznaka za heksadecimalnu oznaku *offseta* (obično za veći *offset*).

**Stupac NODE**
Stupac `NODE` predstavlja *INODE* unos u datotečnom sustavu, za konkretnu datoteku:

- Za normalne *file deskriptore* - ovo je stvarno *INODE* broj datoteke na datotečnom sustavu.
- Za mrežne *file deskriptore* - ovo može biti:
  - `TCP` - za *TCP* konekcije.
  - `UDP` - za *UDP* konekcije.

**Stupac NAME**
Stupac `NAME` predstavlja:

- Ime otvorene datoteke:
  - Primjerice biblioteke: /lib/libutil-2.12.so
  - Ili primjerice izvršne datoteke: /usr/sbin/sshd
- Posebne datoteke koje predstavljaju neku funkcionalnost (primjerice /dev/null).
- Za mrežne konekcije:
  - Predstavlja sâm otvoreni port; primjerice: `*:ssh` **`(LISTEN)`**
  - Ili uspostavljenu konekciju; primjerice: `192.168.1.170:ssh → 192.168.1.164:9926` **`(ESTABLISHED)`**.

**Filtriranje prema transportnom protokolu**
Moguće je i ograničiti ispis samo na vrstu transportnog protokola: ***TCP*** ili ***UDP*** (isto ćemo koristiti i `-n` zbog ispisa u obliku IP adrese, a ne imena računala u komunikaciji):

Za ***TCP*** transportni protokol pokrenimo naredbu `lsof` na sljedeći način:
```
lsof -iTCP -n
```

Odnosno za ***UDP*** transportni protokol pokrenimo naredbu `lsof` na sljedeći način:
```
lsof -iUDP -n
```

**Filtriranje prema portu**
Moguće je raditi filtriranje i prema portu. Probajmo izlistati sve konekcije za koje se koristi port `22` (SSH [`:ssh`]):
```
lsof -i :22 -n

COMMAND   PID USER   FD   TYPE     DEVICE SIZE/OFF NODE NAME
sshd    13414 root   3r  IPv4 1696814439      0t0  TCP 192.168.100.1:ssh->93.137.225.169:60010 (ESTABLISHED)
sshd    22746 root   3u  IPv4  893714238      0t0  TCP *:ssh (LISTEN)
sshd    22746 root   4u  IPv6  893714240      0t0  TCP *:ssh (LISTEN
```

**Filtriranje prema odredišnom računalu**
Sada ćemo filtrirati samo sve otvorene konekcije prema udaljenom računalu s IP adresom: `93.137.225.169` što je naše udaljeno računalo s kojega smo se spojili na poslužitelj.
Ako ne želimo da se prikazuje ime protokola već broj porta možemo dodati `-P`
```
lsof -i@93.137.225.169 -n

COMMAND   PID USER   FD   TYPE     DEVICE SIZE/OFF NODE NAME
sshd    13414 root   3r  IPv4 1696814439      0t0  TCP 192.168.100.1:ssh->93.137.225.169:60010 (ESTABLISHED)
```

**Filtriranje prema stanju konekcije**
Moguće je filtrirati izlistanje stanja konekcije, prema definiranim stanjima.
- `ESTABLISHED`
- `LISTEN`

Uz dodatna stanja:
- `SYN_SENT` i `SYN_RECEIVED`
- `FIN_WAIT_1` i `FIN_WAIT_2`
- `TIME_WAIT` i `CLOSE_WAIT`
- `LAST_ACK` i `CLOSING`



Prisjetimo se stanja TCP veza, vidljivo u:
**24.2.16. Stanja TCP veze i njena vremenska ograničenja *(timeri)*.**

Sada pronađimo sve ***TCP*** konekcije koje su u stanju `ESTABLISHED` dakle ostvarene odnosno trenutno u upotrebi:
```
lsof -i -sTCP:ESTABLISHED -n

COMMAND    PID   USER  FD  TYPE     DEVICE SIZE/OFF NODE NAME
rsyslogd   485   root  2u  IPv4  420783045     0t0  TCP 192.168.100.1:49743->192.168.100.100:514  (ESTABLISHED)
sshd     13414   root  3r  IPv4 1696814439     0t0  TCP 192.168.100.1:ssh->93.137.225.169:60010  (ESTABLISHED)
```



Dodatno pogledajte primjere upotrebe naredbe `lsof` u poglavljima:
**4.5.5. File deskriptori detaljnije.**
**9.4.1. Unix Pipes.**
**9.4.3. Slijedeći koraci: *Unix* i *network socketi*.**
**12.3.4.1 Transparent Huge Pages.**

**Izvori informacija: (222),(223),(224), `man lsof`.**

### 25.7.7. Naredba *socket statistics* (*ss*)

Naredba `ss` pripada novoj generaciji alata iz `iproute2` paketa te je nasljednik naredbe `netstat`. Ona radi na transportnoj razini (OSI razina odnosno sloj **4**) pa prema tome prikazuje detalje o *socketima* i njihovom stanju, uključujući TCP, UDP ili Unix *sockete*. Naredba `netstat` pretražuje `/proc/` direktorij u kojem se nalaze datoteke u kojima se nalaze razni kernel parametri mreže. Stoga u slučajevima kada imamo veliki broj konekcija ili mrežnih parametara, rad naredbe `netstat` zna biti prilično usporen. S druge strane naredba `ss` informacije izvlači direktno iz kernela, što ju čini puno bržom u radu. Dodatno, moguće je koristiti i napredne filtere ugrađene u sâmu naredbu. Najčešći prekidači naredbe `ss` su:

| Naredba i prekidač | Opis |
|---|---|
| `ss -a` | Ispiši sve statistike. |
| `ss -n` | Interpretiraj sve protokole brojčano bez prepoznavanja imena protokola. |
| `ss -s` | Ispiši listu svih otvorenih socketa. |
| `ss -l` | Ispiši listu svih otvorenih portova. |
| `ss -p` | Ispiši i procese (programe/servise) koji su otvorili portove/sockete. |
| `ss -t` | Ispiši samo TCP konekcije. |
| `ss -u` | Ispiši samo UDP konekcije. |
| `ss -i` | Ispiši interne i detaljnije informacije o TCP mrežnom stogu za pojedini *socket.* |
| `ss -m` | Ispiši statistike memorijskih međuspremnika vezanih za socket/komunikaciju. |

**Slijede primjeri**.

**1.**Prikažimo sve (`-a`) mrežne konekcije u brojčanom formatu IP adrese i broja porta (`-n`) upotrebom naredbe `ss`:

```
ss -an

Netid  State      Recv-Q Send-Q   Local Address:Port          Peer Address:Port
tcp    LISTEN     0      5        192.168.1.254:443               *:*
tcp    LISTEN     0      5        192.168.1.254:80                *:*
tcp    LISTEN     0      5        192.168.1.254:5357              *:*
tcp    LISTEN     0      128      *:37295                         *:*
tcp    LISTEN     0      128      *:111                           *:*
```

**2.** Ispišimo listu (`-l`) svih otvorenih portova, upotrebom *TCP* protokola (`-t`) ali brojčano (`-n`):

```
ss -tln

State        Recv-Q Send-Q        Local Address:Port          Peer Address:Port
LISTEN       0      128             *:22                           *:*
LISTEN       0      128           127.0.0.1:5080                   *:*
LISTEN       0      128             *:443                          *:*
LISTEN       0      128             :::22                          :::*
LISTEN       0      128             ::1:5080                       ::*
LISTEN       0      128             :::443                         :::*
```

**3**. Ispišimo sve *TCP* otvorene portove, numerički i to s procesima (`-p`) koji su ih otvorili:

```
ss -tlnp

State    Recv-Q Send-Q  Local Address:Port  Peer    Address:Port
LISTEN   0      128          *:22            *:*
LISTEN   0      128       127.0.0.1:5080     *:*     users:((„ssh‟,pid=21314,fd=5))
LISTEN   0      128          *:443           *:*
LISTEN   0      128          :::22           :::*
LISTEN   0      128          ::1:5080        :::*    users:((„ssh",pid=21314,fd=4))
LISTEN   0      128          :::443          :::*
```

U zadnjem stupcu desno, vidimo imena procesa (servisa) s njihovim pripadajućim ***PID*** brojevima (`pid=`).

**4.** Ispišimo samo IPv4 (`-t4`) za ***TCP*** protokol i to za sve ostvarene konekcije (engl. ***Established***):

```
ss -t4 state established

Recv-Q Send-Q         Local Address:Port          Peer Address:Port
0      0              192.168.1.254:https         194.138.12.171:54566
0      0              192.168.1.254:1023          192.168.100.11:24007
0      0              192.168.1.254:surf          192.168.100.11:49152
0      0              192.168.1.254:exp1          20.20.10.30:49152
0      0              192.168.1.254:38374         192.168.100.110:ssh
0      232            192.168.1.254:https         10.10.10.10:34155
0      0              192.168.1.254:1015          20.20.10.20:49152
0      0              192.168.1.254:https         10.10.10.10:58294
```

Napomena: Važno je razumjeti kako stanje svake veze (*konekcije*) može biti neko od sljedećih:

| Stanje | Opis |
|---|---|
| `established` | Ostvarene konekcije. |
| `syn-sent` | TCP syn paket je poslan. |
| `syn-recv` | TCP syn paket je upravo zaprimljen. |
| `fin-wait-1` | TCP je u stanju zatvaranja i čeka se FIN ACK paket s druge strane. |
| `fin-wait-2` | TCP Socket je zatvoren, ali se čeka potvrda FIN ACK s druge strane. |
| `time-wait` | TCP FIN *Timeout* još nije istekao te je veza u stanju čekanja da se zatvori. |
| `closed` | Konekcija/veza je zatvorena. |
| `close-wait` | Program koji je otvorio TCP *socket* je još aktivan i nije još zatvorio TCP socket. |
| `last-ack` | Stanje, trenutak prije urednog zatvaranja TCP konekcije. |
| `closing` | Konekcija/veza je u stanju zatvaranja. |
| `all` | Označava sva stanja od gore. |
| `connected` | Konekcija je ostvarena. |
| `synchronized` | Označava sva stanja osim `syn-sent`. |
| `bucket` | Stanja, koja se smatraju `minisocket`, poput `time-wait` i `syn-recv`. |
| `big` | Suprotno od `bucket`. |

**5.** Ispišimo sve ***TCP*** ostvarene konekcije na port **443** (*HTTPS* protokol) i to filtrirajući i izvorišni (*source)* port **443** i odredišni (*destination)* port **443** brojčano:

```
ss -atn -o state established '( dport = :443 or sport = :443 )'

Recv-Q Send-Q          Local Address:Port    Peer Address:Port
0      0               192.168.1.254:443     20.20.20.20:54566  timer:(keepalive,47min,0)
0      232             192.168.1.254:443     10.10.10.10:34155  timer:(on,064ms,0)
0      0               192.168.1.254:443     10.10.10.10:58294  timer:(keepalive,47min,0)
```

**6.** Ispišimo vrlo detaljne informacije o ***TCP*** sloju mreže, za ***socket*** komunikaciju na lokalnom *TCP* portu broj **80** (*source port*).

```
ss --info  sport = :80 -n

Netid  State    Recv-Q Send-Q    Local Address:Port     Peer Address:Port
tcp    ESTAB    0      0         192.168.1.254:80       192.168.1.103:7992
cubic wscale:8,7 rto:204 rtt:0.256/0.045 ato:40 mss:1260 cwnd:21 ssthresh:20
bytes_acked:246069 bytes_received:26474 segs_out:221 segs_in:148 data_segs_out:220
data_segs_in:44 send 826.9Mbps lastsnd:2192 lastrcv:2192 lastack:2192 pacing_rate
989.8Mbps rcv_space:29200 minrtt:0.102
```

## Prvo ćemo malo detaljnije objasniti što se događa u mrežnoj komunikaciji, vezano za *network sockete*.


Međutim prvo se podsjetite poglavlja: **9.4.3.2. Network *socketi*.**

Kako smo vidjeli u poglavljima o komunikaciji među procesima (**9.4. Komunikacija između procesa**) lokalni procesi mogu među sobom komunicirati pomoću *Unix socketa*, slanjem ili primanjem podataka na konkretni *Unix socket* koji predstavlja *file deskriptor* (br. *XY)*. U slučaju kada procesi (aplikacije) žele komunicirati preko mreže, oni moraju koristiti mrežne *sockete* (Engl. *Network socket*) na isti način. Međutim, ako se spustimo na nižu razinu rada, ovdje stvari postaju malo kompliciranije.
Naime u inicijalnom trenutku kada naš proces (aplikacija) želi komunicirati preko mreže s nekim mrežnim servisom, primjerice s web (*HTTP*) poslužiteljem na IP adresi 10.10.10.10 na portu 80, ona još nema *mrežni socket* prema toj udaljenoj točki, koju definiraju IP adresa i njen pripadajući port (pr. 10.10.10.10:80). Zato što takav *mrežni socket* postoji samo lokalno na računalu tog udaljenog (*HTTP*) poslužitelja. U tom trenutku na našem računalu (IP: 20.20.20.20) će sustav (Linux) kreirati novi *mrežni socket* za tu vezu i dodijeliti mu identifikator; primjerice `347036604` te ga povezati s *file deskriptorom* primjerice ***FD*** broj `17`. Taj *file deskriptor* će se potom koristiti za komunikaciju s tom krajnjom točkom (10.10.10.10:80).
Dakle taj konkretni *file deskriptor* (pr.`17`) postaje veza naše aplikacije prema konkretnom mrežnom (*network) socketu*.

Ne zaboravite da je identifikacijski broj *file deskriptora* jedinstven sâmo za točno određeni proces (program) prema njegovom ***PID*** broju. Tako bi recimo za našu lokalnu aplikaciju koja ima ***PID*** broj `1230` to bio file deskriptor broj `17` vidljiv kao datoteka: `/proc/1230/fd/17`.

Tek sada, naša aplikacija (pr. web preglednik) koji ima primjerice ***PID*** broj `1230` može i koristiti naš novi *mrežni socket*, broj `347036604`, ali indirektno, s upotrebom *file deskriptora* broj `17` unutar svog ***PID*** prostora.
S druge strane, kada se mi kao klijent (web preglednik) prvi puta spojimo na web poslužitelj, na web poslužitelju se također za ovu kao i svaku novu pojedinu konekciju isto kreira po jedan novi *mrežni socket* i povezuje se s određenim *file deskriptorom*.



*Ponovimo: svaki pojedini mrežni socket se identificira s pet parametara:*
***protokol*** *(pr. TCP), te* ***par: izvorišna IP adresa i njen pripadajući port te odredišna IP adresa i njen pripadajući port****.*

U konkretnom slučaju na našem klijentu (web preglednik) imat ćemo sljedeći *mrežni socket*:
*Broj socketa:* `347036604` *identifikatori network socketa: TCP, 20.20.20.20:**63738** → 10.10.10.10:**80***
*Ovaj mrežni socket broj* `347036604` *povezuje se s file deskriptorom (17): /proc/*`1230`*/fd/*`17` *→ socket: [*`347036604`*]*
Na strani web poslužitelja je slično.

A sada možemo proći opis nekih od polja koje ste vidjeli u primjeru broj **6**:

- Na početku imamo dva stupca: `Recv-Q` i `Send-Q` koji označavaju iskorištavanje *socket međumemorije* (*socket buffer*).
- `cubic` - označava TCP algoritam za izbjegavanje zagušenja.
  - Pogledajte poglavlje: **24.2.9. Nadzor zagušenja (*Congestion control*)**.
- `wscale` - označava da se koristi skaliranje prozora; prvi broj je poslani faktor skaliranja, a drugi, onaj primljeni,
  - Pogledajte poglavlje: **24.2.8.2. Skaliranje TCP prozora (*Window Scaling*).**
- `rto` - označava vrijeme TCP retransmisije u milisekundama.
  - Pogledajte poglavlje: **24.2.13. Mehanizam TCP retransmisije.**
- `rtt` - označava prolazno vrijeme TCP paketa (prvi broj) a drugi broj označava srednju devijaciju ***RTT*** vremena.
  - Pogledajte poglavlje: **24.2.7.1. Vrijeme prolaska paketa (*Round-Trip Time*)**.
- `ato` - označava TCP ACK *timeout* vrijeme, kod upotrebe "*Delay ACK*" metode.
  - Pogledajte poglavlje: **24.2.5. .Standardne potvrde (*ACK*).**
- `mss` - označava maksimalnu veličinu TCP segmenta.
  - Pogledajte poglavlje:**24.2.1. Maximum segment size (*MSS*).**
- `cwnd` - označava veličinu TCP prozora zagušenja („*Congestion window*").
  - Pogledajte poglavlje: **24.2.9.1. Usporeni početak i izbjegavanje zagušenja.**
- `ssthresh` - označava TCP prag veličine prozora zagušenja.
  - Pogledajte poglavlje: **24.2.9.1. Usporeni početak i izbjegavanje zagušenja.**
- `bytes_acked` - označava koliko bajta podataka u prijenosu je potvrđeno (TCP ACK).
- `bytes_received` - označava koliko bajta podataka je zaprimljeno.
- `segs_out` - označava koliko je mrežnih segmenata poslano.
- `segs_in` - označava koliko je mrežnih segmenata primljeno.
- `send` - označava koliko je podataka poslano (na ovaj mrežni *socket)****.
- `lastsnd` - označava koliko je prošlo vremena u milisekundama, od kada je zadnji paket poslan.
- `lastrcv` - označava koliko je prošlo vremena u milisekundama, od kada je zadnji paket zaprimljen.
- `lastack` - označava koliko je prošlo vremena u milisekundama, od kada je zadnji ACK paket zaprimljen.
- `pacing_rate` - prva vrijednost označava „*pacing rate*" a druga „*max pacing rate*". Ona označava takozvani „***Pacing rate***" koji se odnosi na (novu) funkcionalnost TCP stôga[779] koja donosi finu kontrolu u slanju podataka prema nižim slojevima mreže. Statistika i funkcionalnost za ***TCP pacing*** ovisi o tome je li on uopće podržan i uključen u kernelu. I ove vrijednosti se odnose na statistiku o tome koliko podataka je proslijeđeno ovom mrežnom *socketu*, a ne nužno i koliko je stvarno i završilo na mreži*****.
- `rcv_space` - ovo je interna varijabla za automatsko podešavanje TCP komunikacije vezana za prijemnu *socket* memoriju (Engl. *Socket receive buffer*).

Opis svih ovdje vidljivih polja pogledajte na stranici: http://man7.org/linux/man-pages/man8/ss.8.html

Osim što smo se u ovom primjeru (broj **6.**) spojili na mrežni *socket* vezan za servis koji je lokalno otvorio TCP port 80, mogli smo se spojiti i na sve mrežne *sockete* koji su recimo otvoreni prema udaljenoj (*destination*) IP adresi.

Primjerice za odredišnu (*DST*) IP adresu:10.10.10.10 bi to napravili na sljedeći način:

```
ss --info dst 10.10.10.10
```

**Izvor informacija: (777),(778),(779),** `man ss`,`man 7 ip`,`man 7 tcp`,`man 7 socket`.

### 25.7.8. Naredba *tcpdump*

Naredba odnosno program `tcpdump` koristi se za analizu mrežnih paketa, a pokreće se u naredbenom retku. Ona nam omogućuje prikaz TCP/IP i drugih vrsta mrežnih paketa koji se prenose ili primaju preko mreže na koju je priključeno naše računalo. Naime postoje slučajevi u kojima zbog potrebe raščlanjivanja rada odnosno komunikacije između programa na mreži imamo potrebu snimiti mrežnu komunikaciju to jest pakete na mreži, kako bismo mogli vidjeti što se stvarno događa u komunikaciji. Drugi primjer upotrebe ovakvog alata je analiza mrežnog prometa zbog nekih drugih grešaka koje se događaju na mreži ili zbog otkrivanja grešaka u konfiguraciji mreže ili mrežnih servisa (*daemona*) pod Linuxom, kao i zbog potencijalne loše konfiguracije mrežne opreme ili uređaja. ***Tcpdump*** radi na većini operativnih sustava sličnih *Unixu*: ***Linux***, ***Solaris***, ***FreeBSD***, ***DragonFly BSD***, ***NetBSD***, ***OpenBSD***, ***OpenWrt***, ***MacOS***, ***HP-UX*** i ***AIX***. U tim operativnim sustavima *tcpdump* koristi biblioteku ***libpcap***, preko ***pcap*** API-ja, za dohvaćanje paketa na mreži. Pošto se dobar dio tih operativnih sustava nalazi i u raznim mrežnim uređajima (usmjerivači, preklopnici, *load balanceri* i sl.) to znači kako vrlo često i na njima možemo naći ovu naredbu. *Tcpdump* može čitati pakete s mrežne kartice ili iz prethodno stvorene spremljene paketne datoteke; a koja obično dolazi u formatu ***.pcap*** (Engl. *Packet Capture*). On standardno ispisuje sadržaj mrežnih paketa koje pratimo, na standardni izlaz („ekran“) ili ih može zapisivati u datoteku. Osim programa *tcpdump* često je u upotrebi i znatno veći, prošireniji i kompleksniji program sa znatno većim nîzom mogućnosti, a koji radi u grafičkom sučelju i zove se ***Wireshark***. O njemu nećemo ovdje govoriti jer nadilazi naše potrebe. Mi ćemo se fokusirati na osnovne mogućnosti odnosno potrebu da možemo snimiti i trenutno ili kasnije analizirati mrežne pakete koji dolaze na našu mrežnu karticu ili odlaze s nje, sve s pomoću programa `tcpdump`.
Program `tcpdump` obično dolazi već instaliran, a ako to nije slučaj, možete ga instalirati sa sljedećom naredbom:

```
yum -y install tcpdump
```

Krenimo s popisom samo nekih od osnovnih prekidača programa `tcpdump`:

- `-c` **X** - (malo slovo c) označava da ćemo snimiti samo **X** (broj) paketa te automatski izaći iz programa.
- `-C` **X** - (veliko slovo c) označava da ćemo snimati samo **X** (MB) paketa te automatski izaći iz programa.
- `-D` - označava kako želimo samo ispisati mrežna sučelja/kartice na kojima možemo snimati pakete.
- `-e` - označava kako želimo prikaz i *Ethernet* dijela zaglavlja mrežnih paketa.
- `-i` ***interface*** - označava da ćemo snimati mrežne pakete na mrežnom sučelju imena; ***interface****:* `eth0`, `bond0`, …
- `-K` - izričito isključuje provjeru ***IP***, ***TCP*** ili ***UDP*** provjernog zbroja (*checksum*) što je korisno samo, ako imamo mrežne kartice koje *hardverski* rade ovaj izračun. Naime tada vrijednosti koje ćemo vidjeti ovdje neće biti ispravne i bit će označene kao da imamo grešku. Za opis rada ovakvih mrežnih kartica pogledajte poglavlje: **25.5.3**.
- `-l` - u slučaju kada ispis dodatno filtriramo s nekom drugom naredbom poput naredbe `grep` ili slične, paketi se prvo spremaju u međumemoriju, a potom prosljeđuju na naredbu koja slijedi (upotrebom *pipe* funkcionalnosti).
- `-n` - označava da ne želimo da sustav konvertira adrese u imena; pr. IP-*hostname* ili broj porta-servis (22→SSH).
- `-N` - označava da ne želimo da se uz ime računala dodaje i ime domene.
- `-s` - naznačavamo sustavu maks. veličinu paketa koju ćemo prikazati/snimiti. `-s0` znači da dohvaćamo sve.
- `-v` - označava da želimo detaljniji ispis (*verbose*), možemo koristiti i `-vv` (za još detaljniji) ili čak `-vvv` za posebno detaljan ispis.
- `-t` - označava kako ne želimo ispis vremenske oznake paketa. Međutim možemo koristiti i:
  - `-tt` - označava da želimo prikaz vremena u sekundama od 1.1.1970 (*UNIX vrijeme*).
  - `-ttt` - prikaz razlike vremena (***delta*** *u mikro sekundama*) između trenutne linije i one prije nje.
  - `-tttt` - označava da želimo prikaz vremena u minutama, sekundama i frakcijama sekundi od ponoći.
- `-w` *FILE* -označava kako želimo ispis snimljenih mrežnih paketa zapisivati u datoteku imena *FILE* (**.pcap**).
- `-X` - označava da kod ispisa želimo detaljan ispis paketa (i heksadecimalni), ali bez podataka s OSI 2 sloja.

Za ***TCP*** transportni protokol vidjet ćemo i zastavice, koje ćemo opisati u tablici:

| Oznaka u *tcpdump*-u | Vrsta zastavice prema TCP protokolu | Opis zastavice (značenje) |
|---|---|---|
| **S** | ***TCP SYN*** | TCP Synchronize – ovo je prvi paket za uspostavljanje TCP veze. |
| **F** | ***TCP FIN*** | TCP Finish – ovo je paket za zatvaranje TCP veze. |
| **P** | ***TCP PUSH*** | TCP Push – „*Data push*“ – slanje podataka direktno prema aplikacijskom sloju. |
| **R** | ***TCP RST*** | TCP Reset – za resetiranje TCP veze. |
| **.** | ***TCP ACK*** | TCP Acknowledge – za potvrđivanje paketa. |

Idemo na primjere kroz koje ćemo naučiti nešto više. Provjerimo na kojim sve mrežni sučeljima (i mrežnim karticama) na našem računalu uopće možemo snimati mrežne pakete, upotrebom naredbe `tcpdump` na sljedeći način:

```
tcpdump -D

1.enp7s0 [Up, Running]
2.eth0 [Up, Running]
3.bond0 [Up, Running]
4.bond0.10 [Up, Running]
5.any (Pseudo-device that captures on all interfaces) [Up, Running]
6.lo [Up, Running, Loopback]
7.nflog (Linux netfilter log (NFLOG) interface)
8.nfqueue (Linux netfilter queue (NFQUEUE) interface)
```

Ovdje vidimo razna mrežna sučelja, poput mrežnih kartica: `enp7s0`, `eth0` te agregiranih sučelja: `bond0` te agregiranog *VLAN* sučelja: `bond0.10`, kao i *loopback* sučelja: `lo`. Vidimo i kako sva mrežna sučelja definira posebno sučelje imena: `any` koje možemo koristiti, ako želimo snimati pakete sa apsolutno svih dostupnih mrežnih sučelja na sustavu.

1. **Prikupimo sve mrežne pakete na svim dostupnim mrežnim sučeljima** (i ispisujmo ih na terminal/*ekran*):

```
tcpdump -i any
```



Kada budemo analizirali ovakve pakete (`-i any`), primjerice s programom ***Wireshark***, na OSI sloju dva (***Ethernet***) je moguće da cijeli ***Ethernet*** sloj bude označen kao: **`"linux cooked capture"`** te da vidimo duplicirane pakete.

I dobit ćemo konstantan ispis uhvaćenih mrežnih paketa poput ovoga dolje (skratili smo ispis na samo nekoliko paketa):

```
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
20:14:49.773953 IP server1.local.hr.ssh > hh-pc1.3247: Flags [P.], seq
2650730775:2650730983, ack 2223629491, win 379, length 208
20:14:49.774223 IP hh-pc1.3247 > server1.local.hr.ssh: Flags [.], ack 208, win 2083,
length 0
20:14:49.774664 IP hh-pc1.3247 > server1.local.hr.ssh: Flags [P.], seq 1:161, ack
208, win 2083, length 160

^C
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```



Hvatanje mrežnih paketa uvijek zaustavljamo s kombinacijom tipki ***CTRL C*** (vidljivo u ispisu kao **`^C`**).

Gore vidimo cijeli nîz mrežnih paketa koje smo uhvatili na mreži na svim spojenim mrežnim sučeljima i mrežnim karticama. Pogledajmo ponovno jedan od snimljenih mrežnih paketa na kojem ćemo objasniti svaki njegov dio:

```
20:14:49.774223 IP hh-pc1.3247 > server1.local.hr.ssh:  Flags [.], ack 208, win 2083,  length 0
   |(1)          |(2) |(3)   |(4)    |(5)     |(6)    |(7)        |(8)       |(9)    |(10)      |(11)
```

Opisat ćemo polja unutar svakog mrežnog paketa, kako je vidljivo iznad, prema njegovim dijelovima, malo detaljnije:

- (1) - vrijeme je dospijeća paketa: 20.h. 14.min i 49 sekundi, te `774223` mikro sekundi.
- (2) - ovo je protokol (`IP` u ovom slučaju), što je uobičajeno jer radimo sa TCP/IP mrežama.
- (3) - ovo je ime računala (*hostname*) koje šalje mrežni paket; konkretno: `hh-pc1`.
- (4) - ovo je izvorišni TCP (ili UDP) broj porta, on je ovdje konkretno `3247`.
- (5) - ovo je odredišna (*destination*) strana koja prima odnosno ime tog računala (*hostname*); konkretno: `server1`.
    - (6) - ime domene odredišnog računala; konkretno: `local.hr`.
- (7) - ovo je odredišni (*destination*) port - konkretno označen kao SSH (`.ssh`) dakle prima se paket na SSH servis.
- (8) - ovo su TCP zastavice ako se radi o TCP protokolu; konkretno imamo zastavicu `[.]` što znači kako je to TCP ACK poruka/paket.
- (9) - TCP- konkretno: potvrđuje mrežni okvir `208` s porukom *TCP ACK*.
- (10) - TCP- konkretno: postavlja veličinu TCP prozora (*TCP Window size*) na `2083` konkretno.
- (11) - TCP- konkretno: kaže kako je duljina TCP segmenta konkretno nula (`0`).



Važno je razumjeti kako svaki mrežni paket moramo znati analizirati, poznavanjem TCP/IP protokola, kao i aplikacijskih protokola (***DNS***, ***HTTP***, ***IMAP***, ***LDAP***, …) u slučajevima kada radimo analizu paketa na aplikacijskoj razini.



Ako želimo snimati lokalni promet (unutar računala), on prolazi samo kroz *loopback* mrežno sučelje (`lo`), a ne kroz fizičko ili logičko (pr. `eth0`) . Stoga kao mrežno sučelje koristite `lo`. Primjerice ovako:

Dakle ovo bi bio najbolji način za snimanje (ili praćenje) mrežnog prometa koji ne izlazi iz vašeg računala na mrežu:

```
tcpdump -i lo -s0
```

2. **U ovom primjeru ćemo odabrati hvatanje mrežnih paketa sâmo na mrežnom sučelju `eth0` pomoću prekidača `-i` i to samo 5 paketa (`-c5`) uz naređenje da se ne razlučuju IP adrese u imena računala niti brojevi portova u imena mrežnih servisa, što dobivamo pomoću (`-nn`).**

```
tcpdump -i eth0 -c 5 -nn
```

Dakle ispisat će se sâmo 5 uhvaćenih mrežnih paketa na sučelju `eth0` koje nećemo ovdje sve prikazati (već sâmo njih par):

```
20:34:06.000495 IP 192.168.1.254.22 > 192.168.1.8.3247: Flags [P.], seq 720:880, ack 1, win 397, length 160
20:34:06.000585 IP 192.168.1.8.3247 > 192.168.1.254.22: Flags [.], ack 208, win 2086, length 0
```

Sada vidimo kako se više ne razlučuju imena računala, pa imamo njihove (za prvi paket):

- (***SRC***) izvorišnu IP adresu: `192.168.1.254` i njen broj porta; source port `22`, što znači kako se radi o SSH protokolu.
- (***DST***) odredišnu IP adresu: `192.168.1.8`) i njen broj odredišnog porta; *destination* port `3247`.
- U ostalim dijelovima se ne radi razlučivanje imena te vidimo sve isto kao u primjeru koji smo objasnili prethodno.

**Dalje ćemo nastaviti s primjerima, ali bez prikaza ispisa naredbe.**
Moguće je filtrirati pakete i prema broju porta odnosno sâmim time protokolu koji želimo filtrirati.

**3. Dohvatimo mrežne pakete kao u primjeru broj 2, ali namijenjene samo za port 80 (*http*):**

```
tcpdump -i eth0 -c 5 -nn port 80
```

Moguće je filtrirati pakete i prema odredišnoj adresi (engl. *destination*) [`dst`] ili izvorišnoj adresi (engl. *source*) [`src`].

**4. Sada ćemo koristiti isto što i u primjeru 2, ali ćemo filtrirati samo pakete koji dolaze s izvorišne adrese: 192.168.1.8:**

```
tcpdump -i eth0 -c 5 -nn src 192.168.1.8
```

Moguće je koristiti i logičke operatore:

- `and` - logičku operaciju **I**.
- `or` - logičku operaciju **ILI**.
- `not` - logičku operaciju **NE**.

**5. Sada kombinirajmo filtere iz promjera 3. i 4.**
**Dakle filtrirajmo samo port 80 (http) i (*logičko I*) samo pakete sa izvorišne IP adrese 192.168.1.8:**

```
tcpdump -i eth0 -c 5 -nn src 192.168.1.8 and port 80
```

To znači kako će se filtrirati i prikazati sâmo paketi koji dolaze s IP adrese 192.168.1.8 kojima je odredišni ili izvorišni port 80.
Moguće je gledati i sadržaj paketa kao *ASCII* kôd (tekst) što je osobito korisno za HTTP (port 80) promet.

**6. Ponovit ćemo primjer 3, za filtriranje prometa samo za port 80, ali ćemo dodati prekidač `-A`, kako bi u ispisu dobili i sadržaj paketa, što je kako smo rekli korisno za HTTP pakete odnosno HTTP promet:**

```
tcpdump -i eth0 -c 5 -nn port 80 -A
```

I dobit ćemo sadržaj *HTTP* paketa, u komunikaciji koju pratimo (na portu 80).
Nadalje, moguće je filtrirati pakete koje prikazujemo i prema protokolu i to na dva načina:

- Imenom protokola:
  - `tcp` - filter samo za TCP promet.
  - `udp` - filter samo za UDP promet.
  - `ether` - filter samo *ethernet* promet.
  - `arp` - filter samo ARP protokol.
  - `rarp` - filter samo RARP prookol.
  - `ip` - filter samo IP (v4) ili `ip6` - samo IP (v6),
  - `icmp` – za ICMP protokol i tako dalje.
- Brojčanom oznakom protokola:
  - `proto X` - pri čemu je `X` broj protokola, kako je definirano prema **IANA** organizaciji (standardu): https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

Moguće je koristiti i ključne riječi poput:

- `host` - za definiranje samo IP adrese od ili s koje želimo gledati pakete; primjerice: `host 192.168.1.1`.
- `net` - za definiranje cijele mreže od ili s koje želimo gledati pakete; primjerice: `net 192.168.1.0/24`.
- `portrange` *X-Y* - za definiranje opsega portova; primjerice: `portrange 20-22`.

Naravno, moguće je kombinirati sve do sada navedene filtere, ključne riječi i logičke operatore.

**7. Filtrirajmo samo sav `UDP` promet na mrežnoj kartici `eth0` za pet paketa:**

```
tcpdump -i eth0 -c 5 -nn udp
```

Moguće je koristiti i napredne filtere unutar transportnog protokola, primjerice za TCP možemo filtrirati pakete koji sadrže sâmo određene zastavice.
Pogledajmo brzinske primjere za filtriranje na mrežnom sučelju `eth0`, prema TCP zastavicama (vrstama TCP poruka):

- `tcpdump -i eth0 'tcp[tcpflags] == tcp-syn'` - sâmo ***TCP SYN*** poruke.
- `tcpdump -i eth0 'tcp[tcpflags] == tcp-rst'` - sâmo ***TCP RST*** (reset) poruke.
- `tcpdump -i eth0 'tcp[tcpflags] == tcp-fin'` - sâmo ***TCP FIN*** poruke.
- `tcpdump -i eth0 'tcp[tcpflags] == tcp-urg'` - sâmo ***TCP URG*** (urgent) poruke.
- `tcpdump -i eth0 'tcp[tcpflags] == tcp-ack'` - sâmo ***TCP ACK*** poruke.
- `tcpdump -i eth0 'tcp[tcpflags] == tcp-psh'` - sâmo *TCP PSH* (push) poruke.
- `tcpdump -i eth0 'tcp[13]=18'` - sâmo ***TCP SYN*** + ***ACK*** poruke.

Ako imamo potrebu kasnije detaljnije analizirati mrežni promet odnosno pakete koje pratimo, sve je moguće i snimiti u datoteku.

**8. U ovom primjeru nećemo ograničavati broj paketa, već ćemo nakon nekog vremena prekinuti snimanje sa *CTRL C*.**
**Sve ćemo snimati u datoteku imena: `promet.pcap`.**
Za potrebe snimanja u datoteku koristimo kao prekidač malo slovo `-w`

```
tcpdump -i eth0 -nn -w promet.pcap
```

I nakon željenog vremena stišćemo: *CTRL C* te prekidamo snimanje.
Sada smo dobili datoteku imena `promet.pcap` koju možemo primjerice prekopirati na drugo računalo i analizirati (otvoriti) i s programom ***Wireshark***.
U slučajevima kada želimo pokrenuti snimanje paketa s mreže u datoteku zbog kasnije analize možemo ograničiti i veličinu datoteke u koju snimamo i uz broj datoteka koliko će ih se rotacijski snimati.

**8.1 Snimat ćemo promet kao u primjeru 8, ali ćemo ograničiti snimanje na datoteke veličine 10MB, pomoću prekidača:** `-C` (veličina je u MB)**. Broj rotacijskih datoteka smo postavili na 2:**

```
tcpdump -i eth0 -nn -w promet.pcap -C10 -W2
```

To znači kako će se sav navedeni promet snimati u prvu datoteku veličine 10 MB, pa u drugu do 10MB, a potom ponovno u prvu dok se ne popuni do 10MB, pa drugu i tako stalno u krug.
Na kraju sve moramo prekinuti s kombinacijom tipki *CTRL C*.



U slučaju da nismo koristili **`-W2`** paketi bi se snimali u beskonačno veliki broj datoteka veličine 10MB i s vremenom zagušili cijeli sustav odnosno računalo, prepunjivanjem diska.



U računalnim mrežama, ***pcap*** je sučelje aplikacijskog programskog sučelja (**API**) za hvatanje mrežnog prometa. Operativni sustavi *Unix i Linux* implementiraju **pcap** u biblioteku ***libpcap.*** Dok za starije inačice *Windowsa* postoji takozvana ***libpcap*** implementacija ove funkcionalnosti, pod imenom ***WinPcap***. Međutim novije inačice *Windowsa* koriste implementaciju pod imenom ***Npcap***. Dakle softver koji ima potrebu za dohvaćanje mrežnih paketa koji putuju preko računalne mreže, može koristiti ***libpcap***, ***WinPcap*** ili ***Npcap*** ovisno iz kojeg operativnog sustava se to radi.

Međutim, kod potreba za ekstremnim brzinama dohvaćanja mrežnih paketa (10+Gbps), dohvaćanje mrežnih paketa pomoću ***pcap*** **API**-ja dolazi do svojih ograničenja. Rješenje je upotreba **XDP** (Engl. *Express Data Path*) i **eBPF** (Engl. *Extended Berkeley Packet Filter*) mehanizama za dohvaćanje mrežnih paketa.



O ovim mehanizmima pročitajte više u poglavlju:
**26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP + eBPF*).**



**XDP+eBPF** daju programima mogućnost da rade stvari koje nisu moguće u korisničkom prostoru aplikacija: poput izravnog pristupa strukturama (podacima) koje koriste upravljački programi (pr. za mrežnu karticu). Odnosno praktično nam daju direktni pristup mrežnoj kartici i dohvaćanju mrežnih paketa s nje. Time se izbjegava korištenje **pcap API**-ja za dohvaćanje mrežnog prometa (mrežnih paketa) koji kod vrlo brzih mreža znatno opterećuje CPU.

Pogledajmo još jedan napredniji filter s kojim ćemo filtrirati sav *ICMP* promet, koji unutar **IP** zaglavlja u **ToS** polju ima postavljenu vrijednost **`0x02`** odnosno decimalno **`2`** to možemo postići sa:

```
tcpdump -i eth0 -n -vvv 'icmp and ip[1] & 0x02 == 2'
```

Pogledajmo i kako dohvatiti pakete koji u IP zaglavlju (***Flags***) imaju postavljen **DF** (*Dont fragment*) bit:

```
tcpdump -n -vv -i eth0 'ip[6] = 64'
```

U ispisu ćemo uz svaki ovakav paket vidjeti i poruke poput (**`flags [DF]`**). Za detaljniji opis, pogledajte poglavlje (23.2.).



Pogledajte i druge primjere upotrebe naredbe **`tcpdump`** u sljedećim poglavljima:
**20.6.7. Link Layer Discovery Protocol (LLDP).**
**23.1.1. Osiguranje kvalitete (*QoS*) i klasifikacija prometa (*ToS i DSCP*).**
**23.2. IP fragmentacija.**
**23.4.1. Time to live (*TTL*).**
**25.7.1.1. ICMP poruka.**
**25.8.8.1. Konfiguracija i rad s elektroničkom poštom.**

**Izvori informacija:(645),(646),(647),(648),(649),(650),(651),(1021),(1134),(1135),(1366),** **`man tcpdump`**, **`man 7 ip`**, **`man 7 tcp`**, **`man 7 socket`**.

## 25.7.9. Naredba ip

Naredbu `ip` smo već upoznali u prethodnim cjelinama i poglavljima, ali ćemo probati kratko navesti sve njene mogućnosti. Naime naredba `ip` je nasljednik starije naredbe `ifconfig`, koja je dolazila u paketu ***net-tools*** zajedno s naredbama: `netstat`, `arp`, `ipmaddr`, `iptunnel`, `route` i drugima, čije funkcionalnosti ona mijenja. Osim toga paket ***net-tools*** s pripadajućim naredbama se u novije vrijeme više ne instalira automatski. Nadalje, naredba `ip` pristupa mrežnom stôgu linuxa na nižoj razini od navedenih starijih naredbi (preko ***NetLink*** sloja), pa samim time radi brže te nudi neke funkcije koje prije nisu bile dostupne. Pogledajmo koje sve funkcionalnosti imamo dostupne unutar ove nove naredbe `ip`, koje su dostupne kao njeni objekti, prema sintaksi: `ip` *opcije* ***objekt*** *naredba*.



Za sve opcije i objekte, moguće je koristiti i skraćeni način rada; primjerice naredba **`ip address show`** se može pozvati i skraćeno, kao: **`ip a s`**.
Nakon što upišemo `ip` možemo stisnuti tipku *TAB*, te će nam se ponuditi opcije, ili nakon njih i objekti koje imamo dostupne. Na taj način ne moramo pamtiti sve opcije objekte ili naredbe koje se nalaze unutar sučelja naredbe `ip`.

Za ispis ***NetLink*** poruka prema kernelu (pr. stanje sučelja *UP/DOWN*), možemo koristiti sljedeću naredbu:

```
ip monitor
```

### 25.7.9.1. Rad s IP parametrima mrežnih sučelja

**Objekt `address`** koristi se za rad s IP parametrima mrežnih sučelja. Dakle pomoću naredbe `ip address` konfiguriraju se IP parametri mrežnih sučelja; bilo da se radi o fizičkim, logičkim ili virtualnim mrežnim sučeljima.
Pogledajmo brze primjere upotrebe naredbe `ip address`.

**Ispišimo sve IP, ali i druge parametre rada svih mrežnih sučelja:**

```
ip address show
1:lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000 link/ether 08:00:27:ba:82:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.129/24 brd 192.168.1.255 scope global dynamic noprefixroute enp0s3
       valid_lft 863999874sec preferred_lft 863999874sec
    inet6 fe80::a00:27ff:feba:8209/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Ovdje vidimo dva mrežna sučelja (`lo` i `enp0s3`) te njihove IP parametre rada, ***MTU*** veličinu, stanje rada (`state UP`), podržavaju li *multicast* (`MULTICAST`) te *Queuing* mehanizam (`qdisc noqueue`) te veličinu predmemorije *[transmit queue]* prema *socketu* (`qlen 1000`). Zatim vidimo njihovu ***MAC*** adresu (pr. `link/ether 08:00:27:ba:82:09`) te **IP** adresu (pr. `inet 192.168.1.129/24`) i *broadcast* **IP** adresu (pr. `brd 192.168.1.255`).

**Moguće je ispisati i samo IP parametre svih mrežnih sučelja u skraćenom obliku:**

```
ip --brief address show
```

**Moguće je ispisati parametre rada svih mrežnih sučelja s opisom o vrsti sučelja (bridge, TUN, TAP, …)** *[ovo je malo poznato]*:

```
ip -d address
```

**Konfigurirajmo ručno IP adresu (`192.168.1.22` s maskom 255.255.255.0 [`/24`]) našem mrežnom sučelju `enp0s3`:**

```
ip address add 192.168.1.22/24 dev enp0s3
```

**Za brisanje IP parametara za mrežno sučelje `enp0s3`, možemo pokrenuti:**

```
ip address del 192.168.1.22/24 dev enp0s3
```

**Odnosno, ako imamo više definiranih IP adresa na istom sučelju i želimo ih sve obrisati to možemo postići sa:**

```
ip address flush dev enp0s3
```



Za dodatne primjere, pogledajte sljedeća poglavlja:
**25.6.1. Konfiguracijska datoteka /etc/hosts**.
**25.6.5.1.1. Privremena statička konfiguracija**.
**25.6.5.2.2. Trajna konfiguracija mreže korištenjem DHCP poslužitelja.**

**Izvor informacija: (1141),(1142),(1143), `man ip-address`, `man ip`, `man 7 netlink`.**

### 25.7.9.2. Rad s parametrima rada mrežnih sučelja

**Objekt `link`** koristi se za rad s parametrima mrežnih sučelja, ali i sâmim sučeljima, poput primjerice: ***bond, bridge, ipip, ipvlan, ipvtap, macvlan, macvtap, team, veth, vlan, vrf, vxlan*** i mnogih drugih. Dakle pomoću naredbe `ip link` konfiguriraju se mrežna sučelja; bilo da se radi o fizičkim, logičkim ili virtualnim mrežnim sučeljima.

**Ispišimo stanja rada svih mrežnih sučelja koja imamo na sustavu:**
```
ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode
DEFAULT group default qlen 1000
    link/ether 08:00:27:ba:82:09 brd ff:ff:ff:ff:ff:ff
```
Ovdje vidimo dva mrežna sučelja (`lo` i `enp0s3`) bez **IP** parametara rada, jer sada gledamo sučelja na nižem OSI sloju, ali vidimo ***MTU*** veličinu, stanje rada (`state UP`), podržavaju li *multicast* (`MULTICAST`) te *Queuing* mehanizam (`qdisc noqueue`) te veličinu predmemorije *[transmit queue]* prema *socketu* (`qlen 1000`). Zatim vidimo njihovu **MAC** adresu (pr. `link/ether 08:00:27:ba:82:09`) i to je sve što želimo vidjeti na ovom nižem sloju mreže.

**Za promjenu MTU vrijednosti na 1400, za mrežno sučelje `enp0s3` možemo pokrenuti:**
```
ip link set mtu 1400 dev enp0s3
```

**Za promjenu `txqueuelen` (*transmit queue*) vrijednosti, primjerice na 2000, za mrežno sučelje `enp0s3`:**
```
ip link set txqueuelen 2000 dev enp0s3
```

**Za isključivanje multicast-a na mrežnom sučelju `enp0s3`, možemo pokrenuti [*za uključivanje koristimo:* on *umjesto* off]:**
```
ip link set enp0s3 multicast off
```

**Za ispis detaljnijih statistika rada mrežnog sučelja `enp0s3` možemo pokrenuti:**
```
ip -s -s link ls enp0s3
```

**Za isključivanje mrežnog sučelja `enp0s3` možemo pokrenuti:**
```
ip link set enp0s3 down
```

**Za ponovno uključivanje mrežnog sučelja `enp0s3` možemo pokrenuti:**
```
ip link set enp0s3 up
```

**Bridge mrežna sučelja i rad s njima**

**Za kreiranje *bridge* vrste mrežnog sučelja imena `br0`, trebamo pokrenuti:**
```
ip link add name br0 type bridge
```

**Za aktivaciju mrežnog sučelja imena `br0`, možemo pokrenuti:**
```
ip link set dev br0 up
```

**Brisanje mrežnog sučelja imena `br0` sa sustava, možemo odraditi sa:**
```
ip link del br0
```

**Za dodavanje fizičkih mrežnih sučelja `eth0` i `eth1` u *bridge* sučelje imena `br0`:**
```
ip link set dev eth0 master br0
ip link set dev eth1 master br0
```
Zatim možemo aktivirati ovo novo mrežno sučelje:
```
ip link set br0 up
```
Potom možemo kreirati IP parametre na tom mrežnom sučelju, sa sljedećom naredbom:
```
ip address add 192.168.1.22/24 dev br0
```

**VLAN mrežna sučelja i rad s njima**

**Za kreiranje *VLAN* vrste mrežnog sučelja s VLAN 10 pripadnosti, na mrežnoj kartici `eth0`, trebamo pokrenuti:**
```
ip link add link eth0 name eth0.10 type vlan id 10
```
Nakon ovog koraka dobit ćemo virtualno mrežno sučelje imena `eth0.10`, koje će biti u VLAN 10 mreži.
Potom možemo kreirati IP parametre na mrežnom sučelju `eth0.10` s naredbom:
```
 ip address add 192.168.1.22/24 dev eth0.10
```



Za dodatne primjere, pogledajte sljedeća poglavlja:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**
**20.6.1.1. Privremena konfiguracija *bridge* mrežnih sučelja.**
**20.6.2. VLANovi odnosno virtualne lokalne mreže.**
**20.6.4. VETH - posebno mrežno sučelje.**

**Izvori informacija:** `man ip-link`, `man ip`, `man 7 netlink`.

### 25.7.9.3. Rad s multicastom

**Objekt `maddress`** koristi se za rad s *multicast* adresama te `mroute` za rad s *multicast* usmjeravanjem (*multicast routing*). Dakle pomoću naredbe `ip maddress` konfiguriraju se *multicast* parametri rada mrežnog sučelja.

**Za ispis pripadnosti *multicast* adresama svih mrežnih sučelja, možemo pokrenuti:**

```
ip maddress show
```



Za dodatne primjere, pogledajte poglavlje:
**22.3.3. *Multicast* u upotrebi.**

**Izvori informacija: `man ip-maddress`,`man ip`, `man 7 netlink`.**

### 25.7.9.4. Mrežni imenični prostori

**Objekt `netns`** koristi se za rad s mrežnim imeničnim prostorima. Dakle pomoću naredbe `ip netns` konfiguriraju se izolirani mrežni prostori Linuxa. Pogledajmo brze primjere upotrebe naredbe `ip netns`. Izolirani mrežni prostor linuxa potpuno je mrežno izoliran od našeg linuxa u kojem smo ga kreirali. Stoga, da bi ga mrežno povezali s našim Linuxom, obično moramo koristiti posebno virtualno mrežno sučelje imena ***VETH*** (*Virtual Ethernet*).
***VETH*** sučelje se kreira u pâru, a ponaša se kao mrežni tunel: sve što uđe u jedno sučelje iz pâra, izlazi kroz dugo.

**Stoga prvo kreirajmo jedan izolirani mrežni linux prostor imena: `PRVI`, na sljedeći način:**

```
ip netns add PRVI
```

**Potom kreirajmo jedan *VETH* pâr mrežnih sučelja i to sučelja sljedećih imena: `veth0` i `veth1`:**

```
ip link add veth0 type veth peer name veth1
```

**Sada ćemo *VETH* sučelje `veth1` povezati s našim izoliranim mrežnim linux prostorom imena: `PRVI`**

```
ip link set veth1 netns PRVI
```

**Moguće je i pokretati naredbe unutar imeničnog mrežnog prostora.**
**Primjerice pokrenimo naredbu: `ip link list`, unutar izoliranog mrežnog prostora imena `PRVI`:**

```
ip netns exec PRVI ip link list
```

Obično se potom sučelje `veth0` povezuje preko novog ***bridge*** sučelja s fizičkom mrežnom karticom računala, kako bi imali komunikaciju između našeg Linuxa i izoliranog mrežnog prostora unutar njega. Ovaj postupak nećemo dodatno objašnjavati.



Za dodatne primjere, pogledajte sljedeća poglavlja:
**20.6.4. VETH - posebno mrežno sučelje.**
**26.8.1. Network *Namespaces*.**

**Izvori informacija: (742),(743),(744), `man ip-netns`, `man ip`,`ip netns help`,`man 7 netlink`.**

### 25.7.9.5. ARP unosi

**Objekt `neighbour`** koristi se za rad s **ARP** i **NDISC** parametrima i unosima. Dakle pomoću naredbe `ip neighbour` radimo s **ARP** ili **NDISC** unosima. Pogledajmo brze primjere upotrebe naredbe `ip neighbour`.
**Ispišimo lokalnu ARP tablicu na našem računalu (skratili smo ju zbog jednostavnijeg prikaza):**

```
ip neighbour show

192.168.1.1 dev enp0s3 lladdr b4:f5:8e:96:f8:50 REACHABLE
192.168.1.112 dev enp0s3 lladdr 40:8d:5c:57:26:63 REACHABLE
```

Ovdje vidimo našu **ARP** tablicu pa pogledajmo prvi unos. Za IP adresu (`192.168.1.1`) na mrežnom sučelju (`enp0s3`) vidimo da je njena pripadajuća **MAC** adresa (`b4:f5:8e:96:f8:50`) te da je ova IP adresa dostupna (`REACHABLE`).

**Za permanentno (trajno) dodavanje novog unosa u ARP tablicu, koristimo sintaksu poput:**

```
ip neigh add 192.168.1.10 lladdr 00:a1:32:ad:a8:00 dev enp0s3 nud permanent
```

Ovime smo trajno za IP (`192.168.1.10`) povezali MAC adresu (`00:a1:32:ad:a8:00`).

**Za brisanje prethodnog ARP unosa, možemo koristiti sljedeću naredbu:**

```
ip neighbour del 192.168.1.10 dev enp0s3
```

**Za brisanje svih ARP unosa koje trenutno imamo, možemo pokrenuti:**

```
ip neighbour flush all
```



Za dodatne primjere, pogledajte poglavlje:
**20.7.1.2. Rad s ARP protokolom**.

**Izvori informacija:** `man ip-neighbour`,`man ip`, `ip neighbour help`,`man 7 netlink`.

### 25.7.9.6. Tablice usmjeravanja

**Objekt** `route` koristi se za rad s tablicama usmjeravanja (*routing tablicama*). Dakle pomoću naredbe `ip route` radimo s tablicama usmjeravanja (*routing tablicama*). Pogledajmo brze primjere upotrebe naredbe `ip route`.

**Ispišimo sve tablice usmjeravanja koje imamo na sustavu:**

```
ip route list

default via 192.168.1.1 dev enp0s3 proto dhcp metric 100
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.129 metric 100
```

U prvom retku ispisa vidimo da je zadano (`default`) kako je uređaj na IP adresi `192.168.1.1` naš podrazumijevani usmjerivač (*Default Gateway*) te da se do njega dolazi preko mrežnog sučelja `enp0s3`.

**Moguće je provjeriti postoji li *ruta* do određene mreže, primjerice do mreže 192.168.1.0/24:**

```
ip route list 192.168.1.0/24
```

**Dodavanje podrazumijevane rute (*Default gateway* uređaja s IP adresom `192.168.1.1`) postižemo sljedeći način:**

```
ip route add default via 192.168.1.1
```

**Odnosno za brisanje podrazumijevane rute, možemo koristiti:**

```
ip route del default
```

**Dodajmo rutu za mrežu: 192.168.100.0 / 255.255.255.0, da ide preko *Gateway-a s IP adresom*: 192.168.1.100**

```
ip route add 192.168.100.0/24 via 192.168.1.100
```

**Odnosno za brisanje upravo dodane rute, možemo koristiti:**

```
ip route del 192.168.100.0/24 via 192.168.1.100
```

**Moguće je i provjeriti preko koje rute će se s našeg računala doći do mreže 192.168.100.0/24:**

```
ip -s route get 192.168.100.0/24
```



Za dodatne primjere, pogledajte sljedeća poglavlja:
**25.6.7. Konfiguracija pravila usmjeravanja (*ruta*)** te :
**25.6.8. Ruting i optimizacija ruting parametara**.

**Izvor informacija:** `man ip-route`,`man ip`,`ip route help`,`man 7 netlink`.

### 25.7.9.7. PBR (Policy based routing)

**Objekt** `rule` koristi se za rad s pravilima unutar tablica usmjeravanja (*routing policy*). Dakle pomoću naredbe `ip rule` radimo s pravilima unutar tablica usmjeravanja (tzv. *Policy based routing* mehanizmima). Pogledajmo nekoliko primjera:

Ispišimo sva pravila usmjeravanja koja imamo na sustavu:

```
ip rule ls
```

Kreirajmo pravilo usmjeravanja za sve koji dolaze s mreže `192.168.2.0/24`:da koriste tablicu usmj.: `serveri-1`:

```
ip rule add from 192.168.2.0/24 table serveri-1
```

Sada ćemo dodati i novo pravilo unutar tablice usmjeravanja serveri-1, a u kojem definiramo da je podrazumijevani usmjerivač (*default gateway*) za mrežu 192.168.2.0/24, uređaj s IP adresom: 192.168.1.254 jer imamo drugu mrežnu karticu eth1 u toj mreži, koja je spojena na njega. Dodajmo ovo novo pravilo usmjeravanja:

```
ip route add default via 192.168.2.254 table serveri-1
```

Sadržaj ove tablice (serveri-1) usmjeravanja možemo vidjeti sa:

```
ip route list table serveri-1
```



Za dodatne primjere, pogledajte poglavlje:
**25.6.8.1. *Routing policy* (*Policy based routing*).**

**Izvori informacija:(536),(537),(538),(539),(540),(541),(542),(543),(544),(545),(546),(547),(548),(549),(550),(551),(552),(553),(554),(555),(910)**,`man ip-rule`,`man ip`,`ip rule help`,`man 7 netlink`.

## 25.7.9.8. TUN i TAP sučelja

**Objekt `tuntap`** koristi se za rad s **TUN** i **TAP** mrežnim sučeljima. Dakle pomoću naredbe `ip tuntap` radimo s posebnim **TUN** i **TAP** mrežnim sučeljima. ***TUN*** i ***TAP*** su posebna virtualna mrežna sučelja koja se koriste za posebne namjene.
***TUN*** (Engl. *Network TUNnel*) simulira mrežno sučelje koje radi na OSI sloju tri (**IP**). Dakle ***TUN*** mrežno sučelje se koristi za namjene usmjeravanja (*routinga*) poput raznih ***VPN*** tunela (pr. ***OpenVPN***) i slično. S druge strane ***TAP*** (Engl. *Network TAP*) simulira mrežno sučelje na nižem sloju odnosno OSI sloju dva, pa prema tome radi s *ethernet* mrežnim okvirima.
Pogledajmo nekoliko primjera upotrebe ovih mrežnih sučelja.

**Kreirajmo *TAP* mrežno sučelje imena `tap0` i aktivirajmo ga:**
```
ip tuntap add tap0 mode tap
ip link set dev tap0 up
```

**Brisanje mrežnog *TAP* sučelja imena `tap0`, možemo napraviti sa:**
```
ip tuntap del tap0 mode tap
```

**Kreirajmo *TUN* mrežno sučelje imena `tun0`, i aktivirajmo ga:**
```
ip tuntap add tun0 mode tun
ip link set dev tun0 up
```
**Brisanje mrežnog *TUN* sučelja imena `tun0`, možemo napraviti sa:**
```
ip tuntap del tun0 mode tun
```



Za dodatne primjere, pogledajte sljedeće poglavlje:
**20.6.3. TUN i TAP - posebna mrežna sučelja.**

**Izvori informacija:(702),(703),(704),** `ip tuntap help`,`man ip`,`ip tuntap help`,`man 7 netlink`.

## 25.7.9.9. Sučelja za tuneliranje

**Objekt `tunnel`** koristi se za rad s tzv. *tunnel* mrežnim sučeljima. Ova posebna *tunnel* sučelja enkapsuliraju podatke u IP pakete i zatim ih šalju preko IP infrastrukture. Podržane vrste tunela su: ***ipip***, ***gre***, ***sit***, ***isatap***, ***vti***, ***ip6ip6***, ***ipip6***, ***ip6gre*** i ***vti6***.
**Za prikaz svih *tunnel* sučelja, pokrenimo:**
```
ip tunnel show
```

**Za kreiranje GRE tunela, na (udaljeni) poslužitelj s IP adresom: 192.168.1.100, napravimo**
```
ip tunnel add tunel1 mode gre remote 192.168.1.100
```

**Za brisanje istog tunela, trebamo pokrenuti sljedeću naredbu:**
```
ip tunnel del tunel1
```

**Izvor informacija: (1144),** `man ip-tunnel`,`ip tunnel help`,`man 7 netlink`.

### 25.7.9.10. VRF (virtual routing and forwarding)

**VRF** (engl. *virtual routing and forwarding*) tehnologija virtualnog usmjeravanja i prosljeđivanja omogućuje korisnicima konfiguriranje više instanci tablica usmjeravanja koje mogu istovremeno koegzistirati unutar istog usmjerivača. IP adrese koje se preklapaju mogu se koristiti bez sukoba jer su višestruke instance usmjeravanja neovisne i mogu odabrati različita odlazna sučelja. VRF-ovi se koriste za izolaciju odnosno virtualizaciju mreže na OSI sloju tri (OSI 3). Korisnici obično implementiraju VRF-ove prvenstveno kako bi odvojili mrežni promet i učinkovitije koristili mrežne usmjerivače ili uređaje koji obavljaju njihovu funkciju; primjerice kada se Linux koristi kao usmjerivač ili kao ***hipervizor*** za virtualna računala i njihovo mrežno razdjeljivanje. To se koristi u posebnim primjenama sustava virtualizacije (pr. *Cloud sustavi*) za izolaciju to jest upotrebu virtualnih računala unutar virtualnih mreža u oblaku (engl. *virtual cloud network*) to jest privatnih mreža u oblaku (engl. *private cloud network*). To uključuje dodjeljivanje i upotrebu privatnih IP adresnih prostora, kreiranje podmreža i tablica usmjeravanje te konfiguriranje vatrozida, sve unutar određenog VRF-a. To isto tako znači da se čak i IP adresni prostor unutar jednog VRF-a može preklapati s onim unutar drugog VRF-a. Što konkretno znači da primjerice sva virtualna računala unutar prvog VRF-a (pr. **VRF-A**) koja koriste IP adresni prostor: 10.10.10.0/24, mogu normalno mrežno komunicirati, iako postoji i drugi VRF (pr. **VRF-B**) unutar kojega je u upotrebi isti adresni prostor (10.10.10.0/24). Virtualnim usmjeravanjem i prosljeđivanjem također se mogu stvoriti VPN tuneli koji će biti posvećeni isključivo jednoj mreži ili klijentu.
Nekoliko je prednosti virtualnog usmjeravanja i prosljeđivanja:

- Omogućuje virtualno stvaranje više ruta na jednom fizičkom uređaju
- Omogućuje korisnicima istovremeno upravljanje višestrukim tablicama usmjeravanja
- Može se koristiti za implementacije ***MP BGP***-a i ***MPLS***-a
- Više VPN-ova za korisnike može koristiti IP adrese koje se preklapaju, bez sukoba
- Korisnici mogu segmentirati mrežne putove bez upotrebe više (fizičkih) usmjerivača, poboljšavajući mrežnu funkcionalnost.

**Objekt `vrf`** koristi se tako da takozvani **VRF** uređaj u kombinaciji s *ip pravilima* (*ip rules*) pruža mogućnost stvaranja virtualne domene za usmjeravanje i prosljeđivanje unutar mrežnog stôga Linuxa. Oni su *de facto* standard kada govorimo o izolaciji mreža na OSI sloju tri (OSI 3). Dakle *VRF* pruža izolaciju prometa na OSI sloju tri za usmjeravanje, slično kao što se *VLAN*-ovi koriste za izolaciju prometa na OSI sloju dva. Bazično ***VRF*** se oslanja na zasebne tablica usmjeravanja.
Proći ćemo kroz osnovnu teoretsku konfiguraciju, jer je ova tema poprilično kompleksna za razradu konkretnog slučaja.
Stoga prvo kreirajmo *VRF* i dodijelimo mu *ip pravilo* to jest zasebnu tablicu usmjeravanja (*ip rule* [***Policy Based Routing***]):

```
ip link add vrf-prvi type vrf table 10
ip link set dev vrf-prvi up
```

Nakon toga bi trebali dodavati *ip pravila* (`ip rule xy`) unutar tablice koja je ovdje nazvana `table 10`, te ostale rute.



Za dodatne primjere vezane za ***PBR (Policy Based Routing)***, pogledajte sljedeće poglavlje:
**25.6.8.1. *Routing policy* (*Policy based routing*).**

Pogledajte i priču o virtualizaciji i mrežama koje se koriste za njene potrebe:
**27.1.2. Rad s virtualizacijom (KVM+QEMU).**

I na kraju je potrebno povezati ovaj **VRF** s virtualnim/logičkim mrežnim sučeljem ili mrežnom karticom; primjerice s `eth1`:

```
ip link set dev eth1 master vrf-prvi
```

Za provjeru odnosno izlistanje svih **VRF**-ova, možemo koristiti naredbu:

```
ip vrf show
```

Za ostatak konfiguracije i primjere upotrebe, pogledajte izvore informacija. **Pripazite da je inačica kernela veća od 4.8!**

**Izvori informacija: (1146),(1147),(1148),(1216),(1217), `man ip-vrf`,`ip vrf help`,`man 7 netlink`.**

### 25.7.9.11. IPSec protokol

**Objekt `xfrm`** koristi se za rad s ***IPSec*** mrežnim protokolom. S obzirom na činjenicu da je konfiguracija *IPSec*-a znatno složenija i s mnoštvom parametara, dat ćemo vam samo neke osnovne parametre i primjere.
Sintaksa upotrebe je: `ip xfrm policy add` *SELECTOR* `dir` *DIR* [*LIMITS*] [*TEMPLATES*] [*MODE*]

**Pogledajmo jedan primjer konfiguracije IPSec tunela (s jedne strane) [redovi su razlomljen sa znakom `\` ]:**

```
ip xfrm policy add src 10.1.0.0/16 dst 10.2.0.0/16 proto icmp dir out \
    tmpl proto route2 dst 10.3.0.1 mode ro level use
```

**Izvori informacija: (1145), `man ip-xfrm`,`man ip`,`man 7 netlink`.**

## 25.7.10. Naredba ethtool

Naredbu `ethtool` smo već upoznali u mnogim poglavljima koja smo već prošli, a koristit ćemo ju i kasnije, ali ćemo ju ipak spomenuti i ovdje. Ova naredba radi na OSI razinama odnosno slojevima 1 i 2 (fizički i podatkovni sloj). Ova naredba se koristi za pregled ili promjene parametara mrežnih sučelja i mrežnih kartica, dakle logičkih i/ili fizičkih mrežnih sučelja.
Da bismo mogli koristiti ovu naredbu moramo instalirati dva programska paketa na sljedeći način
(iako su obično već instalirani):

```
yum -y install ethtool net-tools
```

Krenimo s kratkim primjerima njene upotrebe. U svim primjerima koristit ćemo mrežnu karticu (sučelje) imena `eth0`.

**1. Interrupt moderation mehanizam (mehanizam rada sa signalima prekida [*IRQ*]).**
Za isključivanje *interrupt moderation* mehanizma možemo koristiti sljedeću naredbu:

```
ethtool –C eth0 rx-usecs 0 tx-usecs 0 rx-frames 1 tx-frames 1
```

Za trajno isključivanje *interrupt moderation* mehanizma, u datoteku (za mrežno sučelje `eth0`):
`/etc/sysconfig/network-scripts/ifcfg-eth0` je potrebno dodati sljedeći redak konfiguracije:

```
ETHTOOL_OPTS="$ETHTOOL_OPTS ; -C eth0 rx-usecs 0 tx-usecs 0 rx-frames 1 tx-frames 1"
```

Postoji i druga metoda odnosno mehanizam rada, koji se zove adaptivna kontrola (engl. *Adaptive moderation*).
Ova funkcionalnost se može uključiti na sljedeći način i za RX (dolazni) i TX (odlazni promet):

```
ethtool –C eth0 adaptive-rx on
ethtool –C eth0 adaptive-tx on
```

**2. Informacije o mrežnoj kartici (sučelju) i upravljačkim programima koje koristi.**
U slučaju kada za primjerice mrežno sučelje **eth0** želimo saznati koji upravljački program koristi i koje značajke on podržava, potrebno je napraviti sljedeće. Opise smo stavili s desne strane uz ispis naredbe:

```
ethtool -i eth0

driver: e1000e                  ← Ovo je upravljački program koji se koristi (e1000e)
version: 5.16.14-1.el8.elrepo.x86_64  ← Inačica upravljačkog programa
firmware-version: 0.5-7               ← Inačica firmware-a mrežne kartice
expansion-rom-version:
bus-info: 0000:09:00.0          ← Ovdje vidimo na koji PCI utor je spojena mrežna kartica (09:00.0)
supports-statistics: yes        ← Upravljački program podržava statistike na nižoj razini (yes)
supports-test: yes              ← Ovdje vidimo da upravljački program podržava i testiranje (yes)
supports-eeprom-access: yes     ← Ovdje vidimo da podržava i pristup EEPROM memoriji (yes)
supports-register-dump: yes     ← Ovdje vidimo da podržava i ispis registara (yes)
supports-priv-flags: yes        ← Ovdje vidimo da podržava i tzv. privilegirane zastavice (yes)
```

Ova mrežna kartica je, kako vidimo, spojena na PCI utor oznake: `09:00.0`.
To možemo provjeriti s naredbom (skratili smo ispis):

```
lspci -s 09:00.0 -vv
09:00.0 Ethernet controller: Intel Corporation 82573L Gigabit Ethernet Controller
  Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR+
  Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort-
        Latency: 0, Cache Line Size: 64 bytes
        Interrupt: pin A routed to IRQ 24
        Capabilities: [c8] Power Management version 2
        Capabilities: [d0] MSI: Enable+ Count=1/1 Maskable- 64bit+
        Capabilities: [e0] Express (v1) Endpoint, MSI 00
         DevCap: MaxPayload 256 bytes, PhantFunc 0, Latency L0s <512ns, L1 <64us
         ExtTag- AttnBtn- AttnInd- PwrInd- RBE- FLReset- SlotPowerLimit 0.000W
                DevCtl: CorrErr- NonFatalErr- FatalErr- UnsupReq-
                LnkSta: Speed 2.5GT/s (ok), Width x1 (ok)
        Capabilities: [100 v1] Advanced Error Reporting
        Capabilities: [140 v1] Device Serial Number 00-08-60-ff-ff-00-79-20
        Kernel driver in use: e1000e
        Kernel modules: e1000e
```

Na kraju, poslije svih detalja o ovoj mrežnoj kartici i njenim mogućnostima te postavkama PCI sabirnice te signalu prekida koji koristi (`IRQ 24`), vidimo njen upravljački program (`e1000e`).



Za detalje o naredbi `lspci`, pogledajte poglavlje:
**3.4.3. Osnovne naredbe vezane uz operativni sustav i komponente računala.**

**3. Informacije u mrežnoj kartici i parametrima veze te podešavanja parametara veze.**
Informacije u mrežnoj kartici i parametrima veze možemo dobiti sa (ispisali smo samo osnovni dio, uz naše komentare):

```
ethtool eth0
Settings for eth0:
Supported ports: [ TP ]      ← Vrsta mrežnog sučelja (TP=Twisted-Pair [UTP])
Supported link modes: 10baseT/Half 10baseT/Full      ← Podržane brzine/duplex sučelja
                      100baseT/Half 100baseT/Full    ← Podržane brzine/duplex sučelja
                      1000baseT/Half 1000baseT/Full  ← Podržane brzine/duplex sučelja
Supported pause frame use: No  ← Je li podržana upotreba ethernet mrežnih okvira za pauziranje prometa
Supports auto-negotiation: Yes  ← Podržava li Auto negotiation (yes)
Supported FEC modes: Not reported  ← Podržava li FEC (Forward Error Correction) način rada (za 25/40/100+Gbps)
Advertised link modes:  ←  Oglašavane postavke veze se mogu mijenjati s opcijom --advertise
                      10baseT/Half 10baseT/Full  ← Oglašavana brzina/duplex sučelja
                      100baseT/Half 100baseT/Full    ← Oglašavana brzina/duplex sučelja
                      1000baseT/Half 1000baseT/Full  ← Oglašavana brzina/duplex sučelja
Advertised pause frame use: No  ← Oglašava li se upotreba ethernet mrežnog okvira za pauziranje prometa
Advertised auto-negotiation: Yes  ← Oglašava li se Auto negotiation
Advertised FEC modes: Not reported  ← Oglašava li se FEC (Forward Error Correction) način rada
Link partner advertised link modes:  ← Oglašene brzine sučelja koje šalje druga strana
              10baseT/Half 10baseT/Full      ← Oglašene brzine sučelja (druga strana)
              100baseT/Half 100baseT/Full    ← Oglašene brzine sučelja (druga strana)
              1000baseT/Full                 ← Oglašene brzine sučelja (druga strana)
Link partner advertised pause frame use: Symmetric ← Oglašavana upotreba mrežnog okvira za pauziranje prometa
Link partner advertised auto-negotiation: Yes  ← Oglašavana upotreba Auto negotiation načina rada (yes|no)
Link partner advertised FEC modes: Not reported  ← Oglašavana upotreba FEC načina rada
 Speed: 1000Mb/s   ← Dogovorena i postavljena brzina rada sučelja
 Duplex: Full      ← Dogovoren i postavljen Duplex način rada sučelja (Half|Full)
 Auto-negotiation: on  ← Dogovoren i postavljen Auto negotiation načina rada (on|off)
 Port: Twisted Pair  ← Vrsta mrežnog sučelja (TP|AUI|BNC|MII|Fibre)
 PHYAD: 1  ← Odnosi se na fizičku (MAC) adresu mrežnog sučelja. Može se mijenjati sa: --phyad.
 Transceiver: internal  ← Vrsta primopredajnika (električnih/optičkih) signala: (internal|external).
  MDI-X: on          ← Je li postavljen MDI-X način rada sučelja (on|off).
  Supports Wake-on: pumg  ← Koji WoL (Wake On Lan) načini rada su podržani (p|u|m|b|g|s|d)
    Wake-on: g  ← Koji Wake On Lan način rada je odabran (p|u|m|b|g|s|d)
   Current message level: 0x000000ff (255)  ← Razina logiranja (ispisa) grešaka (opcija: msglvl)
        drv probe link timer ifdown ifup rx_err tx_err  ← Nabrojano je sve što se logira
   Link detected: yes      ← Je li detektirana uspostava veze (yes|no)
```

**Ručno podešavanje brzine i duplex načina rada**
Pogledajmo kako isključiti ***Auto negotiation*** način rada:

```
ethtool  -s eth0 autoneg off
```

Odnosno kako ga ponovno uključiti:

```
ethtool  -s eth0 autoneg on
```

Sada pogledajmo kako ručno konfigurirati brzinu i *duplex* mrežnog sučelja uz isključen ***Auto negotiation*** način rada:

```
ethtool -s eth0 speed 1000 duplex full autoneg off
```

Pogledajmo i kako vidjeti postavljene brzinu i *duplex* način rada sučelja, kao i *ethernet pause* okvire (*Ethernet flow control*):

```
ethtool -a eth0
Pause parameters for eth0:
Autonegotiate:  on
RX:             on
TX:             on
RX negotiated: off
TX negotiated: off
```

Moguće je i ograničiti koje brzine i *duplex* načine rada naša kartica oglašava, pomoću prekidača: `--advertise`.

Moguće je isključiti i *Ethernet flow control (za pause okvire)* i za primanje (RX) i slanje (TX), a tada moramo napraviti sljedeće:

```
ethtool -A eth0 tx off rx off autoneg off
```

**Promjene Wake on Lan (WoL) načina rada su također moguće, a ovise o tome koji od njih podržava mrežno sučelje.**
Za mrežno sučelje isključimo **WoL** značajku sa:

```
ethtool --wol d eth0
```

Odnosno **Wake on Lan** (**WoL**) ponovno možemo uključiti da reagira na standardne **WoL** pakete sa sljedećom postavkom:

```
ethtool --wol g eth0
```

**Pogledajmo i kako povećati razinu logiranja (snimanja) grešaka na razini upravljačkog programa mrežnog sučelja**
U ovom primjeru podići ćemo razinu logiranja poruka na maksimalnu moguću (ove vrijednosti se mogu i zbrajati):
```
ethtool -s eth0 msglvl 0x000000ff
```

**4. Statistike primitka i slanja mrežnih paketa na razini mrežnog sučelja.**
Pogledajmo kako vidjeti statistike primitka i slanja mrežnih paketa na razini mrežnog sučelja (drastično smo skratili ispis):
```
ethtool -S eth0
NIC statistics:
     rx_octets: 85925
     rx_fragments: 0
     . . .
```


Za detalje o statistikama mrežnih paketa, pogledajte poglavlje:
**25.3. Statistike, analiza i praćenje mrežnih paketa.**

**5. Detekcija fizičke mrežne kartice, ako nismo sigurni koju od više mrežnih kartica u računalu konfiguriramo.**
Za detektiranje mrežne kartice (odabrana će blinkati [*svijetlit će joj diode sa stražnje strane*]) možemo koristiti naredbu:
```
ethtool -p eth0
```

**6. Dijagnostika mrežne kartice.**
Moguće je (za one mrežne kartice koje to omogućuju), pokrenuti i potpunu dijagnostiku rada mrežne kartice, koji uključuju test registra, EEPROM test, test signala prekida (IRQ), test povratne petlje i test same veze:
```
ethtool --test eth0 offline
```
← Rezultati testa i sam test ovisi o upravljačkom programu i samoj mrežnoj kartici!

Za potrebe dodatne dijagnostika, moguće je koristiti vremensko označavanje mrežnih paketa na razini mrežnog *socketa* (mrežne/programske utičnice). Mogućnosti mrežne kartice po pitanju ovakvog označavanja mrežnih paketa možemo vidjeti sa:
```
ethtool -T eth0
```

Moguće je napraviti i takozvani *firmware dump* odnosno analizu firmware-a mrežne kartice, sa sljedećom naredbom:
```
ethtool -d eth0 raw on > /root/lan.eth0.dump.raw
```
Tada centru za tehničku podršku (pr. *RedHat*/*Rocky*/*Oracle*/…) treba proslijediti datoteku koju smo izradili: `/root/lan.eth0.dump.raw`.

## Konfiguracija hardverski ubrzanih i drugih naprednih značajki

U ovoj cjelini proći ćemo kroz načine konfiguracije nekoliko hardverski ubrzanih značajki (ako ih mrežna kartica podržava), kao i drugih naprednih značajki. Ove značajke se mogu vidjeti na sljedeći način (drastično smo skratili ispis):
```
ethtool -k eth0

rx-checksumming: on [fixed]
tx-checksumming: on
rx-vlan-filter: on [fixed]
rx-all: off [fixed]
tx-checksum-ipv4: off [fixed]
tx-vlan-stag-hw-insert: off
tx-checksum-ip-generic: on
tx-checksum-ipv6: off [fixed]
macsec-hw-offload: off [fixed]
tx-gso-robust: on [fixed]
tx-checksum-sctp: off
hw-tc-offload: off [fixed]
scatter-gather: on
tx-scatter-gather: on
. . .
tcp-segmentation-offload: on
tls-hw-tx-offload: off [fixed]
tx-tcp-segmentation: on
tls-hw-rx-offload: off [fixed]
rx-gro-hw: on [fixed]
tx-gso-partial: off [fixed]
tx-tcp6-segmentation: on
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off [fixed]
tx-udp-segmentation: off [fixed]
rx-vlan-offload: off [fixed]
tx-vlan-offload: off [fixed]
. . .
```
One značajke koje su aktivirane imaju oznaku `on`, dok one koje nisu imaju oznaku `off`, a one koje se ne mogu koristiti imaju oznaku `off [fixed]`.

**Aktivacija hardverske podrške (mrežne kartice) za rasterećenje VLANova**
Ako naša mrežna kartica podržava rasterećenje procesora (*offload*) za rad s VLAN paketima i to u dolazu (*rx-vlan*) i odlazu (*tx-vlan*), obično ćemo vidjeti da je oboje uključeno. To je i preporuka jer želimo rasteretiti CPU i obradu VLAN paketa (rad s 802.1Q zaglavljem) u potpunosti prebaciti na mrežnu karticu (ako to ona podržava).
```
ethtool -k eth0 | grep vlan
rx-vlan-offload: on
tx-vlan-offload: on
```

Ako primjerice *rx-vlan* baš želimo isključiti, to možemo postići sa:

```
ethtool -K eth0 rxvlan off
```

Odnosno, ako želimo isključiti *tx-vlan* tada trebamo napraviti sljedeće:

```
ethtool -K eth0 txvlan off
```

***Ako već imamo ovu podršku u hardveru, dobro je imati ju uključeno da bi rasteretili CPU od obrade VLAN zaglavlja.***

**Prstenaste međumemorije (Ring buffer)**

Kada su podaci u nîzu za slanje, upravljački program (driver) ih potom dohvaća i zapisuje u posebnu takozvanu prstenastu memoriju odnosno tzv. *TX ring*, koji mrežna kartica vrlo brzo može dohvatiti i pripremiti, a potom pohraniti u svoju posebnu internu ekstremno brzu međumemoriju (*NIC internal buffer*) koja se nalazi na sâmoj mrežnoj kartici. Kako se ova vrlo brza interna međumemorija mrežne kartice popuni s većim brojem paketa, mrežna kartica ih može, sve od jednom u nîzu i poslati vrlo brzo, na mrežu. Pogledajmo koje su veličine ove memorije postavljene, na sljedeći način (`-g` za čitanje vrijednosti):

```
ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
RX: 4078
RX Mini: 0
RX Jumbo: 0
TX: 4078
Current hardware settings:
RX: 1333
RX Mini: 0
RX Jumbo: 0
TX: 4078
```

Vidimo maksimalne moguće vrijednosti (4078 za RX i TX u konkretnom slučaju), ali i one koje su postavljene (1333 za **RX**). Sada povećajmo obje prstenaste međumemorije (i TX i RX ring *buffer*) na 4078, korištenjem prekidača (veliko slovo) `-G`:

```
ethtool -G eth0 rx 4078 tx 4078
```



Za detalje pogledajte poglavlje:
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**

**Receive Side Scaling (RSS)**

*Receive Side Scaling* (***RSS***) tehnologija je koja distribuira dolazne pakete na više CPU jezgri i to tako da svi paketi koji dolaze s jedne logičke konekcije (pr. pojedina TCP konekcija: klijent-poslužitelj) uvijek završe na obradi na istoj CPU jezgri procesora. Na Linuxu je moguće vidjeti i mijenjati (unutar granica hardvera i upravljačkog programa), veličinu takozvanih RX i TX prstenastih međumemorija (*ring buffers)* odnosno nîzova.

Podržava li mrežna kartica **RSS**, vidjet ćemo RX prsten za svaki RX kanal, što možemo provjeriti sa (`-x`):

```
ethtool -x eth0
```

Istu informaciju o broju **RSS** kanala možemo vidjeti sa (prekidač `-n`):

```
ethtool -n eth0
```

U slučaju kada imamo primjerice 8 **RSS** kanala, možemo dodatno naznačiti sustavu raspodjelu mrežnih paketa jednoliko na sve kanale, sa (prekidač `-X`):

```
ethtool -X eth0 equal 8
```

Ovdje su moguće i mnoge druge opcije.



Za detalje pogledajte poglavlje:
**25.2.1. *Receive Side Scaling* (*RSS*).**

**Multiqueue značajka (*receive queue*)**

Upravljački program koji upravlja mrežnom karticom koja hardverski podržava *multiqueue* značajku, preko svog upravljačkog programa daje nam parametar s kojim možemo definirati koliki broj nîzova (*receive queue*) želimo koristiti. Naravno ovisno o maksimalnom broju koliko ih sâma mrežna kartica hardverski podržava te broju CPU jezgri koje imamo. U idealnom slučaju bilo bi ih dobro koristiti maksimalno onoliko koliko imamo CPU jezgri na računalu.

Svaki prijemni nîz (*receive queue*) pri tome koristi svoj hardverski signal prekida (***IRQ***), a svaki signal prekida se potom veže za svoju CPU jezgru.

Za provjeru podržava li mrežna kartica više dolaznih nîzova, možemo koristiti:

```
ethtool -l eth0
Channel parameters for eno1:
Pre-set maximums:
RX: 0
TX: 0
Other: 0
Combined: 30
Current hardware settings:
RX: 0
TX: 0
Other: 0
Combined: 8
```

Na ispisu je vidljivo kako naš hardver (`Pre-set maximums`) mrežne kartice podržava do 30 kanala (`Queue`), ali da mi koristimo njih samo 8 (`Current hardware settings`) što je u redu jer imamo samo 8 CPU jezgri. Ovdje vidimo i kako naš hardver mrežne kartice podržava sve do 30 kombiniranih (TX+RX) kanala, zbog toga naziv „`Combined`". Dakle preporuka je da bi jedna CPU jezgra trebala biti zadužena za jedan kanal. U slučaju kada bi imali primjerice 12 CPU jezgri i istu mrežnu karticu, povećali bi broj ovih kanala na 12 i to ovako (prekidač `-L`):

```
ethtool -L eth0 combined 12
```



Za detalje pogledajte poglavlje:
**25.2.2. Multiqueue funkcionalnost**.

**Receive Flow Steering (RFS)**

Neke snažnije mrežne kartice poput *Intelovih* X520, X540, XL710 i novijih, podržavaju i hardverski ubrzanu ***RFS*** značajku koja se naziva *Accelerated RFS* ili *aRFS*. Naime poput klasičnog ***RFSa***, paketi se ovdje prosljeđuju na temelju lokacije aplikacije koja treba obraditi određeni paket odnosno nîz paketa. Međutim za razliku od tradicionalnog **RFSa**, paketi se šalju izravno na CPU jezgru koja je lokalno u upotrebi za izvršavanje programa ili njegove programske nîti, koja i treba te podatke, odnosno koja ih treba obraditi.

Za provjera podržava li to mrežna kartica, možemo vidjeti na sljedeći način:

```
ethtool -k eth0 | grep ntuple
```

Ako dobijemo nešto poput:

```
ntuple-filters: off
```

Odnosno imamo li ovdje vrijednost `off` te nemamo ništa u zagradi nakon toga, odnosno u zagradi nemamo i `(fixed)` to znači kako ipak imamo hardversku podršku. Ako naša mrežna kartica podržava *aRFS*, ali on nije uključen, probajmo ga uključiti sa:

```
ethtool -k eth0 ntuple on
```

Ako je sve u redu, provjerit ćemo status:

```
ethtool -k eth0 | grep ntuple
ntuple-filters: on
```

S ovime vidimo da je podrška uključena.



Za detalje pogledajte poglavlje:
**25.2.3. Receive packet steering (RPS), Receive flow steering (RFS) i Transmit Packet Steering (XPS).**

**Large receive offload (LRO)**

*Large receive offload* **(LRO)** je tehnika za povećanje propusnosti za dolazni mrežni promet, smanjivanjem potrebe za kontaktiranjem procesora (CPU-a) kod zaprimanja svakog pojedinog mrežnog paketa. ***LRO*** djeluje skupljanjem višestrukih dolaznih paketa iz jednog tôka, u veći međuspremnik prije nego što se svi oni proslijede na više mrežne slojeve.

Kako uključiti LRO?

Prvo provjerimo podržava li ***LRO*** naša mrežna kartica i u kojem je trenutnom stanju::

```
ethtool -k eth0 | grep large-receive-offload
```

Sve dok imamo sljedeći odgovor:

```
large-receive-offload: off
```

Odnosno dok nemamo `off [fixed]` sve je u redu jer imamo (hardversku) podršku za ***LRO***. Uključiti ga možemo sa:

```
ethtool -K eth0 lro on
```

**Napomena: neki mrežni protokoli se ne ponašaju najbolje s uključenim *LRO!***

Pogledajmo i kako isključiti ***LRO***:

```
ethtool -K eth0 lro off
```

***Generic Receive Offloading* (GRO)**

*Generic Receive Offloading* (**GRO**) je praktično softverska implementacija ***LRO*** funkcionalnosti uz još neke optimizacije. Rješenje za potencijalno problematično ponašanje ***LRO*** je upotreba ***GRO*** mehanizma koji je također dostupan u Linuxu.
Pogledajmo je li **GRO** uključen za našu mrežnu karticu:

```
ethtool -k eth0 |grep generic-receive-offload
generic-receive-offload: on
```

Vidimo kako je uključen (`on`), a ako bi bilo potrebno uključiti ga kada je isključen, to bi napravili sa sljedećom naredbom:

```
ethtool -K eth0 gro on
```

Ako ga ipak želimo isključiti, što u nekim slučajevima može biti korisno (u slučaju problema u radu), to možemo napraviti sa:

```
ethtool -K eth0 gro off
```



Za detalje pogledajte poglavlje:
**25.5.1. Large receive offload (LRO) i Generic receive offload (GRO).**

**LSO, TSO i GSO**

*Large Send Offload* (***LSO***) je tehnika za povećanje propusnosti za odlazni promet smanjivanjem kontaktiranja centralnog procesora (CPU) tijekom slanja svakog pojedinog mrežnog paketa. Ona djeluje tako što skuplja više mrežnih paketa s viših mrežnih (OSI) slojeva u jedan veći međuspremnik koji se onda šalje na mrežnu karticu. Mrežna kartica zatim sâma razlama taj međuspremnik u zasebne pakete. Ova tehnika kada se primjenjuje na TCP se naziva *TCP segmentation offload* (**TSO**) odnosno generički gledano mehanizam koji nije specifičan samo za TCP protokol se naziva *Generic segmentation offload* (**GSO**).
Pogledajmo kako uključiti ***TSO*** za mrežnu karticu:

```
ethtool -K eth0 tso on
```

Pogledajmo i što smo sada dobili:

```
ethtool -k eth0 | grep -i tcp
tcp-segmentation-offload: on
tx-tcp-segmentation: on
tx-tcp6-segmentation: on
```

Dakle vidimo globalnu vrijednost (`tcp-segmentation-offload`) uključenu (`on`). Isto tako vidimo kako je ***TSO*** uključen i za TCP v.4 (`tx-tcp-segmentation`) i TCP v.6. (`tx-tcp6-segmentation`). Sada je potrebno napraviti mjerenje performansi sustava, i vidjeti je li se mreža ubrzala. Ako ipak želimo isključiti ***TSO***, trebamo napraviti sljedeće:

```
ethtool -K eth0 tso off
```

**Generic segmentation offload (GSO)**

Ako želimo uključiti ***GSO***, to možemo učiniti na sljedeći način:

```
ethtool -K eth0 gso on
```

Pogledajmo stanje koje imamo sada:

```
ethtool -k eno1 | grep -i generic-segmentation-offload
generic-segmentation-offload: on
```

U drugom slučaju kada ipak želimo isključiti ***GRO***, to možemo napraviti sa:

```
ethtool -K eth0 gso off
```



Za detalje pogledajte poglavlje:
**25.5.2. LSO, TSO i GSO.**

**Checksum offload**

Tehnologija naziva *checksum offload* je tehnologija s kojom mrežna kartica sama izračunava TCP/UDP ili IP provjerni zbroj (*checksum*) za svaki TCP/IP mrežni paket te tako rasterećuje CPU koji bi bez ove akceleracije (ubrzanja od strane mrežne kartice) sâm morao izračunavati provjerne zbrojeve. Pogledajmo kako uključiti ovu funkcionalnost:

```
ethtool -K eth0 rx on tx on
```

Sada provjerimo što smo promijenili:

```
ethtool -k eth0 | grep -i check
rx-checksumming: on
tx-checksumming: on (tx-checksum-ipv4: on) (tx-checksum-ipv6: on)
```

Vidimo da je sada sve uključeno.



Za detalje pogledajte poglavlje:
**25.5.3. Checksum offload.**

**Scatter/Gather**
U računalnim mrežama vektorski ulaz/izlaz (Engl. *Vectored* I/O) također je poznat kao *Scatter/Gather I/O*. To je metoda ulaza i izlaza podataka u kojoj se jednim pozivom sekvencijalno čitaju podaci iz više memorijskih međuspremnika, a potom zapisuju u jedan nîz ili tok podataka. S druge strane je to metoda u kojoj se čita podatke iz toka podataka, a zapisuje ih se na više memorijskih međuspremnika, kako je definirano u vektoru međuspremnika. Ova tehnologija obično ubrzava mrežnu obradu.
Pogledajmo trenutno stanje naše mrežne kartice:

```
ethtool -k eth0 | grep -i gather
scatter-gather: on
tx-scatter-gather: on
```

Vidimo kako je *Scatter/gather* uključen (`on`). Pogledajmo kako ga ručno isključiti za mrežnu karticu:

```
ethtool -K etho sg off
```

Odnosno pogledajmo i kako ga prema potrebi uključiti:

```
ethtool -K etho sg on
```


Za detalje pogledajte poglavlje:
**25.5.4. *Scatter-gather*.**

## SFP moduli
Ukoliko mrežna kartica koju koristimo ima SFP utor i u njemu određeni SFP modulč informacije o tom SFP modulu možemo dobiti sa (pr. za `eth0` mrežno sućelje):

```
ethtool -m eth0
```

ili

```
ethtool --module-info eth0
```


Za detalje pogledajte poglavlje:
**19.2.2.2. Small Form-factor (SFF) i Small Form-factor Pluggable (SFP).**

## Trajna aktivacija *ethtool* opcija
Za trajnu aktivaciju željene ***ethtool*** opcije, za mrežno sučelje `eth0`, u datoteku:
`/etc/sysconfig/network-scripts/ifcfg-eth0` je potrebno dodati sljedeći redak konfiguracije:

```
ETHTOOL_OPTS="$ETHTOOL_OPTS ; XYZ"
```

Pri tome kao **"*XYZ*"** trebamo upisati željene ***ethtool*** opcije.
Primjerice za trajnu ručnu postavku brzine na 1000Mbps (1Gbps) i *duplex* načina rada u *full duplex*, potrebno je dodati redak:

```
ETHTOOL_OPTS="$ETHTOOL_OPTS ; speed 1000 duplex full"
```


Za druge primjere i detalje upotrebe ove naredbe pogledajte sljedeća poglavlja:
**10.5.2.2. *Interrupt moderation*.**
**11.1.2.1.1. *Udev* pravila (*udev rules*).**
**11.1.2.1.2. Nadogradnja *firmwarea* uređaja.**
**17.3.1. Kratka analiza problema na sustavu.**
**19.6.1. Konfiguracija brzine te *duplex* i *auto negotiation* načina rada.**
**20.5. Kontrola protoka (*flow control*) na OSI sloju 2.**
**20.6.1.1. Privremena konfiguracija *bridge* mrežnih sučelja.**
**20.6.2.3. Rad s VLANovima pod Linuxom.**
**25.1.3. Prstenasta međumemorija (*Ring buffer*) [*TX Ring* i *RX Ring*].**
**25.2. Dodatne mogućnosti linuxa.**
**25.2.1. *Receive Side Scaling* (*RSS*).**
**25.2.2. Multiqueue funkcionalnost.**
**25.2.3. Receive packet steering (RPS), Receive flow steering (RFS) i Transmit Packet Steering (XPS).**
**25.3. Statistike, analiza i praćenje mrežnih paketa.**
**25.5.1. Large receive offload (LRO) i Generic receive offload (GRO).**
**25.5.2. LSO, TSO i GSO.**
**25.5.3. Checksum offload.**
**25.5.4. *Scatter-gather*.**
**25.5.5. Pregled navedenih dostupnih tehnika, tehnologija i protokola.**
**25.6.5.1.2. Trajna (permanentna) ručna konfiguracija mreže.**
**27.1.2. Rad s virtualizacijom (KVM+QEMU).**

Pogledajmo listu svih navedenih, ali i još pokojih korisnih opcija naredbe `ethtool`, uz njihov kratki opis:

- `-a` - za ispis upotrebe *Ethernet* okvira za pauziranje (*Ethernet flow controll*).
    - `-A` - za promjenu upotrebe *Ethernet* okvira za pauziranje (*Ethernet flow controll*):
        - `autoneg` – definira hoće li se *Ethernet pause* okviri koristiti u auto-negotiation (`on`|`off`).
        - `rx` – definira hoće li se *RX Ethernet pause* okviri koristiti (`on`|`off`).
        - `tx` – definira hoće li se *TX Ethernet pause* okviri koristiti (`on`|`off`).
- `-c` – za ispis *interrupt moderation* mehanizma mrežnog sučelja.
    - `-C` za promjene *interrupt moderation* mehanizma mrežnog sučelja.
- `--cable-test-tdr` - za testiranje kablova (*Time Domain Reflectometry*), ako to kartica podržava.
- `-d` - za ispis sadržaja *firmware*-a mrežne kartice (*firmware dump*).
- `-g` - za ispis prstenaste memorije (*ring buffer*).
    - `-G` - za promjene prstenaste memorije (*ring buffer*).
- `-i` - za ispis informacija o upravljačkom programu i drugih općih informacija o mrežnom sučelju.
- `-k` - za ispis hardverski ubrzanih i ostalih naprednih značajki.
    - `-K` - za promjene ubrzanih i ostalih naprednih značajki.
    - `-rx` ***XY*** - za *RX* ***checksum*** *offload* značajku (`on`|`off`).
    - `-tx` ***XY*** - za *TX* ***checksum*** *offload* značajku (`on`|`off`).
    - `-sg` ***XY*** - za ***scatter-gather*** značajku (`on`|`off`).
    - `-tso` ***XY*** - za ***TSO*** značajku (`on`|`off`).
    - `-ufo` ***XY*** - za *UDP fragmentation offload* značajku (`on`|`off`).
    - `-gso` ***XY*** - za ***GSO*** značajku (`on`|`off`).
    - `-gro` ***XY*** - za ***GRO*** značajku (`on`|`off`).
    - `-lro` ***XY*** - za ***LRO*** značajku (`on`|`off`).
    - `-rxvlan` ***XY*** - za ***VLAN*** *offload* (za RX) značajku (`on`|`off`).
    - `-txvlan` ***XY*** - za ***VLAN*** *offload* (za TX) značajku (`on`|`off`).
    - `-ntuple` ***XY*** - za RX ***ntuple*** *filtere* (`on`|`off`).
- `-l` - za ispis postavki ***RSS***.
    - `-L` - za promjene postavki ***RSS***.
- `-m` - za ispis informacija o **SFP** modulima (ako se nalaze u SFP utoru mrežne kartice).
- `-p` - za vizualni test (identifikaciju) mrežne kartice (treperenje dioda na kartici).
- `-r` - za restart procedure *Auto-negotiation* mrežne kartice, ako je uključen.
- `--reset` ***XY*** - za reset pojedinih dijelova mrežne kartice (pr. `mgmt`|`irq`|`dma`|`mac`|`phy`|…).
- `-s` - za promjene rada opcija; primjerice:
    - `eth0 advertise` ***XY*** - za definiciju parametara veze koji će se oglašavati (*postoji cijela lista*).
    - `eth0 autoneg` ***XY*** - za promjenu *Auto-negotiation* načina rada mrežnog sučelja (`on`|`off`).
    - `eth0 duplex` ***XY*** - za promjenu *duplex* načina rada mrežnog sučelja (`duplex` ili `full`).
    - `eth0 mdix` ***XY*** - za promjenu *MDI-X* načina rada mrežnog sučelja (`auto`|`on`|`off`).
    - `eth0 msglvl` ***XY*** - za promjenu razine logiranja poruka na razini upravlj. programa.
    - `eth0 phyad` ***XY*** - za promjenu *MAC adrese* mrežne kartice.
    - `eth0 port` ***XY*** - za promjenu *porta*, ako mrežna kartica ima više portova (`tp`|`aui`|`bnc`|`mii`).
    - `eth0 speed` ***XY*** - za promjenu brzine rada mrežnog sučelja (100,1000,10000, …)
    - `eth0 wol` ***XY*** - za promjenu *Wake on Lan* načina rada (`p`|`u`|`m`|`b`|`a`|`g`|`s`|`f`|`d`):
        - `p` – buđenje nakon *PHY* aktivnosti (na mrežnom sučelju).
        - `u` – buđenje nakon *Unicast* poruka.
        - `m` – buđenje nakon *Multicast* poruka.
        - `b` – buđenje nakon *Broadcast* poruka.
        - `a` – buđenje nakon *ARP* poruka.
        - `g` – buđenje nakon *WoL Magic paketa*.
        - `s` – omogućavanje *SecureOn* lozinke za *WoL Magic paket*.
        - `f` – ***WoL*** filteri.
        - `d` – onemogućavanje *WoL* (isključivanje).
    - `eth0 xcvr` ***XY*** - za promjenu *primopredajnika unutar* mrežne kartice (`internal`|`external`)
- `--show-fec` - za ispis **FEC** (*Forward Error Correction*), ako to kartica podržava (za 25/40/100Gbps).
- `--set-fec` ***XY*** - za postavljanje **FEC**, ako to kartica podržava (za 25/40/100Gbps kartice).
- `-S` - za ispis statistika (mrežnih paketa) na razini upravljačkog programa (ako to upravljački program podržava).
- `--test` ***XY*** - za testiranje funkcionalnosti mrežne kartice: registri, EEPROM, IRQ (`offline`|`online`)
- `-T` - za definiranje upotrebe to zapisivanja vremenskih oznaka uz pakete (za dijagnostiku).

**Izvori informacija: (1369),(1370),(1371),(1372),(1373),(1374), `man ethtool`.**

# 25.8. Popis (i opis) osnovnih mrežnih protokola i servisa

Zbog potrebe za razumijevanjem rada mreža, potrebno je okvirno shvatiti i način rada osnovnih mrežnih protokola. Stoga ćemo u narednim poglavljima spomenuti neke od osnovnih mrežnih protokola i njihovih pripadajućih servisa odnosno takozvanim *daemona* kao i neke od konfiguracijskih datoteka važnih za njihov rad.

---

## 25.8.1. Datoteka `/etc/services`

Svi UNIX i Linux operativni sustavi imaju posebnu servisnu datoteku: `/etc/services`. U ovoj datoteci pohranjene su informacije o brojnim uslugama (*servisima*) koje klijentske aplikacije mogu koristiti na računalu. Unutar ove datoteke je definiran: naziv servisa, broj porta i protokol koji se koristi, kao i svi primjenjivi pseudonimi naziva za taj servis/protokol.
U ovoj datoteci svaki novi redak predstavlja definiciju jednog porta odnosno mrežnog protokola. Dakle broj porta se povezuje za određeni mrežni servis, slično kao i kod ***hosts*** datoteke (`/etc/hosts`) u kojoj je za svaku IP adresu definirano mapiranje odnosno povezivanje s pripadajućim imenom računala. Međutim ova datoteka servisa UNIX/Linux operativnog sustava ne uključuje IP adrese već informacije o tome koji je transportni protokol; TCP i/ili UDP u upotrebi za točno definirani port odnosno mrežni protokol. Ova datoteka se prema tome koristi za pretvaranje broja porta u naziv servisa odnosno mrežnog protokola, a koju mogu koristiti i koriste mnoge naredbe u Linuxu. Sintaksa ove datoteke je sljedeća:
*IME MREŽNOG PROTOKOLA    BROJ PORTA/TRANSPORTNI PROTOKOL    ALIAS (PSEUDONIM) #KOMENTAR*

Pogledajmo nekoliko prvih redova ove datoteke (izrezali smo nezanimljive dijelove i ostavili samo one većini poznate):

```
cat /etc/services

ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
bootps          67/tcp                          # BOOTP server
bootps          67/udp
bootpc          68/tcp                          # BOOTP client
bootpc          68/udp
tftp            69/udp
...
http            80/tcp          www             # WorldWideWeb HTTP
http            80/udp                          # HyperText Transfer Protocol
pop3            110/tcp         pop-3           # POP version 3
pop3            110/udp         pop-3
```

Naime kada god neka naredba (ili neki drugi servis) ima potrebu pretvoriti ime mrežnog porta u protokol, prvo se čita upravo ova datoteka u kojoj je definicija svih poznatih protokola/portova i iz koje se onda povezuje to ime odnosno naziv.

Primjerice kada pokrenemo naredbu `netstat` na način kada nas zanimaju brojevi portova (`-tapn`) bez pretvaranja u ime servisa, dobivamo broj porta koji koristi određeni mrežni servis (ispis smo skratili na samo par servisa):

```
netstat -tapn

Proto Recv-Q Send-Q Local Address    Foreign Address        State       PID/Program
name
tcp        0      0 0.0.0.0:22       0.0.0.0:*              LISTEN      4298/sshd
tcp        0      0 0.0.0.0:80       0.0.0.0:*              LISTEN      4161/apache
tcp        0      0 0.0.0.0:443      0.0.0.0:*              LISTEN      4161/apache
tcp        0      0 0.0.0.0:514      0.0.0.0:*              LISTEN      3804/rsyslogd
```

Ono što vidimo na ovom ispisu je činjenica kako na našem poslužitelju imamo pokrenute sljedeće mrežne servise:
**ssh** (TCP port `22`), **http** (TCP port `80`), https (TCP port `443`) te syslog servis (TCP port `514`).
Međutim mi ovdje u stupcu (`Local Address`) ne vidimo imena servisa već sâmo brojeve portova.
Kada bi ipak ovu naredbu pozvali bez prekidača `-n` dobili bi sljedeće:

```
netstat -tap

Proto Recv-Q Send-Q Local Address     Foreign Address   State       PID/Program name
tcp        0      0 *:ssh                    *:*        LISTEN      4298/sshd
tcp        0      0 *:http                   *:*        LISTEN      4161/apache
tcp        0      0 *:https                  *:*        LISTEN      4161/apache
tcp        0      0 *:syslog              *:*        LISTEN      3804/rsyslogd
```

Dakle sada se umjesto portova (`22`, `80`, `443` i `514`) pomoću datoteke: `/etc/services` prepoznaju imena servisa, pa umjesto broja portova imamo redom nazive: `ssh`, `http`, `https` i `syslog` jer su sada brojevi portova zamijenjeni s imenima njihovih pripadajućih mrežnih servisa.

**Izvori informacija: `man 5 services`.**

---

### 25.8.2. TFTP i FTP protokoli

**TFTP** (*Trivial File Transfer Protocol*) i **FTP** (*File Transfer Protocol*) naizgled izgledaju isti ili barem vrlo slični.
Zapravo se radi o dva prilično različita protokola kojima je jedino zajedničko to što se koriste za prijenos datoteka preko mreže. Ovdje prestaju sve sličnosti. Pogledajmo o čemu se radi.

#### 25.8.2.1. TFTP

*Trivial File Transfer Protocol* (**TFTP**) je vrlo jednostavan protokol za razmjenu datoteka preko mreže. Točnije rečeno omogućava klijentskoj strani da dohvati (engl. metoda *Get*) ili pošalje (engl. metoda *Put*) datoteku s ili na *TFTP* poslužitelj.
Dakle logika rada *TFTP* protokola je: klijent ←→ poslužitelj. Najčešća dva primjera upotrebe ovog protokola su:
**1.** Korištenje za instaliranje (kopiranje) operativnog sustava, koji se u pravilu nalazi u jednoj datoteci, na neki uređaj, što se ponekad naziva i *Flash-iranje* odnosno učitavanje cijelog operativnog sustava na uređaj (pr. za *set-top-box* uređaje i sl.).
**2.** Za pokretanje ili instaliranje dijela (određenih datoteka) ili cijelog operativnog sustava preko mreže (o ovoj metodi kasnije).

***TFTP*** je dizajniran da bude mali i jednostavan za implementaciju, pa mu zato nedostaje većina naprednih značajki koje nude robusniji protokoli za prijenos datoteka. ***TFTP*** omogućava čitanje ili zapisivanje datoteke s udaljenog poslužitelja ili na njega, ali ne podržava izlistanje, brisanje ili preimenovanje datoteka ili direktorija te nema metode za autentifikaciju korisnika.
Danas se *TFTP* obično koristi sâmo u lokalnim mrežama (**LAN**). ***TFTP*** se koristi zbog toga što je vrlo jednostavan za implementaciju, a sâma implementacija zahtjeva vrlo malo prostora za pohranu. Dakle cijeli *TFTP* program može se pohraniti i u vrlo male *Flash* memorije, čak na *flash* memoriju sâme mrežne kartice. *TFTP* protokol koristi **UDP** protokol za transport, što ga čini ne zahtjevnim i brzim, ali i neotpornim na greške. Zbog upotrebe **UDP**-a kao brzog, ali i nesigurnog, ***TFTP*** protokol ima ugrađen dodatni mehanizam provjere primitka podataka; slično poput **TCP**-a.

**TFTP** protokol je prvo definiran u **RFC783**, a potom u **RFC1350** te **RFC2347**. **TFTP protokol koristi UDP port 69.**
Prijenos podatak pokreće klijentska strana, tražeći čitanje (**RRQ**) ili zapisivanje (**WRQ**) određene datoteke na *TFTP* poslužitelju.
Ako je poslužitelj odobrio zahtjev, prijenos podataka kreće slanjem na niži (***UDP***) sloj mreže preko koje se šalju mrežni paketi, u tzv. blokovima koji sadrže **512** bajta podataka, a koje *TFTP* označava rednim brojevima.
Međutim moguće je i da se dvije strane dogovore za drugu (obično veću) veličinu od standardno zadanih 512 bajta. Svaki poslani paket se potvrđuje slanjem potvrdnog paketa (engl. *Acknowledge*), što se odrađuje na aplikacijskoj razini *TFTP-a,* pošto **UDP** ne nudi ovu mogućnost. U slučaju kada neki paket nije potvrđen (na razini *TFTP*-a), on se ponovno šalje.
Ako se neki paket izgubi u mreži tj. nikada ne dođe na odredište, čeka se isticanje vremenskog okvira za slanje (engl. *Timeout*) te pošiljatelj (*TFTP* poslužitelj) ponovno šalje isti paket.


Za više detalja pogledajte poglavlje o instalaciji i konfiguraciji ***TFTP*** servisa:
**26.3. TFTP**.

**Izvor informacija: (717), RFC783, RFC1350, RFC2347.**

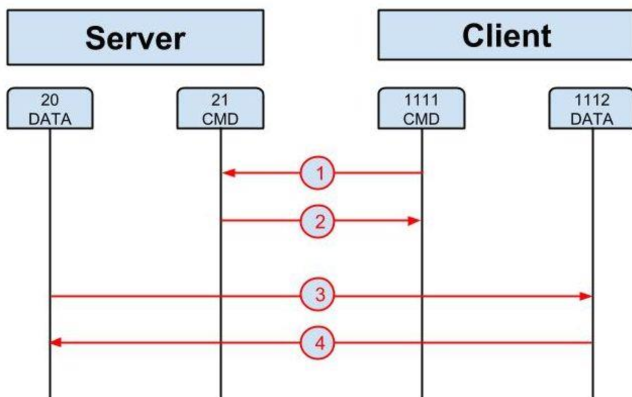
#### 25.8.2.2. FTP

*File Transfer Protocol* (**FTP**) je mrežni protokol za prijenos datoteka preko mreže. On za razliku od *TFTP* protokola koristi **TCP** protokol za prijenos te se sâmim time i oslanja na provjeru integriteta podataka od strane TCP protokola.
Najčešće se koristi za prijenos jedne ili veće količine datoteka u lokalnim mrežama (LAN) ili preko interneta (WAN mreže).
*FTP* također radi prema principu: klijent ←→ poslužitelj. ***FTP*** je prvo definiran u **RFC114** te: **RFC765**, **RFC959**, te proširen sa: **RFC1579** **(***passive mode***)**, **RFC2228** (sigurnosna poboljšanja), **RFC2428** (uvodi podršku za **IPv6** te dodatke na pasivni način rada).
Kod *FTP-a* je specifično i to da koristi dva kanala u komunikaciji: jedan za kontrolni promet i sâmu vezu, a drugi za prijenos podataka. Tijekom spajanja na *FTP* poslužitelj može se raditi i *autentikacija* korisnika prema principu slanja: korisničko ime i lozinka, ali ne kriptirano. Dakle moguće je praćenjem paketa na mreži vidjeti korisničko ime i lozinku u čistom tekstualnom obliku.
Osim toga moguća je i upotreba tzv. "*Anonimous*" korisnika, ako je tako konfigurirano na poslužiteljskoj strani.

Za osiguranje sigurnosti prijenosa podataka često se koristi komunikacija kroz neki *SSL* ili *TLS* kanal koji kriptira promet.
U današnje vrijeme ***FTP*** je zamijenjen sa *SSH File Transfer Protokolom* (***SFTP***), ako je potrebna sigurna razmjena podataka.

**FTP koristi TCP port 20 za podatke (engl.** *Data***).**
**FTP koristi i TCP port 21 za kontrolu (engl.** *Command***).**

Slika 237. Aktivni način rada FTP poslužitelja..




Kako radi prijenos podataka korištenjem *FTP* protokola?

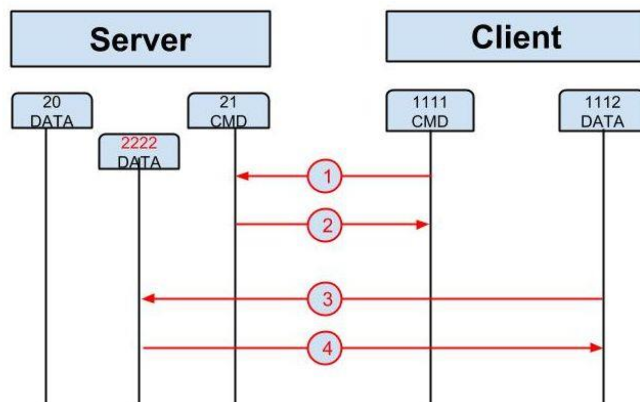
*FTP* može raditi u ***aktivnom*** (engl. *Active*) ili ***pasivnom*** načinu rada (engl. *Passive*) o kojima ovisi kako će se konekcija u konačnici ostvariti.
U oba slučaja klijentska strana kreira TCP kontrolnu vezu (TCP *odredišni port* **21**) s time da je TCP *izvorišni port* odabran standardnom metodom: 1024 + slučajan broj = *izvorišni port* (nazovimo ga **X**).

Na slici je to port broj **1111**.

U ***aktivnom*** načinu rada (slika 237) klijent čeka dolazeću "*Data*" konekciju na portu **X**+1 s poslužitelja i šalje *FTP* poruku poslužitelju da očekuje poruku na portu broj: **X**+1 (na slici je to port broj: **1112**), a tada se ostvaruje nova konekcija (*TCP Three Way Handshake*). Ako je klijent iza nekog vatrozida (*firewall*) uređaja, ovakav način rada nije moguć te se mora koristiti pasivni način rada. U ***pasivnom*** načinu rada (slika 238) klijent koristi kontrolnu konekciju za slanje **`PASV`** naredbe poslužitelju (zahtjeva prebacivanje u pasivni način rada) i dobiva od njega TCP port koji će koristiti kao odredišni (*destination*) port, a poruka koju prima je "`Passive Port : `**`Y`**" tj. **`Y`** je broj porta koji mu poslužitelj dodjeljuje uz njegovu IP adresu (poslužitelja). On dobiva nazad poruku "`Passive IP Address=a.b.c.d`" tj. konkretnu IP adresu poslužitelja.

Sada klijent otvara konekciju na poslužitelj s IP adresom koju mu je on dao (pr. `192.168.100.1`) na odredišni port koji mu je također poslao (port: **Y**), a kao izvorišni (*source*) port postavlja **X**+1. Nakon toga se TCP konekcija uspostavlja standardnom metodom (TCP *Three Way Handshake*). Konekcije se zatvaraju standardno za TCP protokol.

Slika 238. Pasivni način rada.




**Izvori informacija: (718), RFC114, RFC765, RFC959, RFC1579, RFC2228 i RFC2428.**

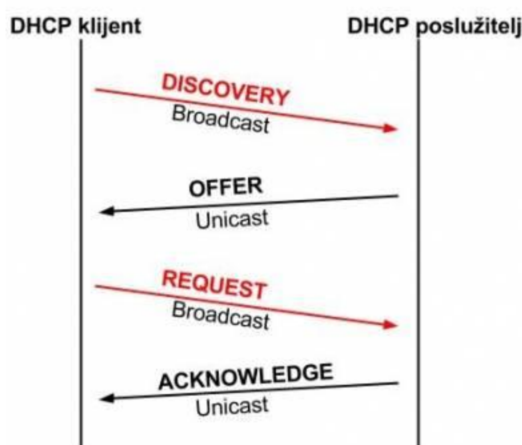
## 25.8.3. DHCP protokol

*Dynamic Host Configuration Protocol* (**DHCP**) je protokol sličan **BOOTP** protokolu, ali s naprednijim mogućnostima.
Dakle *DHCP* protokol podržava punu funkcionalnost *BOOTP* protokola uz dodatne funkcionalnosti koje nudi.
DHCP se koristi za automatsko dodjeljivanje mrežnih IP parametara na strani klijenta, poput IP adrese i maske mreže (*Netmaska*), podrazumijevanog usmjerivača (*Default Gateway-a*), DNS poslužitelja te mnogih drugih parametara mreže.
***DHCP*** protokol funkcionira prema principu: klijent ←→ poslužitelj.

**DHCP (kao i BOOTP) koristi UDP protokol za transport, na portu 67**.
*DHCP* se koristi u svim modernim mrežama, počevši od kućnih mreža, mreža u malim i većim tvrtkama sve do najvećih mreža u najvećim korporacijama i *internet providerima* (*ISP*).

**Kako radi DHCP?**
Kada se računalo pokrene i učita se operativni sustav, *DHCP* klijent na mrežnoj kartici na računalu, šalje na mrežu *DHCP* upit (engl. *DHCP Query*) koji je prema vrsti *Broadcast* poruka, što znači kako ju svi na lokalnoj (LAN) mreži moraju primiti.
Dakle u ovom trenutku naše računalo još uvijek nema jedinstvenu IP adresu niti druge IP parametre mreže.

Slika 239. DHCP komunikacija.




*DHCP* poslužitelj na mreži zaprima ovaj zahtjev, pregledava svoju tablicu s listom dostupnih IP adresa (engl. *Pool*) i nalazi prvu slobodnu (neiskorištenu) IP adresu i pripadajuću joj masku mreže (*Netmask*) te čita ostale konfiguracijske parametre koji su aktivirani, poput primjerice:

- IP adresu podrazumijevanog usmjerivača (*Default Gateway-a).*
- IP adrese ***DNS*** poslužitelja.
- Ime domene (*Domain name*).
- Poslužitelja s točnim vremenom [Time Server (***NTP***)] i drugo.

DHCP poslužitelj može biti konfiguriran i da određeni parovi IP adresa + MAC adresa budu rezervirani. Tako će DHCP klijenti koji imaju rezervaciju, svaki puta dobiti istu IP adresu.
DHCP poslužitelj ima definirano i maksimalno vrijeme čuvanja (rezervacije) (engl. *Lease Time*) koje vrijedi za svaku IP adresu koju dodjeli *DHCP* klijentima. Kada to vrijeme istekne, *DHCP* klijent ponovno traži obnavljanje te iste IP adrese te mu *DHCP* poslužitelj odgovara prema istom principu. *Lease time* je isto jedna od opcija (standardna) za *DHCP*.
Nakon toga *DHCP* poslužitelj šalje *DHCP* klijentu direktno (*unicast*) poruku sa svim parametrima koji su bili aktivirani. Potom *DHCP* klijent konfigurira svoju mrežu na osnovu parametara koje je primio od strane *DHCP* poslužitelja.

**Izvori informacija: (719), RFC903 (RARP), RFC951 (BOOTP), DHCP: RFC1531, RFC1541, RFC2131 i drugi.**

## 25.8.3.1. Oblik DHCP poruke

U ovoj cjelini upoznat ćemo se s detaljima *DHCP* poruka. Svaka *DHCP* poruka sastoji se od sljedećih polja:

| Polje | Opis |
|---|---|
| **Operation** | Opisuje vrstu poruke. |
| **HType** | Vrsta hardvera. |
| **HLen** | Duljina hardverske adrese. |
| **Hops** | *HOP* brojač. |
| **XID** | Identifikator transakcije. |
| **Secs** | Broj sekundi koliko se čeka na DHCP poslužitelj. |
| **Flags** | Zastavice za BOOTP protokol. |
| **CIAddr** | IP adresa klijenta. |
| **YIAddr** | *Your IP adress* - dodijeljena IP adresa (DHCP klijentu). |
| **SIAddr** | IP adresa BOOTP poslužitelja *(Next Server IP).* |
| **GIAddr** | IP adresa DHCP *relay* poslužitelja. |
| **CHAddr** | MAC Adresa DHCP klijenta. |
| **SName** | Ime računala (*Hostname*) DHCP poslužitelja. |
| **File** | BOOTP (PXE/LAN BOOT) datoteka. |
| **Options** | Ovo su DHCP opcije. |

Slijedi opis polja:

- `Operation` može biti:
    - `1` - *request* – dakle ovo je DHCP upit.
    - `2` - *response* – dakle ovo je DHCP odgovor.
- `HType` - vrsta mreže - obično je to `1` (*Ethernet*).
- `HLen` - duljina hardverske adrese - za *ethernet* je to MAC adresa, pa je duljina 6 *bajta*.
- `Hops` - broj *HOP*ova (prolaznih *Layer 3* uređaja)- standardno je `0`. Samo se u slučaju upotrebe *DHCP Relay* uređaja, može povećati (ovo se uglavnom ne mijenja).
- `XID` - identifikacija transakcije (komunikacije DHCP klijent - poslužitelj).
- `Secs` - za DHCP protokol je ovo broj sekundi koliko DHCP klijent čeka na odgovor od poslužitelja.
- `Flags` - zastavice:
    - Ima vrijednost `0` (u većini slučajeva), ako DHCP poslužitelj može vratiti poruku direktno DHCP klijentu dakle kao *unicast*. Ovo polje postavlja DHCP klijent.
    - Ima vrijednost `1` ako DHCP poslužitelj ne može vratiti poruku direktno DHCP klijentu, kao *unicast*, već ju mora slati na cijelu mrežu (kao *broadcast*). I ovo polje postavlja DHCP klijent.

- `CIAddr` - je IP adresa klijenta, koja može biti:
  - `0.0.0.0` - ako DHCP klijent još nema IP adresu, odnosno ako je ovo inicijalni upit.
  - **IP Adresa** - konkretna IP adresa klijenta, koji je već dobio IP adresu, a to je samo u slučajevima kada je DHCP klijent u nekom od sljedećih stanja:
    - `BOUND` - je stanje nakon što je DHCP klijent primio poruku DHCP ACK te postavio svoju IP i ostale parametre koje je dobio.
    - `RENEWING` - je stanje nakon *BOUND* stanja, ako DHCP klijent prima novu DHCP ACK poruku s novim parametrima (ili osvježenim), pa potom mijenja konfiguraciju i potom prelazi ponovno u stanje *BOUND*.
    - `REBINDING` - je stanje kada DHCP klijent šalje *DHCP request* (*broadcast*) te nakon primitka odgovora mijenja svoje parametre i potom ponovno odlazi u stanje *BOUND*.
- `YIAddr` - ovo je IP adresa koju DHCP poslužitelj dodjeljuje DHCP klijentu. Ako je ovo bio DHCP upit od klijenta prema poslužitelju, tada su ovdje nule (`0.0.0.0`).
- `SIAddr` za DHCP ovo je tzv. *Next Server IP*. To znači IP adresa *BOOTP* poslužitelja koji se nudi za *BOOTP*, kao mrežni instalacijski poslužitelj.
- `GIAddr` - ovo je IP adresa DHCP/BOOT *relay* poslužitelja: ova funkcionalnost se uglavnom ne koristi, pa je stoga IP adresa ovdje obično `0.0.0.0`.
- `CHAddr` - ovo je MAC adresa DHCP klijenta odnosno njegove mrežne kartice (mrežnog sučelja s kojeg se šalju DHCP upiti prema DHCP poslužitelju)
- `SName` - ovo je *hostname* DHCP poslužitelja, koje DHCP poslužitelj ne mora poslati (uglavnom i ne šalje).
- `File` - ovo je *PXE/LAN BOOT* - ime datoteke dostupne na ***TFTP*** poslužitelju za pokretanje sustava preko mreže.
- `Options` - ovo su DHCP opcije koje se mogu tražiti i/ili dobiti (objasniti ćemo ih u sljedećem poglavlju).

Objasnit ćemo neke od DHCP opcija (`Options`) koje se često koriste. Naime unutar DHCP poruke, nalazi se i polje s mogućim opcijama. Opcije su definirane u **RFC 2132**, a cijeli popis je vidljiv u izvoru informacija (**1153**). Ove opcije se odnose na parametre koje možemo slati DHCP klijentima. Standardno definiranih opcija postoji nekoliko stotina.
Pogledajmo i listu samo nekih od osnovnih *DHCP* opcija:

| **DHCP opcija** *(broj)* | **Ime opcije** | **Opisano u RFC-u** | **Opis** |
|---|---|---|---|
| **1** | *Subnet Mask* | RFC 2132 | **Ovo je osnovna opcija**: Maska mreže (*subnet mask*) koja se dodjeljuje DHCP klijentu. |
| **2** | *Time Zone Offset* | RFC 2132 | Pomak vremenske zone (u sekundama). |
| **3** | *Gateway* | RFC 2132 | **Ovo je osnovna opcija**: Definira se *Default Gateway* IP adresa, koja se dodjeljuje DHCP klijentu. |
| **4** | *Time Server* | RFC 2132 | *NTP* poslužitelj, vezan uz opciju **2.** |
| **5** | *Name Server* | RFC 2132 | Lista imeničnih poslužitelja (IEN-116), koja se dodjeljuje DHCP klijentu (Ovo **nisu** DNS poslužitelji). |
| **6** | *Domain Name Server* | RFC 2132 | **Ovo je osnovna opcija**: Lista **DNS** poslužitelja, koja se dodjeljuje DHCP klijentu. |
| **7** | *Log Server* | RFC 2132 | **MIT-LCS UDP log** poslužitelj na mreži, koji se dodjeljuje DHCP klijentu. |
| **12** | *Hostname* | RFC 2132 | *Hostname* - ime računala DHCP klijenta. Označava samo ime računala bez domene. |
| **13** | *Boot File Size* | RFC 2132 | U ovoj opciji se definira veličina “boot” datoteke, definirane u blokovima od 512 *bajta*. (Vezano za opciju 67, ali se ne mora ovako koristiti). |
| **15** | *Domain Name* | RFC 2132 | Ime domene, koje će biti dodijeljeno DHCP klijentu. |
| **19** | *Forward On/Off* | RFC 2132 | Ova opcija DHCP klijentu šalje naredbu za uključivanje ili isključivanje IP *Packet forwardinga (Na računalima s više mrežnih kartica, mogućnost usmjeravanja paketa s jedne mrežne kartice na drugu).* |
| **26** | *Interface MTU* | RFC 2132 | Postavljanje ***MTU*** veličine za mrežnu karticu, na kojoj se nalazi DHCP klijent. |
| **28** | *Broadcast Address* | RFC 2132 | *Broadcast* adresa mreže koja se daje DHCP klijentu. |
| **31** | *Perform Router Discovery* | RFC 2132 | Uključivanje mogućnosti DHCP klijentskog računala da pronađe Router/Gateway usmjerivač na mreži. Prema standardu RFC 1256. |
| **33** | *Static Route* | RFC 2132 | Statičke *route* koje se mogu propagirati DHCP klijentima. Može biti zamijenjen s novijim RFC 3442 odnosno **opcijom 121.** |
| **35** | *ARP Timeout* | RFC 2132 | *ARP cache timeout*, koji se može poslati DHCP klijentu. |
| **42** | *NTP Servers* | RFC 2132 | NTP poslužitelj, koji će DHCP klijent koristiti. |
| **44** | *NETBIOS Name Srv* | RFC 2132 | Definicija NETBIOS *Name Servers* (***WINS***) odnosno ***NETBIOS*** poslužitelja. |

| **DHCP opcija *(broj)*** | **Ime opcije** | **Opisano u RFC-u** | **Opis** |
|---|---|---|---|
| **50** | *Address Request* | RFC 2132 | DHCP klijent u poruci *DHCPREQUEST* može zatražiti od DHCP poslužitelja da mu dodjeli specifičnu IP adresu, definiranu u ovom polju. |
| **51** | *Lease Time* | RFC2132 | **Ovo je osnovna opcija** u kojoj se definira vrijeme trajanja rezervacije IP adrese na DHCP poslužitelju. |
| **53** | ***DHCP Message Type*** | RFC 2132 | **Ovo je osnovna opcija, u kojoj se definira vrsta DHCP poruke.** |
| **55** | *Parameter List* | RFC 2132 | Lista parametara koje DHCP klijent može zatražiti od DHCP poslužitelja.<br>Primjerice: DHCP klijent traži opcije: 72 i 252. |
| **58** | *Renewal Time* | RFC 2132 | **Ovo je osnovna opcija (T1** *vrijeme***)** u kojoj bi DHCP klijent unutar ovog vremena trebao osvježiti svoju IP adresu, od DHCP poslužitelja. Ovo je vrijeme nakon kojega DHCP klijent ulazi u stanje *RENEWING*.<br>Ovo vrijeme je obično oko 50 posto vremena od *Lease time* (opcije 51). |
| **59** | *Rebinding Time* | RFC 2132 | **Ovo je osnovna opcija (T2** *vrijeme***)** u kojoj DHCP klijent unutar ovog vremena mora osvježiti svoju IP adresu, od bilo kojeg DHCP poslužitelja. Ovo je vrijeme nakon kojega DHCP klijent ulazi u stanje *REBINDING*.<br>Ovo vrijeme je obično minimalno 87.5 posto vremena od *Lease time* (opcije 51). |
| **66** | *TFTP Server name* | RFC 2132 | IP adresa ili ime TFTP poslužitelja namijenjenog za pokretanje ili instalaciju sustava preko mreže. **Pogledajte poglavlje: 25.8.4 Bootp protokol.** |
| **67** | *Filename* | RFC 2132 | Ime datoteke na ***TFTP*** poslužitelju (iz opcije **66**), za pokretanje ili instalaciju sustava preko mreže. Ovdje se definira ***Boot Loader*** datoteka.<br>**Pogledajte poglavlje: 25.8.4 Bootp protokol.** |
| **69** | *SMTP-Server* | RFC 2132 | U ovoj opciji se specificira lista SMTP poslužitelja, dostupnih za DHCP klijenta. |
| **70** | *POP3-Server* | RFC 2132 | U ovoj opciji se specificira lista POP3 poslužitelja, dostupnih za DHCP klijenta. |
| **72** | *WWW-Server* | RFC 2132 | U ovoj opciji se specificira lista WWW poslužitelja, dostupnih za DHCP klijenta. |
| **119** | *Domain Search* | RFC 3397 | ***DNS suffix search lista*** u kojoj se navode domene koje će se automatski nadodati na bilo koje ime računala *hostname* (bez domene) kojem želimo pristupiti.<br>Istovjetna opcija je opisana u poglavlju:<br>**25.6.2. Konfiguracijska datoteka /etc/resolv.conf.** |
| **121** | *Classless Static Route Option* | RFC 3442 | Koristi se za dodavanje statičkih *ruta* na DHCP klijenta.<br>Ova opcija je novija i može mijenjati opciju **33**. |
| **252** | *WPAD* | RFC 2132 | U ovoj opciji se definira poslužitelj i URL na kojemu se nalazi *Web Proxy Automatic Discovery (WPAD)* datoteka, za automatsku konfiguraciju *proxy* poslužitelja, za web preglednike na DHCP klijentu/računalu.<br>**Pogledajte poglavlje: 26.4.5. Proxy poslužitelji.** |

Unutar **DHCP opcije 53** (***DHCP Message Type***), koja je osnovna opcija, definira se vrsta ***DHCP*** poruke, a koja može biti:

| **Broj poruke** | **DHCP vrsta poruke (Opcija 53)** | **Opis** |
|---|---|---|
| **1** | **DHCPDISCOVER** | DHCP klijent traži dostupni DHCP poslužitelj (*Broadcast*). |
| **2** | **DHCPOFFER** | DHCP poslužitelj odgovara na upit DHCP klijenta (*Unicast*). |
| **3** | **DHCPREQUEST** | DHCP klijent potvrđuje DHCP poslužitelju, i traži ponuđene opcije (*Broadcast*). |
| **4** | **DHCPDECLINE** | DHCP klijent šalje poruku DHCP poslužitelju kako je IP adresa koju je dobio, već u upotrebi negdje na mreži. Tada DHCP poslužitelj mora ovu IP adresu označiti kao nedostupnu. |
| **5** | **DHCPACK** | DHCP poslužitelj odgovara na upit DHCP klijenta i daje mu tražene opcije (*Unicast*). |
| **6** | **DHCPNAK** | DHCP poslužitelj odgovara na upit koji je dobio, s negativnim odgovorom odnosno odbija DHCP zahtjev od DHCP klijenta. |
| **7** | **DHCPRELEASE** | DHCP klijent šalje poruku DHCP poslužitelju, da "oslobađa" IP adresu te ju DHCP poslužitelj označava kao slobodnu (za novu/drugu upotrebu). |
| **8** | **DHCPINFORM** | DHCP klijent traži od DHCP poslužitelja neke od DHCP opcija (parametara).<br>DHCP poslužitelj na ovu poruku odgovara s DHCPACK. |



Mi smo naveli sâmo one osnovne poruke iako postoji i nadogradnja na navedene poruke.

**Izvori informacija: (719),(1153), RFC 2132.**

## 25.8.3.2. DHCP - kako to izgleda u praksi

Na sljedećem primjeru, vidjet ćemo kako teče komunikacija između DHCP poslužitelja i klijenta. Zamislimo sljedeću mrežu:

- MAC adresa DHCP klijenta (mrežne kartice računala koje kontaktira DHCP poslužitelj) je: ***08:00:27:ec:c0:1a***
- IP adresa DHCP poslužitelja je: ***192.168.101.1***.
- *Broadcast MAC* adresa cijele (**LAN**) mreže je: ***ff:ff:ff:ff:ff:ff***.
- *Broadcast IP* adresa cijele (**LAN**) mreže je: ***255.255.255.255***.
- IP adresa koju će naš DHCP klijent dobiti je: ***192.168.101.166***.

Mrežne pakete smo snimali s programom Wireshark, a isto smo mogli napraviti i s programom `tcpdump`.

### 25.8.3.2.1. Poruka *DHCP Discover*

Prva poruka (mrežni paket) koju šalje *DHCP* klijent je ***DHCP Discover*** poruka, stoga pogledajmo kako ona izgleda.
**1**. *DHCP* klijent šalje `DHCP Discover` upit na cijelu lokalnu (*LAN*) mrežu u nadi kako će mu barem jedan *DHCP* poslužitelj odgovoriti. Pogledajmo dolje kako izgleda `DHCP Discover` upit. Sa slovima **A-D** smo označili OSI slojeve: **1-4** (**A,B,C,D**).

```
A. Frame 5: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits)
B. Ethernet II, Src: 08:00:27:ec:c0:1a, Dst: ff:ff:ff:ff:ff:ff
C. Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
D. User Datagram Protocol, Src Port: 68, Dst Port: 67
    Bootstrap Protocol (Discover)
      Message type: Boot Request (1)
      Hardware type: Ethernet (0x01)
      Hardware address length: 6          Hops: 0
     Bootp flags: 0x0000 (Unicast)
     Client IP address: 0.0.0.0
     Your (client) IP address: 0.0.0.0
     Next server IP address: 0.0.0.0
     Relay agent IP address: 0.0.0.0
     Client MAC address: 08:00:27:ec:c0:1a
     Client hardware address padding: 00000000000000000000
     Server host name not given
     Boot file name not given
     Magic cookie: DHCP
     Option: (53) DHCP Message Type (Discover)
     Option: (50) Requested IP Address
     Option: (55) Parameter Request List                      Length: 13
        Parameter Request List Item: (1) Subnet Mask
        Parameter Request List Item: (28) Broadcast Address
        Parameter Request List Item: (2) Time Offset
        Parameter Request List Item: (121) Classless Static Route
        Parameter Request List Item: (15) Domain Name
        Parameter Request List Item: (6) Domain Name Server
        Parameter Request List Item: (12) Host Name
        Parameter Request List Item: (40) Network Information Service Domain
        Parameter Request List Item: (41) Network Information Service Servers
        Parameter Request List Item: (42) Network Time Protocol Servers
        Parameter Request List Item: (26) Interface MTU
        Parameter Request List Item: (119) Domain Search
        Parameter Request List Item: (3) Router
     Option: (255) End
```

**Što sve vidimo na ispisu:**

**A.** Ovo je **OSI sloj jedan (OSI 1)** odnosno ono što se šalje "na žicu".

**B.** Na **OSI sloju dva (OSI 2)** se formira mrežni okvir čija je izvorišna MAC adresa, MAC adresa mrežne kartice s koje se šalje upit (`Src:` ***08:00:27:ec:c0:1a***). Odredišna MAC adresa `Dst:` je ***ff:ff:ff:ff:ff:ff*** što znači kako će ovaj mrežni okvir biti poslan na cijelu lokalnu mrežu odnosno kako će ga svi uređaji na mreži **morati** zaprimiti. Naime kada ovakav mrežni okvir dođe do preklopnika (*switcha*) i on pregleda njegovu odredišnu MAC adresu, a koja je ovdje ***ff:ff:ff:ff:ff:ff***, preklopnik će odmah taj mrežni okvir proslijediti na sva svoja mrežna sučelja (*portove/interface*). Zbog toga će ovaj mrežni okvir doći do svih uređaja na mreži.

**C.** Na **OSI sloju tri (OSI 3)** je vidljivo kako je izvorišna IP adresa nepoznata `Src:` ***0.0.0.0*** jer ju zapravo i tražimo, a odredišna IP adresa `Dst:` je ***255.255.255.255*** što označava također *Broadcast* poruku (poruku na sve). To znači kako će ovu poruku prihvatiti i svi uređaji koji rade na OSI sloju tri *(OSI 3)*. Dakle svi usmjerivači (*routeri*), *multilayer* preklopnici i vatrozidi (*firewalli*). Potom sve ovisi o njihovoj konfiguraciji hoće li ju proslijediti negdje drugdje ili odbaciti.

**D**. Na **OSI sloju četiri (OSI 4)** vidimo kako se za transport koristi *UDP* protokol prema odredišnom portu `Dst Port`**:** ***67*** što definira DHCP servis. Izvorišni port `Src Port` je uvijek ***68***. ***To inače NIJE PRAVILO, već je UVIJEK izvorišni port nasumično odabrani port, za većinu mrežnih protokola***. Na još višem sloju je vidljivo kako se radi o **Bootstrap Protocol** odnosno *DHCP* protokolu. U konkretnom slučaju vidljivo je kako je vrsta poruke (*Message type:*) *Boot Request* (**1**).
Ovo je polje ***Option***, gdje broj **1** označava poruku vrste: ***Request***.

- Ovdje je još zanimljivo polje ***Transaction ID*** u kojemu je naveden broj transakcije *(0xd0798009)* koja označava komunikaciju između ovog (konkretnog) DHCP klijenta i DHCP poslužitelja.

Potom slijedi opcija (**53**) unutar koje se definira o kojoj se točno DHCP poruci radi. U našem slučaju je to broj **1** koji govori kako je poruka tipa **DHCPDISCOVER**. Potom slijede i druge opcije:

- *Opcija 50* – znači kako tražimo IP adresu od DHCP poslužitelja.
- *Opcija 55* – znači kako tražimo cijeli niz opcija koje nam trebaju odnosno koje naš DHCP klijent/ operativni sustav traži. U našem slučaju su to opcije:
  - (1) *Subnet Maska odnosno maska mreže i* (28) *Broadcast* adresa.
  - (2) *Time Offset* - pomak vremenske zone.
  - (121) *Classless Static Route* - nove *route*, za našu tablicu usmjeravanja (*routing tablicu*), ako ih ima.
  - (15) *Domain Name* - ime domene za naše računalo.
  - (6) *Domain Name Server* - lista DNS poslužitelja u našoj mreži.
  - (12) *Host Name* - ime računala bez domenskog nastavka u imenu.
  - (40) i (41) su za NIS poslužitelje.
  - (42) *Network Time Protocol Servers* - NTP poslužitelji na mreži (za točno vrijeme).
  - (26) *Interface MTU* - MTU (*Maximum Transmission Unit*) postavke za naše mrežno sučelje (mrežnu karticu).
  - (119) *Domain Search* - listu domena za pretraživanje (pogledajte opis u tablici opcija).
  - (3) *Router* - je IP adresa za podrazumijevani usmjerivač (*Default Gateway*) preko kojeg će naše računalo "izlaziti" na druge mreže poput interneta.

Važno je znati kako DHCP klijent traži opcije koje uglavnom ovise o operativnom sustavu, ali se i one mogu mijenjati.
U našem konkretnom primjeru DHCP klijent je Linux računalo.

---

### 25.8.3.2.2. Poruka *DHCP Offer*

Ovo je druga poruka u DHCP komunikaciji. Dakle dobivamo DHCP paket broj dva u kojem je DHCP poslužitelj zaprimio naš upit te nam šalje ovaj odgovor (***DHCP Offer***). *Sa slovima **A-D** smo ponovno označili OSI slojeve: **1-4 (A,B,C,D)**.* Pogledajmo:

```
A. Frame 2: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits)
B. Ethernet II, Src: 00:08:60:00:79:20, Dst: 08:00:27:ec:c0:1a
C. Internet Protocol Version 4, Src: 192.168.101.1, Dst: 192.168.101.166
D. User Datagram Protocol, Src Port: 67, Dst Port: 68
    Bootstrap Protocol (Offer)
     Message type: Boot Reply (2)
     Hardware type: Ethernet (0x01)
     Hardware address length: 6            Hops: 0
     Transaction ID: 0xd0798009
     Seconds elapsed: 0
     Bootp flags: 0x0000 (Unicast)
     Client IP address: 0.0.0.0
     Your (client) IP address: 192.168.101.166
     Next server IP address: 192.168.101.1
     Relay agent IP address: 0.0.0.0
     Client MAC address: 08:00:27:ec:c0:1a
     Client hardware address padding: 00000000000000000000
     Server host name not given
     Boot file name: pxelinux/pxelinux.0
     Magic cookie: DHCP
     Option: (53) DHCP Message Type (Offer)
     Option: (54) DHCP Server Identifier
     Option: (51) IP Address Lease Time
     Option: (58) Renewal Time Value
     Option: (59) Rebinding Time Value
     Option: (1) Subnet Mask
     Option: (28) Broadcast Address
     Option: (6)  Domain Name Server
     Option: (15) Domain Name
     Option: (42) Network Time Protocol Servers
     Option: (3)  Router
     Option: (255)End
```

Na prethodnom ispisu, vidljivo je sljedeće:

- Na **OSI sloju dva (OSI 2)** - (***B.***) dio, izvorišna MAC adresa je MAC adresa DHCP poslužitelja što je normalno, a odredišna MAC adresa je adresa našeg Linux računala (DHCP klijenta) koje prima ovu poruku.
- Na **OSI sloju tri (OSI 3)** - (***C.***) dio, izvorišna IP adresa je IP adresa DHCP poslužitelja, a odredišna je nova IP adresa koju nam DHCP poslužitelj upravo nudi. Dakle mi još nismo prihvatili ovu ponuđenu IP adresu (u ovom koraku).
- Na **OSI sloju četiri (OSI 4)** (*UDP*) - (***D.***) dio, izvorišni UDP port (*Src:*) je **67** jer je ovo DHCP poslužitelj, a port je za DHCP protokol. Odredišni *UDP* port je **68**, jer je ovo odgovor klijentu, čiji je to izvorišni (*source*) port.

Pošto naše računalo (DHCP klijent) u ovom trenutku još nema IP adresu koja je definirana kao odredišna, ona nam još ništa ne znači. Važno je razumjeti kako prvi uređaj na mreži, a to je preklopnik (*switch*) kada zaprimi ovaj mrežni okvir, on će gledajući *OSI sloj dva (OSI 2)* i odredišnu MAC adresu u njemu, znati dostaviti mrežni okvir na pravo mjesto odnosno na naše računalo. Stoga jer on zna gdje se ta MAC adresa nalazi (na kojem njegovom mrežnom sučelju), prema svojoj ARP odnosno *CAM* tablici.

Dalje u ***BOOTP/DHCP*** dijelu, potvrđuje se transakcijski broj (*Transaction ID: 0xd0798009*) koji je identifikator ove komunikacije odnosno ove sesije.

- `Bootp flags` - govori kako je vrsta komunikacije *unicast* (što nam je jasno jer odgovor ide direktno na MAC adresu DHCP klijenta). Inače ova zastavica i u konačnici sâm odgovor može biti i *broadcast* u posebnim slučajevima (kada se koristi *BOOTP/DHCP relay agent*).
- `Client IP address`: ***0.0.0.0*** - znači kako još nije postavljena IP adresa DHCP klijentu.
- `Your (client) IP address`: ***192.168.101.166*** - ovo je IP adresa koju DHCP poslužitelj nudi DHCP klijentu.
- `Next server IP address`: ***192.168.101.1*** – IP adresa TFTP poslužitelja, na kojemu se nalazi ***Boot loader*** za potrebe ***PXE Boota***- ako se uopće ***PXE*** koristi na mreži, a koja ne mora biti nužno i IP adresa DHCP poslužitelja.
- `Relay agent IP address`: ***0.0.0.0*** - IP adresa *BOOTP/DHCP Relay agenta*, koji mi ovdje ne koristimo (obično se i ne koristi) pa je ova IP adresa ovdje 0.0.0.0 (nije definirana).
- `Client MAC address`: ***08:00:27:ec:c0:1a*** j- ovo e naša MAC adresa (od DHCP klijenta).
- `Boot file name`: *pxelinux/pxelinux.0* - ovo je ***PXE Boot: boot loader*** datoteka na TFTP poslužitelju.
  Ova funkcionalnost nije nužna za normalan rad DHCP: klijent - poslužitelj.

Potom slijede DHCP opcije koje nam nudi DHCP poslužitelj, a koje nisu nužno sve one koje smo mi kao DHCP klijent zatražili u *DHCP Discovery* upitu, već su sve one koje DHCP poslužitelj može (i želi) ponuditi nama.
Među ovim opcijama su one koje moraju biti poslane, a to su redom:

- *`(53) DHCP Message Type (Offer)`* - vidi se kako je ovo poruka *DHCP offer (ponuda).*
- *`(51) IP Address Lease Time`* - vrijeme je u sekundama, koliko će naša IP adresa biti aktivna i rezervirana kao aktivna na DHCP poslužitelju. Nakon isteka ovog vremena DHCP klijent mora osvježiti IP adresu od strane DHCP poslužitelja.
- *`(58) Renewal Time Value`* - vrijeme je u sekundama unutar kojeg DHCP klijent treba osvježiti svoju IP adresu, od DHCP poslužitelja. Ovo je vrijeme nakon kojega DHCP klijent ulazi u stanje *RENEWING*.
  Ovo vrijeme je obično oko 50 posto vremena od *Lease time* (opcije `51`).
- *`(59) Rebinding Time Value`* - vrijeme u sekundama unutra kojeg DHCP klijent **mora** osvježiti svoju IP adresu, od bilo kojeg DHCP poslužitelja. Ovo je vrijeme nakon kojega DHCP klijent ulazi u stanje *REBINDING*. Ovo vrijeme je obično minimalno 87.5 posto vremena od *Lease time* (opcije `51`).

Ostale opcije koje su važne za DHCP klijenta, a koje moramo dobiti su:

- *`(1) Subnet Mask`* - maska mreže koju dobivamo uz pripadajuću IP koju smo dobili (`Your (client) IP address`).
- *`(28) Broadcast Address`* - *Broadcast* IP adresa.
- *`(3) Router`* - ovo je IP adresa podrazumijevanog usmjerivača (*Default Gateway-a).*

Zatim slijede i druge opcije koje možemo dobiti od DHCP poslužitelja, poput ovih koje smo dobili:

- *`(6) Domain Name Server`* - listu DHCP poslužitelja koje možemo koristiti.
- *`(15) Domain Name`* - ime domene.
- *`(42) Network Time Protocol Servers`* - NTP poslužitelj koji možemo koristiti za dohvaćanje točnog vremena.

**Izvori informacija: (719),(1153), RFC 2131, RFC 2132, RFC 3442, RFC 3942, RFC 4361, RFC 4833, RFC 5494.**

### 25.8.3.2.3. Poruka *DHCP request*

Ovo je treća DHCP poruka u kojoj sada slijedi naš odgovor (od DHCP klijenta), koji se zove **`DHCP Request`**. Dakle mi (DHCP klijent) sada na cijelu mrežu šaljemo konačan DHCP upit koji je zapravo upućen prema DHCP poslužitelju:

```
A. Frame 3: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits)
B. Ethernet II, Src: 08:00:27:ec:c0:1a, Dst: ff:ff:ff:ff:ff:ff
C. Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
D. User Datagram Protocol, Src Port: 68, Dst Port: 67
     Bootstrap Protocol (Request)
      Message type: Boot Request (1)
      Hardware type: Ethernet (0x01)
      Hardware address length: 6
      Hops: 0
      Transaction ID: 0xd0798009
      Seconds elapsed: 0
      Bootp flags: 0x0000 (Unicast)
      Client IP address: 0.0.0.0
      Your (client) IP address: 0.0.0.0
      Next server IP address: 0.0.0.0
      Relay agent IP address: 0.0.0.0
      Client MAC address: 08:00:27:ec:c0:1a
      Client hardware address padding: 00000000000000000000
      Server host name not given
      Boot file name not given
      Magic cookie: DHCP
      Option: (53) DHCP Message Type (Request)
      Option: (54) DHCP Server Identifier
        Length: 4
        DHCP Server Identifier: 192.168.101.1
      Option: (50) Requested IP Address
      Option: (55) Parameter Request List
        Length: 13
        Parameter Request List Item: (1) Subnet Mask
        Parameter Request List Item: (28) Broadcast Address
        Parameter Request List Item: (2) Time Offset
        Parameter Request List Item: (121) Classless Static Route
        Parameter Request List Item: (15) Domain Name
        Parameter Request List Item: (6) Domain Name Server
        Parameter Request List Item: (12) Host Name
        Parameter Request List Item: (40) Network Information Service Domain
        Parameter Request List Item: (41) Network Information Service Servers
        Parameter Request List Item: (42) Network Time Protocol Servers
        Parameter Request List Item: (26) Interface MTU
        Parameter Request List Item: (119) Domain Search
        Parameter Request List Item: (3) Router
     Option: (255) End
```

Vidljivo je:

- Na **OSI sloju dva (OSI 2)** (***B***.) vidimo kako naš DHCP klijent s našeg mrežnog sučelja (*Src MAC: **08:00:27:ec:c0:1a***) šalje upit na *broadcast MAC* adresu (*Dst MAC: **ff:ff:ff:ff:ff:ff***) pa stoga ovaj upit vide svi na lokalnoj mreži, pa tako i DHCP poslužitelj s kojim smo razmjenjivali poruke u prethodnim koracima. Naime ovu poruku mrežni preklopnik (*switch*) prosljeđuje svima na mreži zbog njegove odredišne MAC adrese ***ff:ff:ff:ff:ff:ff***.
- Na **OSI sloju tri (OSI 3)** (***C***.) vidimo kako je odredišna IP adresa, zapravo *broadcast* IP adresa. Ovo znači kako će ovu poruku, prihvatiti i svi uređaji koji rade na OSI sloju tri (OSI 3). Dakle prihvatit će ju svi usmjerivači (*routeri*), *multilayer* preklopnici i vatrozidi (*firewalli*), a dalje ovisi o njihovoj konfiguraciji hoće li će ju proslijediti negdje drugdje, odnosno je li im opcija *DHCP Relay* uključena (a obično **nije**).
- Na **OSI sloju četiri (OSI 4)** (***D***.) se koristi isto UDP protokol za transport, a potom slijedi poruka višeg sloja (*BOOTP/DHCP*).

Na višem sloju (*BOOTP/DHCP*) je vidljivo kako se radi o vrsti poruke: `Message type: Boot` **Request (1)** koji prati oznaka naše transakcije, koja je ostala ista (jer označava ovaj komunikacijski kanal između nas i DHCP poslužitelja).
Polje `Bootp flags` se postavlja u `0x0000 (Unicast)` što znači kako se DHCP poslužitelju naznačava kako se očekuje odgovor kao *unicast* poruka.
Nadalje u poruci `Client MAC address: 08:00:27:ec:c0:1a` šaljemo našu MAC adresu.
Potom u opciji `54 DHCP Server Identifier` šaljemo podatak od kojeg DHCP poslužitelja očekujemo odgovor, a u našem slučaju od onog s kojim smo i razmjenjivali poruke dakle onoga na IP adresi: ***192.168.101.1***. Naime u lokalnoj mreži (***LAN***) može biti i više DHCP poslužitelja koji su nam do sada mogli odgovoriti, pa je ovdje važno navesti od kojega od njih tražimo povratnu poruku. Dalje u opciji `(55) Parameter Request List` ponovno tražimo razne DHCP opcije, koje nam trebaju od DHCP poslužitelja.

**Izvori informacija: (719), RFC 2131, RFC 2132.**

#### 25.8.3.2.4. Poruka *DHCP ACK*

U ovom konačnom četvrtom koraku DHCP poslužitelj nam odgovara s porukom: `DHCP ACK` (DHCP *Acknowledgement*) u kojoj dobivamo konačan odgovor za konfiguraciju IP parametara naše mrežne kartice (i računala):

```
A. Fame 4: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits)
B. Ethernet II, Src: 00:08:60:00:79:20, Dst: 08:00:27:ec:c0:1a
C. Internet Protocol Version 4, Src: 192.168.101.1, Dst: 192.168.101.166
D. User Datagram Protocol, Src Port: 67, Dst Port: 68
    Bootstrap Protocol (ACK)
     Message type: Boot Reply (2)
     Hardware type: Ethernet (0x01)
     Hardware address length: 6
     Hops: 0
     Transaction ID: 0xd0798009
     Seconds elapsed: 0
     Bootp flags: 0x0000 (Unicast)
     Client IP address: 0.0.0.0
     Your (client) IP address: 192.168.101.166
     Next server IP address: 192.168.101.1
     Relay agent IP address: 0.0.0.0
     Client MAC address: 08:00:27:ec:c0:1a
     Client hardware address padding: 00000000000000000000
     Server host name not given
     Boot file name: pxelinux/pxelinux.0

Magic cookie: DHCP
      Option: (53) DHCP Message Type (ACK)
      Option: (54) DHCP Server Identifier
         Length: 4
         DHCP Server Identifier: 192.168.101.1
      Option: (51) IP Address Lease Time
      Option: (58) Renewal Time Value
      Option: (59) Rebinding Time Value
      Option: (1)  Subnet Mask
      Option: (28) Broadcast Address
      Option: (6)  Domain Name Server
      Option: (15) Domain Name
      Option: (42) Network Time Protocol Servers
      Option: (3)  Router
      Option: (255) End
```

Ovdje je vidljivo sljedeće:

- Na **OSI sloju dva (OSI 2)** (**B.**) je mrežni okvir koji DHCP poslužitelj sa svoje MAC adrese (*Src: **00:08:60:00:79:20***) šalje direktno na našu MAC adresu (*Dst: **08:00:27:ec:c0:1a***).
- Na **OSI sloju tri (OSI 3)** (**C.**) je mrežni okvir u kojemu je također vidljivo kako DHCP poslužitelj sa svoje IP adrese (*Src: **192.168.101.1***) mrežni okvir šalje na našu IP adresu (*Src: **192.168.101.1***) iako ju još nemamo jer ćemo ju dobiti u ovom koraku, ali to nije važno jer će nam ovaj cijeli mrežni okvir doći preko preklopnika, koji gleda samo OSI sloj dva (OSI 2). Naša odredišna IP adresa je ovdje važna samo u slučajevima dolaska preko *DHCP Relay* uređaja.
- U višim slojevima (***BOOTP/DHCP***) vidimo kako se prati isti *Transaction ID* te kako je vrsta poruke *ACK* (dekodirano iz opcije `53`).
  - Osim toga konačno dobivamo i svoju IP adresu (*Your (client) IP address: **192.168.101.166***).
  - Vidljiva je i naša MAC adresa (*Client MAC address: **08:00:27:ec:c0:1a***).
  - Dodatno vidimo tko nam šalje poruku - iz opcije `54`.
  - Potom slijede sve opcije koje moramo dobiti (`51`,`58`,`59`,`1` i `3`) te ostale opcije koje smo tražili a DHCP poslužitelj nam može dati.

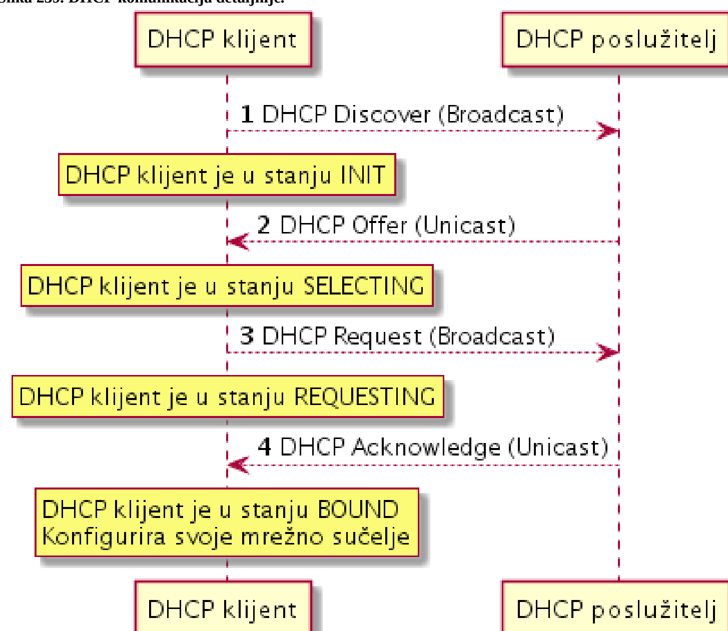
Ovim procesom se završava komunikacija te se na naše mrežno sučelje (mrežnu karticu) i operativni sustav DHCP klijenta dodjeljuju IP parametri koje smo dobili od DHCP poslužitelja.

**Izvori informacija: (719), RFC 2131, RFC 2132.**

#### 25.8.3.2.5. Stanja DHCP klijenta

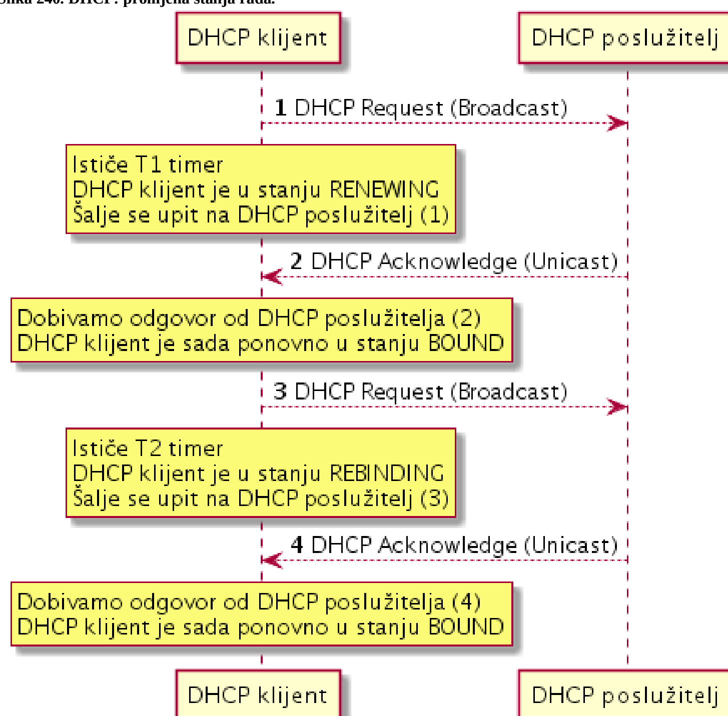
DHCP klijent u komunikaciji s DHCP poslužiteljem, prolazi kroz nekoliko stanja rada. Pogledajmo koja su to stanja i što se događa u kojem od njih, kako je prikazano na slici 239.

**Slika 239. DHCP komunikacija detaljnije.**



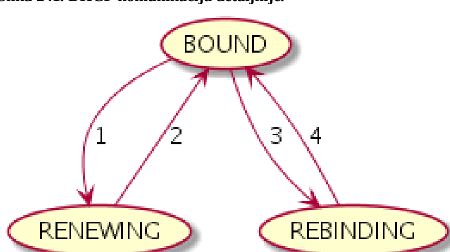

Promjena stanja, kod isteka vremenskih ograničenja (*timera*), prikazana je na slici 240.

**Slika 240. DHCP: promjena stanja rada.**




Proces osvježavanja i prelaske iz jednog stanja u drugo, možemo gledati i ovako: kako je prikazano na slici 241.

**Slika 241. DHCP komunikacija detaljnije.**



### 25.8.3.3. Konfiguracija DHCP poslužitelja

U ovoj cjelini objasnit ćemo osnove instalacije i konfiguracije DHCP poslužitelja, kojega smo koristili za DHCP klijente u prethodnim cjelinama.

Na većini distribucija Linuxa postoji nekoliko DHCP poslužitelja, a mi ćemo objasniti dva najčešće korištena:

- **Internet Systems Consortium DHCP Server.**
- **Dnsmasq.**

## Internet Systems Consortium DHCP Server (dhcpd)

**ISC DHCP Server** jedan je od najstarijih DHCP poslužitelja (razvijen 1999. godine) koji je još uvijek u upotrebi, iako službena podrška za njega ističe krajem 2022. godine. Njega možemo instalirati na sljedeći način:

```
yum -y install dhcp
```

Osnovna konfiguracijska datoteka DHCP poslužitelja je `/etc/dhcp/dhcpd.conf`.

Konfiguracijska datoteka koja se prva čita kod inicijalizacije ovog servisa je datoteka `/etc/sysconfig/dhcpd`.
U njoj se u slučaju kada želimo da se ovaj servis pokreće samo na željenom mrežnom sučelju (pr. `eth1`) treba dodati sljedeća linija odnosno redak konfiguracije:

```
DHCPDARGS=eth1
```

Sada možemo krenuti s konfiguracijom uređivanjem datoteke: `/etc/dhcp/dhcpd.conf` dodavanjem sljedećih redaka.
O ovom primjeru konfiguracije, naš DHCP poslužitelj ima IP adresu **192.168.1.2** na mrežnoj kartici **eth1.**

Prvo trebamo dodati osnovni dio konfiguracije u kojemu definiramo bazične parametre koje će dobiti naši DHCP klijenti:

```
option domain-name "moja.domena.org";
option domain-name-servers 192.168.1.1,158.225.15.16,158.225.15.16;
option broadcast-address 192.168.1.255;
option routers 192.168.1.1;
```

Opis opcija koje smo dodali:

- `option domain-name` - definira ime domene koje će DHCP klijenti dobiti.
- `option domain-name-servers` - definira DNS poslužitelje koje će DHCP klijenti dobiti.
- `option broadcast-address` - definira IP *broadcast* adresu mreže koju će DHCP klijenti dobiti.
- `option routers` - definira podrazumijevani usmjerivač (*Default Gateway*) koji će DHCP klijenti dobiti.

Dodat ćemo i sljedeće linije odnosno retke, u kojima definiramo konkretnu mrežu ili podmrežu (*subnet*) kojoj DHCP klijenti pripadaju odnosno iz čijeg opsega IP adresa će im se dodijeliti IP adresa i drugi parametri mreže:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.10 192.168.1.100;
  default-lease-time 10800;
  max-lease-time 43200;
}
```

Pogledajmo opis opcija koje smo dodali ovdje:

- `subnet 192.168.1.0 netmask 255.255.255.0` - definira IP adresu mreže i masku mreže (*netmask*).
- `range X  Y` - definira opseg IP adresa unutar kojih (od `X` do `Y`) će se DHCP klijentima dodjeljivati IP adrese.
- `default-lease-time` - definira vrijeme u sekundama koliko će DHCP klijentima trajati rezervacija IP adrese.
- `max-lease-time` - definira maksimalno vrijeme u sekundama koliko DHCP klijentima može trajati rezervacija IP adrese.

Sve ove opcije zajedno su dovoljne za osnovnu konfiguraciju DHCP klijenata. Za cijeli niz drugih opcija pogledajte upute: **`man dhcpd.conf`**. Sada kada smo navedene konfiguracijske opcije spremili u navedenu datoteku, možemo i pokrenuti DHCP servis odnosno *daemon* i to ovisno o generaciji *Red Hat/CentOS* linuxa:

Za ***RedHat/CentOS* 6.x**:

```
service dhcpd start
```

Odnosno za ***RedHat/CentOS* 7.x/8.x/9.x+**:

```
systemctl start dhcpd
```

Odnosno za trajnu aktivaciju DHCP servisa (kod svakog pokretanja računala), to možemo postići na slijedeći način:
Za ***RedHat/CentOS* 6.x**:

```
chkconfig dhcpd on
```

Odnosno za ***RedHat/CentOS* 7.x/8.x/9+** ćemo to postići sa:

```
systemctl enable dhcpd
```



Za konfiguraciju DHCP klijenata, pogledajte poglavlje:
**25.6.5.2. Dinamička konfiguracija mreže korištenjem DHCP poslužitelja.**

## Dnsmasq servis

**Dnsmasq** osim što je napredni DHCP i BOOTP (PXE) poslužitelj, on je i Domain Name System (DNS) poslužitelj i TFTP poslužitelj. Razvijen je 2001. godine, ali se i dalje aktivno razvija i koristi.
Primjerice ***OpenStack*** sustav za virtualizaciju koristi ga kao zadani DHCP poslužitelj.
Njega možemo instalirati na sljedeći način

```
yum -y install dnsmasq
```

Ako želite instalirati i koristiti ***dnsmasq***, nemojte instalirati prethodni DHCP poslužitelj (**dhcpd**)!
O ovom primjeru konfiguracije, naš DHCP poslužitelj ima IP adresu **192.168.1.2** na mrežnoj kartici **eth1.**

Centralna konfiguracijska datoteka ***dnsmasq-a*** je: `/etc/dnsmasq.conf`. Stoga ćemo urediti ovu datoteku dodavanjem sljedećih redaka na kraj, uz naše opise [**#**] prije svakog retka konfiguracije:

```
# Definicija mrežne kartice na kojoj će se pokrenuti servis
interface=eth1
# Definiramo samo uporabu gornje mrežne kartice (eth1)
bind-interfaces
# Definicija opsega IP adresa koje će dobivati DHCP klijenti te vrijeme rezervacije (12.h.)
dhcp-range=192.168.1.10,192.168.1.100,12h
# Definicija podrazumijevanog usmjerivača (Default Gateway)
dhcp-option=option:router,192.168.1.1
# Definicija DNS poslužitelja koji će biti dodjeljeni DHCP klijentima
dhcp-option=option:dns-server,192.168.1.1,158.225.15.16,158.225.15.16
```

Još neke od korisnih opcija mogu biti sljedeće; primjerice ako želimo da *dnsmasq* bude i TFTP + PXE poslužitelj:

```
# Aktivacija TFTP poslužitelja
enable-tftp
# Definicija TFTP poslužitelja (DHCP opcija 66)
dhcp-option=eth1,66,192.168.1.2
# Definicija BOOT LOADER datoteke na TFTP poslužitelju
dhcp-boot=pxelinux.0
# Definicija početnog direktorija za TFTP poslužitelj
tftp-root=/tftpboot
```

Vezano za ***DHCP* opcije** koje koristimo, moguće je koristiti dvije sintakse poput simboličke:
`dhcp-option=option:router,`**`IP`** ili možemo definirati *DHCP* opciju prema broju opcije (primjerice DHCP opcija **`66`**):
`dhcp-option=`**`INTF`**`,66,`**`IP`**. Pri tome **`IP`** označava IP adresu, a **`INTF`** mrežnu karticu na koju se primjenjuje.

Pogledajmo skraćenu listu nekih od DHCP opcija (koje se šalju DHCP klijentima) i drugih parametara koje možemo koristiti.

| Naziv opcija | Opis |
|---|---|
| **`no-dhcp-interface=eth2`** | Isključivanje DHCP servisa na točno određenim mrežnim sučeljima (pr. **eth2**) |
| **`dhcp-leasefile=/var/lib/misc/dnsmasq.leases`** | Definicija datoteke u koju će se spremati DHCP rezervacije (za DHCP klijente). |
| **`--dhcp-option=option:router,192.168.1.1`** | DHCP opcija: definicija podrazumijevanog usmjerivača na IP adresi 192.168.1.1. |
| **`--dhcp-option=option:ntp-server,192.168.1.4`** | DHCP opcija: definicija NTP poslužitelja na IP adresi 192.168.1.4. |
| **`domain=domena.hr`** | Definicija domene za DHCP klijente. |
| **`domain=domena.hr,192.168.1.0/24`** | Definicija domene za DHCP klijente, samo za mrežu 192.168.1.0/24. |
| **`server=192.168.1.1`** | Definicija vršnog DNS poslužitelja koji će se koristiti. |
| **`dhcp-range=192.168.1.10,192.168.1.100,12h`** | Definicija DHCP opsega IP adresa za DHCP klijente: 192.168.1.10 do 192.168.1.100 uz vrijeme rezervacija na 12. sati. |
| **`dhcp-host=`<br>`01:02:03:04:A5:B6,pero,192.168.1.50,120m`** | Za DHCP, rezerviraj IP adresu 192.168.1.50, za računalo imena **pero**, s MAC dresom: 01:02:03:04:A5:B6, na 120.minuta. |
| **`dhcp-host=hrvoje,192.168.1.70,infinite`** | Za DHCP, za računalo imena **hrvoje** trajno rezerviraj IP adresu: 192.168.1.70. |
| **`dhcp-option=19,0`** | Postavi DHCP opciju 19 (***ip forwarding***) na isključeno (0=OFF). |
| **`dhcp-option=23,10`** | Postavi TTL (time to live) DHCP opciju na 10. |
| **`dhcp-option-force=209,configs/common`** | Postavi DHCP opciju 209 (TFTP) (**RFC 5071**) s definicijom putanje za TFTP/PXE/BOOTP konf. datoteke. |
| **`dhcp-option=252,"\n"`** | Postavi DHCP opciju 252 (WPAD) na prazno (\n). |
| **`dhcp-option=option:domain-`<br>`search,moja.domena.hr,tvoja.domena.hr`** | Postavi DHCP opciju *DNS domain name search* (RFC 3397) na željene domene. |
| **`enable-tftp`** | Za aktiviranje TFTP servisa |
| **`tftp-root=/var/ftpd`** | Definiranje vršnog TFTP direktorija na poslužitelju. |

🡐 U tablici su nabrojane samo neke osnovne opcije, a za druge opcije i parametre pročitajte upute:

```
man dnsmasq
```



***BOOTP/PXE*** *boot* opcije su opisane u izvoru informacija: **(1153)** odnosno definirane u **RFC 5071**.

Sada možemo pokrenuti *dnsmasq* servis (**za Red Hat 7+**):

```
systemctl start dnsmasq
```

I potom ga možemo trajno aktivirati (**za Red Hat 7+**):

```
systemctl enable dnsmasq
```



Za druge primjene servisa **`dnsmasq`** pogledajte poglavlje:
**25.8.4.4. Kickstart metoda automatizirane instalacije operativnog sustava.**

**Izvori informacija: (719),(1153),(1478),** `man dhcpd`, `man dhcpd.conf`, `man dnsmasq`, **RFC 2131**, **RFC 2132**, **RFC 5071.**

## 25.8.4. Bootp protokol

*Bootstrap Protocol* (**BOOTP**) se koristi za dodjeljivanje IP adrese, ali i drugih parametara rada mrežnom uređaju koji je konfiguriran kao *BOOTP/DHCP* klijent (računalu, poslužitelju, itd.) s *Bootp* poslužitelja. **BOOTP** je izvorno opisan u standardu RFC 951. Njegova osnovna, ali i puna funkcionalnost se sada koristi upotrebom **DHCP** (*Dynamic Host Configuration Protocol*) protokola koji standardno podržava i *BOOTP* mogućnosti.

**BOOTP protokol koristi UDP protokol za transport i to na portu 67.**

### Kako radi BOOTP protokol?

S obzirom na činjenicu da je danas u gotovo svaku bolju mrežnu karticu ugrađen **BOOTP** klijent te **TFTP** klijent unutar **BIOS**a sâme mrežne kartice, zbog toga je moguće i pokretanje odnosno instalacija operativnog sustava upotrebom **BOOTP** i **TFTP** protokola, direktno s mrežne kartice. Ova metoda korištenja mrežne kartice za pokretanje i/ili instalaciju operativnog sustava preko mreže se zove i **PXE** (Engl. *Preboot eXecution Environment*) odnosno ***net boot*** ili ***LAN boot*** metoda.

U slučaju kada na mreži imamo mrežni instalacijski poslužitelj i želimo koristiti navedenu ***PXE boot*** metodu za pokretanje ili instalaciju operativnog sustava drugih računala preko mreže, kako je vidljivo na slici **241.1**, događa se sljedeće:

**1.** Naše računalo (pr. ***PC 1***) se uključuje i pokreće se ***PXE Boot*** program s naše mrežne kartice(ako to mrežna kartica podržava).

**2.** *BOOTP* ili u današnje vrijeme *DHCP* klijent ugrađen u **BIOS** mrežne kartice našeg računala **PC 1** traži IP adresu od DHCP/BOOTP poslužitelja (***Net Install*** *server* na slici).

**3.** *DHCP/BOOTP* poslužitelj mu daje IP adresu, te lokaciju ***Boot managera*** kao i IP adresu s putanjom (URL-om) gdje se ***Boot manager*** nalazi, kako bi ga mogao dohvatiti preko mreže.

**4.** Računalo/mrežna kartica našeg računala **PC 1** pokreće *TFTP* klijent koji preko *TFTP* protokola kopira ***Boot Manager*** s navedenog *TFTP* poslužitelja, koji se tada učitava u memoriju. Zatim se preko *TFTP*-a dohvaća konfiguracijska datoteku u kojoj su popisani parametri potrebni za sljedeći korak.

**5.** ***Boot manager*** se učitao u (RAM) memoriju računala (**PC 1**) te pokreće najosnovnije mogućnosti rada kako bi pročitao konfiguracijsku datoteku u kojoj je upisana lokacija kernela operativnog sustava. Kernel treba (mora) imati podršku za neki brži i pouzdaniji protokol za prijenos datoteka preko mreže poput primjerice: *FTP*, *SAMBA/CIFS*, *NFS* ili nekog drugog, a preko kojeg će se s poslužitelja (***Net Install*** *server* na slici) kasnije i kopirati i učitati kernel operativnog sustava.

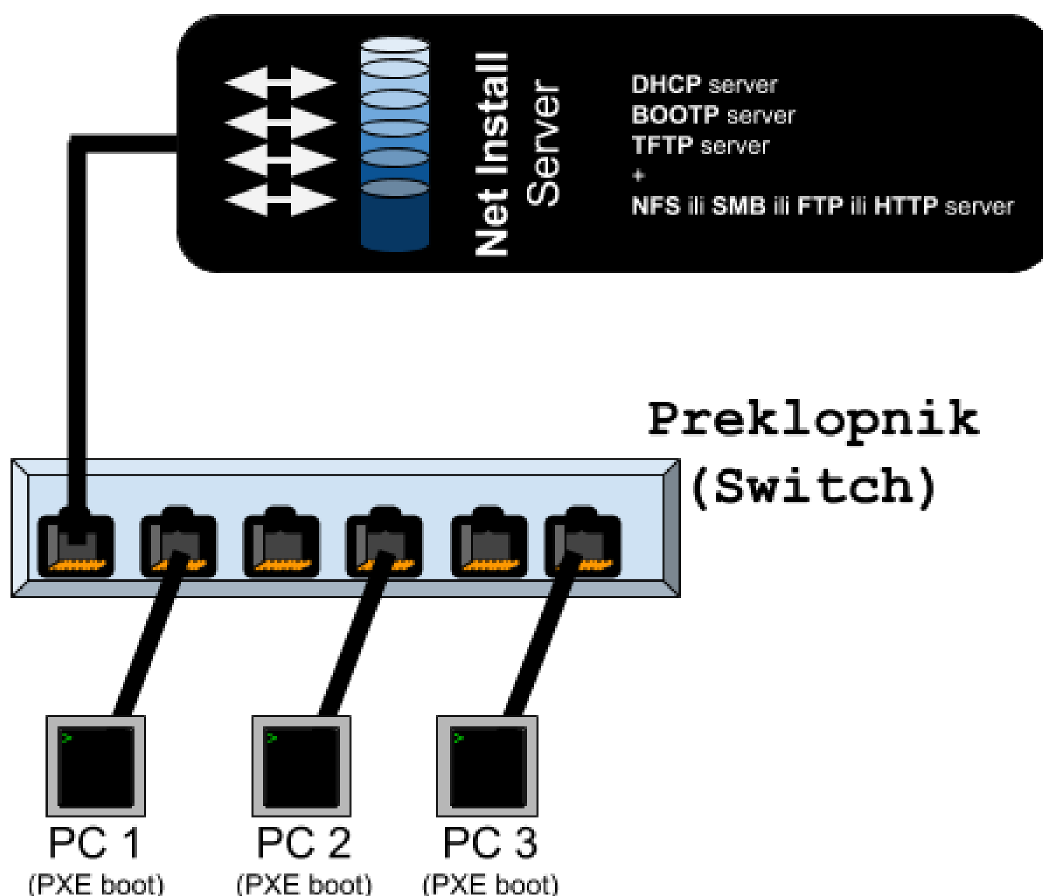
**6.** Pomoću *TFTP*-a se s našeg računala **PC 1** dohvaća kernel operativnog sustava s poslužitelja te se učitava u (RAM) memoriju. Kernel si potom otvara vezu na poslužitelj preko nekog od pouzdanijih protokola za prijenos datoteka (*FTP*, *SAMBA/CIFS*, *NFS*, …) te ovisno o operativnom sustavu koji je pohranjen na poslužitelju, nastavlja s učitavanjem operativnog sustava, kao da se nalazi na nekom lokalnom instalacijskom mediju poput DVD-ROMa, USB diska ili slično.

Dalje se sve odvija kao da je instalacija ili korištenje novog operativnog sustava lokalno s DVD-ROM-a ili lokalnog tvrdog diska.

Ova metoda se koristi i za tzv. "*Diskless*" klijente dakle računala koja nemaju svoj tvrdi disk već cijeli operativni sustav i potrebne programe za rad učitavaju preko mreže, sa ***PXE*** poslužitelja.

Za više detalja o tome što se sve događa tijekom ***PXE*** procedure, pogledajte sljedeća poglavlja, u kojim smo raščlanili ovu komunikaciju u više koraka.

**Slika 241.1** ***PXE boot*** **odnosno** ***network install*** **tj. mrežni** instalacijski poslužitelj.




---

## 25.8.4.1. DHCP dio komunikacije

Sada ćemo proći kroz detalje instalacije operativnog sustava Linux pomoću ***PXE*** metode. U ovom inicijalnom dijelu u kojem je računalo u ***BIOSu*** konfigurirano da pokreće sustav s mrežne kartice (***PXE boot***), događa se sljedeće:

1. Naše računalo (**PC-1**) se pokreće i inicijalizira *PXE boot* proceduru s mrežne kartice.

2. Naše računalo (**PC-1**) prvo učitava *DHCP* klijent (program) s mrežne kartice, koji potom šalje DHCP poruku (***DHCP Discovery***) koja se standardno šalje na cijelu mrežu (to je *BROADCAST* poruka), s kojom tražimo IP i druge parametre.

3. DHCP poslužitelj odgovara s porukom ***DHCP Offer*** i to direktno našem računalu (ovo je *UNICAST* poruka).
Poruka *DHCP Offer* standardno sadrži:

- IP adresu i pripadajuću masku mreže *(Netmask)* te IP adresu podrazumijevanog usmjerivača *(default gateway)* uređaja.
- DNS poslužitelje i ostale opcije, ako ih poslužitelj nudi. U našem slučaju je to važna ***BOOTP*** opcija koja sadrži:
    - IP adresu ***TFTP*** poslužitelja. To je opcija: `next-server` to jest ***DHCP opcija 66***.
    - Ime datoteke (`filename`) to jest ***DHCP opcija 67***, koja zapravo označava ***BOOT LOADER*** datoteku.
      U našem slučaju je to datoteka imena: `pxelinux.0` koja je ***BIOS BOOT LOADER.***

4. DHCP klijent (PC-1) potvrđuje ponudu od DHCP poslužitelja s porukom ***DHCP request*** (ovo je *BROADCAST* poruka) kako bi ju svi na mreži vidjeli, iako je namijenjena sâmo našem DHCP serveru, kojem se potvrđuje poruka ***DHCP offer***. *Broadcast* način slanja je u upotrebi stoga kako drugi DHCP poslužitelji, ako ih ima na mreži, ne bi također rezervirali IP adresu za ovog klijenta odnosno ovo računalo.

5. DHCP poslužitelj tada potvrđuje rezervaciju prema klijentu, s porukom ***DHCP acknowledge*** i šalje vrijeme rezervacije IP adrese (*lease time*) kao i druge opcije, ako ih je DHCP klijent specifično tražio u koraku 4. Naime ako već postoji rezervacija IP adrese za naše računalo odnosno za MAC adresu naše mrežne kartice, rezervira se i koristi upravo ta IP adresa. Ako ne postoji, dobivamo prvu sljedeću slobodnu IP adresu.

6. DHCP klijent (**PC-1**) sada ima IP adresu i sve ostalo potrebno za dalji rad.

---

### 25.8.4.2. Dio sa TFTP klijentom

Vezano za komunikaciju u kojoj sudjeluje **TFTP** klijent, ovdje je stanje sljedeće. Dakle nastavljamo s brojčanim oznakama:

7. DHCP klijent odnosno mrežna kartica našeg računala (**PC 1**) sada pokreće *TFTP* klijent i spaja se na *TFTP* poslužitelj IP adrese koju smo dobili definiranu u opciji `next-server` od strane DHCP poslužitelja u koraku 3. U ovoj opciji se definira IP adresa *TFTP* poslužitelja. Konkretno, ovo je ***DHCP opcija 66***. Potom naš TFTP klijent dohvaća datoteku imena definiranog u opciji `filename` konkretno ovo je ***DHCP opcija 67***.

7.1. *TFTP* klijent (**PC-1**) potom šalje *TFTP* upit na *TFTP* poslužitelj u kojemu traži veličinu definirane datoteke: `pxelinux.0`, a to je ***bootloader***.



Vezano za ***Boot loader*** pogledajte poglavlje:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava.**

7.2. *TFTP* poslužitelj odgovara vraćajući podatak o tome koje je veličine tražena datoteka `pxelinux.0`.

8. *TFTP* klijent sada dohvaća datoteku `pxelinux.0` preko mreže i učitava ju u RAM računala (**PC-1**). Ova učitana datoteka je zapravo Linux BOOT MANAGER (***bootloader***) zadužen za inicijalizaciju sustava i to prije pokretanja sâmog kernela operativnog sustava. Dakle on pokreće inicijalni dio operativnog sustava.

---

### 25.8.4.3. Dio s BOOT MANAGERom i TFTP klijentom - nastavak

Nastavljamo promatrati tijek komunikacije iz prethode cjeline. Sada nam dolaze sljedeći koraci u komunikaciji.

9. U ovom koraku Linux ***BIOS BOOT MANAGER*** (`pxelinux.0`) traži na *TFTP* poslužitelju direktorij imena `pxelinux.cfg` te unutar njega takozvanu ***PXE boot*** konfiguracijsku datoteku. Ime ove datoteke, za svakog **PXE** klijenta (računalo) se traži:

***Napomena: za računala koja ne koriste BIOS već UEFI, koristi se BOOT MANAGER prilagođene UEFI-ju!***

- Prema **MAC** adresi **PXE** klijenta: traži se datoteka imena istog kao njegova **MAC** adresa, ali bez dvotočki u imenu, već s minusima i jedinicom ispred; pr. za **MAC**: *00-30-00-aa-24-48* je pripadajuća datoteka imena: *01-00-30-00-aa-24-48*.
- Ako ne nađe datoteku navedenog imena on (***BOOT MANAGER*** preko *TFTP* klijenta) traži ime datoteke prema svojoj IP adresi, ali pretvoreno u heksadecimalno. Primjerice za IP adresu klijenta: ***10.112.1.1*** je to datoteka imena: *0A700101*
  - Ako nije pronađena niti takva datoteka, briše se jedan heksadecimalni znak s kraja imena datoteke pa to postaje datoteka imena: *0A70010*. Ako i dalje nije pronađena niti datoteka ovakvog imena, briše se još jedan heksadecimalni znak s kraja imena datoteke, pa se traži datoteka imena: *0A7001*. I tako se briše po jedan heksadecimalni znak od kraja imena, sve dok se ne dođe do potrage za datotekom *0* (odnosno sâmo prvog heksadecimalnog znaka od IP adrese).
  - I na kraju, ako nije pronađena niti takva datoteka, traži se datoteka imena: `default` (malim slovima), a koju potom ***BOOT MANAGER*** učitava preko *TFTP* protokola, s *TFTP* poslužitelja.

10. Linux ***BOOT MANAGER*** sada učitava *boot* konfiguracijsku datoteku koju je pronašao u točki **9**.

11. Ova ***PXEboot*** konfiguracijska datoteka sadrži adresu (*URL*) inicijalnog RAM diska i linux kernela, koji se moraju učitati u RAM memoriju, preko *TFTP* protokola. Tek u ovom koraku zapravo se pokreću prvi elementi budućeg operativnog sustava:
U ***PXEboot*** konfiguracijskoj datoteci slijede opcije. Prvo je definirana definicija kernela linuxa. Za kernel bi to konkretno bio redak konfiguracije poput: **`kernel`** `pxeboot/vmlinuz` koji znači sljedeće. Na već definiranom *TFTP* poslužitelju traži se direktorij `pxeboot`, a unutar njega se učitava linux kernel koji je zapravo datoteka imena: `vmlinuz`. Međutim kernel se ne mora nužno nalaziti na *TFTP* poslužitelju jer kernel podržava *HTTP* i *FTP (i druge)* protokole za dohvaćanje datoteka preko mreže. U ovom koraku se učitava i pokreće ovaj linux kernel odnosno datoteka `vmlinuz` s definiranog poslužitelja.

Zatim slijedi: **`append`** `initrd=pxeboot/initrd.img ramdisk_size=10000`.
Ovdje vidimo da se definira učitavanje takozvanog inicijalnog RAM diska s TFTP poslužitelja, odnosno takozvani ***initrd*** (`initrd.img`) s njegovom veličinom (`size=10000`).

*Initrd* sadrži minimalan operativni sustav s osnovnom strukturom direktorija i potrebnih mu datoteka, uz upravljačke programe za diskove (*ATA/SATA/SAS/NVMe*), mrežu i drugo, kao i podršku za mrežne protokole i sustave, poput primjerice: *DHCP, TFTP, NFS, HTTP, FTP* i drugih aplikacijskih protokola. On nakon inicijalizacije kernela i prepoznavanja osnovnog hardvera kreira inicijalni korijen stabla datotečnog sustava koji se *montira* u RAM memoriji i kasnije se na njega montira datotečni sustav. ***Važno je razumjeti da kada su se kernel i initrd učitali, to je prva faza pokretanja potpuno novog operativnog sustava, pa se ponovno traže IP i drugi parametri preko DHCP/BOOTP poslužitelja koji se postavljaju na svoju mrežnu karticu, koja je netom prije prepoznata te su za nju učitani upravljački program. Dakle šalju se klasične DHCP poruke*** *(DHCPDISCOVER, DHCPOFFER, DHCPREQUEST i DHCPACK* ***). Potom se nastavlja s učitavanjem ostatka sustava.***

**Ako se koristi takozvana *kickstart** metoda instalacije operativnog sustava Linux, tada imamo i sljedeći unos:**

```
ks=nfs:192.168.1.1:/opt/install/kickstart/server-1 ksdevice=eno2
```

Navedeni redak konfiguracije znači sljedeće:

- `ks=` znači - ***Kickstart****, a to je datoteka za automatsku instalaciju cijelog operativnog sustava linux koja sadrži upute za instalaciju i konfiguraciju cijelog linuxa koji će se instalirati, ako ju želimo koristiti odnosno imati (njena upotreba nije nužna, ali je korisna). Ova datoteka sadrži upute: od lokacije mrežnog diska s kojeg će se povlačiti instalacija, preko particioniranja diskova, formatiranja particija, instalacije softverskih paketa i svega drugoga. Konkretno mi imamo postavljeno: `ks=nfs:192.168.1.1:/opt/install/kickstart/server-1` – što znači kako se ***kickstart**** datoteka nalazi na ***NFS*** mrežnom dijeljenom disku na adresi 192.168.1.1 unutar direktorija: `/opt/install/kickstart/`, a ime sâme ***kickstart**** datoteke je `server-1`



***Kickstart*** je prema Linux terminologiji procedura automatizirane instalacije operativnog sustava (1149).

**Nakon svake standardne instalacije RedHat kompatibilne distribucije Linuxa, kreira se *kickstart* datoteka: `/root/anaconda-ks.cfg` koja sadrži sve parametre i opcije koje su se koristile tijekom instalacije tog sustava. Nju možete iskoristiti kao inicijalnu *kickstart* datoteku, koju možete prilagoditi za vaše potrebe!**

- `ksdevice=eno2` je mrežna kartica na ovom računalu (PC-1); za *RedHat* Linux v. 6.x bi to bilo primjerice mrežno sučelje imena: `eth0` na koje se instalira linux preko ***kickstart**** procedure. Drugim riječima, ovdje se definira koja mreža kartica će se koristiti za dalju instalaciju Linuxa na računalo **(PC-1)**.

- U nekom drugom slučaju ne bi se morala koristiti opcija `append` ili bi imala drugačiju sintaksu; ovisi kako bi išao tijek instalacije. U primjeru gore je navedena konfiguracija za potpuno automatiziranu proceduru instalacije i linuxa i dodatnog softvera, uz konfiguraciju dodatnog softvera koji se instalira automatski.

**Za instalaciju operativnog sustava, bez automatizirane procedure instalacije i konfiguracije sustava i programa (*kickstart**), zamislimo ovakvu *PXE* konfiguracijsku datoteku**:

```
KERNEL memdisk
INITRD /iso/CentOS-6.9-x86_64-bin-DVD1.iso
APPEND iso raw
```

Pogledajmo opis njenih redaka konfiguracije odnosno njihovo značenje:

- `KERNEL memdisk` - označava kako će se s *TFTP* poslužitelja učitati poseban linux kernel, odnosno datoteka imena `memdisk`, a koja može učitavati i simulirati razne slike odnosno binarne kopije (engl. *Disk image*) instalacijskih medija poput:
    - Slike diskete *(*engl. *boot floppy image)* ili slike tvrdog diska *(*engl. *hard disk image).*
    - Slike ISO medija (CD/DVD) (engl. *ISO image).*

- `INITRD /iso/CentOS-6.9-x86_64-bin-DVD1.iso` - označava kako se unutar *TFTP* poslužitelja unutar vršnog direktorija imena `/iso` nalazi ISO slika (*ISO image*) instalacijskog DVD-a od *CentOS* distribucije linuxa koji će se učitati preko *TFTP* protokola, kao da se radi o lokalno ubačenom DVD-ROM disku. ***Ovdje smo mogli ubaciti DVD ISO sliku bilo kojeg drugog operativnog sustava (Microsoft Windows, FreeBSD Unix, MacOS, …).***

- `APPEND iso raw` - naznačuje kako se *memdisk* kernelu prosljeđuju sljedeće opcije:
    - `iso` - da će biti učitavana ISO datoteka.
    - `raw` - da ISO datoteku treba čitati s *direktnom* metodom pristupa RAM memoriji (tzv. *protected mode*).



Važno je razumjeti kako je za uspješan rad odnosno instalaciju računala preko mreže, upotrebom ***PXE boot*** metode na strani poslužitelja potrebno sljedeće:

✓ Instaliran i konfiguriran ***TFTP*** poslužitelj sa svim potrebnim datotekama i postavljenim ovlastima na iste.
✓ Instaliran i konfiguriran primjerice ***NFS*** poslužitelj sa svim potrebnim datotekama i postavljenim ovlastima na iste.
✓ Instaliran i konfiguriran ***DHCP*** poslužitelj s uključenim opcijama za ***BOOTP*** kao i postavkama za ***TFTP*** i ***NFS.***

Na strani klijenta na koji radimo instalaciju ili pokretanje operacijskog sustava preko ***PXE*** funkcionalnosti potrebno je:

✓ Mrežna kartica koja podržava ***PXE boot.***
✓ ***BIOS*** računala koji podržava ***PXE boot*** uz postavke da se računalo inicijalizira preko ***PXE boot*** (***LAN boot***) metode.

**Izvori informacija: (720),(961),(962),(1149),(1150) RFC 783, RFC 906, RFC 951, RFC 1497, RFC 2132, RFC 2131, RFC 5071.**

### 25.8.4.4. Kickstart metoda automatizirane instalacije operativnog sustava

***Kickstart*** instalacije nude metode za automatizaciju procesa instalacije, bilo djelomično ili u potpunosti. *Kickstart* datoteke sadrže odgovore na sva pitanja koja tijekom instalacije operativnog sustava postavlja instalacijski program, poput toga koju vremensku zonu želite da sustav koristi, kako particionirati diskove i formatirati particije diska ili koje softverske pakete treba instalirati. Postavljanjem pripremljene *kickstart* datoteke kada instalacija operativnog sustava započne omogućuje vam da instalaciju izvedete automatski, bez potrebe za bilo kakvom intervencijom korisnika. Ovo je posebno korisno kada se operativni sustav instalira na veliki broj sustava (računala, poslužitelja i sl.) odjednom. *Kickstart* datoteke obično se spremaju na jednom poslužitelju, a čitaju na pojedinačnim računalima tijekom instalacije.

Ova metoda instalacije podržava korištenje jedne *Kickstart* datoteka za instalaciju operativnog sustava na više računala, što ga čini idealnim za administratore mreže i sustava. Preporučeni pristup pri stvaranju *Kickstart* datoteka je prvo izvođenje ručne instalacije ***Red Hat*** kompatibilnog (*CentOS/Fedora/Rocky/Oracle Linux/...*) operativnog sustava na jednom računalu. Nakon dovršetka instalacije, svi izbori napravljeni tijekom instalacije spremaju se u datoteku pod nazivom `anaconda-ks.cfg`, koja se nalazi u `/root/` direktoriju na instaliranom računalu. Zatim možete kopirati ovu datoteku, napraviti sve potrebne promjene i koristiti dobivenu konfiguracijsku datoteku u daljnjim instalacijama.



Ako smo u koraku 9. za svako računalo koje se instalira, kreirali zasebnu *`pxelinux.cfg`* datoteku, prema njegovoj **MAC** adresi, moguće je u svakoj od njih imati definiranu drugu *Kickstart* datoteku te tako individualizirati instalacije računala.

Pogledajmo samo osnovni dio *kickstart* datoteke koju smo koristili u prethodnom poglavlju (uz naše komentare u istom retku):

```
# Odabir algoritma SHA512 za pohranu lozinki i aktivacija shadow načina pohrane lozinki
auth --enableshadow --passalgo=sha512
# Root lozinka [provjerni zbroj]
rootpw --iscrypted
$6$MnGRqjekOOVg1sdfsx$XS9pxDP/aFBppHKJhweHvSF5b6PiyvCSZPgBJq97QU0TDJlz6MYIFFytQccZ6q/
# Instalacija u tekstualnom načinu rada
text
# Restart računala nakon instalacije)
reboot
# Instalacija sustava od početka [nova instalacija]
install
# Slijedi nastavak instalacije s NFS poslužitelja (s IP adrese 192.168.1.1 u ovom primjeru)
nfs --server=192.168.1.1 --dir=/install/rocky
firstboot --disable
ignoredisk --only-use=sda
# Postavke tipkovnice
keyboard --vckeymap=us --xlayouts='us'
# Postavke jezika i kodiranja
lang en_US.UTF-8
# Postavke vremenske zone
timezone Europe/Zagreb
# Isključivanje vatrozida (servisa/damona)
firewall --disabled
# Postavke GRUB2 boot loadera koje će se primjeniti
bootloader --append=" crashkernel=auto" --location=mbr --boot-drive=sda
# Brisanje MBR zapisa s diska, prije particioniranja diska (za svaki slučaj)
zerombr
# Brisanje svih particija s /dev/sda diska (ako ih ima na ovom disku)
clearpart --all --initlabel --drives=sda
# Kreiranje particija na disku /dev/sda te definiranje njihove veličine i kreiranje datotečnog sustava [u MB]
part swap --fstype="swap" --ondisk=sda --size=16000
part /  --fstype="xfs" --asprimary --ondisk=sda --label=sys --size=30000
# Definiranje konfiguracije eno1 mrežne kartica da koristi DHCP poslužitelj
network  --bootproto=dhcp --device=eno1 --onboot=on

# U ovoj sekciji slijedi definicija softverskih paketa i/ili grupa softverskih paketa koji će biti instalirani na sustav
%packages
@core
nfs-utils
%end

%post --log=/opt/ks-post.log
# U ovoj sekciji (%post) se mogu dodavati željene naredbe koje treba izvršiti na kraju instalacije sustava
%end
```



*Za sve dostupne opcije i parametre **Kickstart** datoteka, pogledajte izvor informacija (**1149**).*

Pogledajmo blok dijagram komunikacije tijekom instalacije računala preko mreže, korištenjem ***PXE*** metode instalacije. U ovom primjeru ***PXE boot*** poslužitelj ima pokrenute ***DHCP***/***BOOTP*** i ***TFTP*** te ***NFS*** servise. Dakle na blok dijagramu na slici **241.2**, pratimo pojednostavljene korake koje smo detaljnije objasnili u prethodnim poglavljima.

**Slika 241.2 *PXE boot* odnosno *network install* procedura s mrežnog** instalacijskog poslužitelja.

Net Install Poslužitelj
DHCP poslužitelj
BOOTP poslužitelj
TFTP poslužitelj
+
NFS poslužitelj

PXE
(net install)
računalo

PXE boot procedura (BIOS ili UEFI)

DHCP poslužitelj

1 DHCP Discovery (upit)

2 DHCP Offer (odgovor)

IP adresa i maska mreže za računalo [pr. 192.168.1.100/24], podrazumijevani usmjerivač, DNS poslužitelji, IP adresa TFTP poslužitelja (opcija 66) [192.168.1.1] i ime Boot Loader datoteke (opcija 67) [pxelinux.0] te eventualne i druge opcije.

3 DHCP request (potvrda odgovora)

4 DHCP Acknowledge (potvrda rezervacije IP parametara)

Računalo je dobilo i postavilo:
IP adresu: 192.168.1.100
Masku mreže : 255.255.255.0
. . .
TFTP poslužitelj: 192.168.1.1
BOOT LOADER: pxelinux.0

TFTP poslužitelj

5 TFTP klijent traži veličinu *BOOT LOADERa* (datoteke pxelinux.0) i dohvaća ju

BOOT LOADER
(pxelinux.0)
je učitan

6 Traži se konfiguracijska datoteka* unutar direktorija (pxelinux.cfg/) i učitava

Na osnovi ove* datoteke se dohvaćaju i učitavaju kernel:
[vmlinuz]
te inicijalni RAM disk:
[initrd.img]

NFS poslužitelj

Kickstart procedura instalacije OS-a

7 Traži se i učitava *Kickstart* datoteka:
[NFS (u primjeru): /opt/install/kickstart/server-1]

Kako je definirano* traži se način dohvaćanja *Kickstart* datoteke (pr. s NFS poslužitelja)

8 Kako je definirano u *Kickstart* datoteci, dohvaća se i instalira ostatak operativnog sustava



Dobro proučite cijelo prethodno poglavlje:
**25.8.4. Bootp protokol** i sve njegove cjeline.



Navedeni redoslijed prilikom pokretanja ili instalacije operativnog sustava preko mreže (**PXE**) je za računala/poslužitelje koji koriste **BIOS**, a ne **UEFI**.

**Za UEFI je drugačiji korak 5 na slici jer se koristi drugi BOOT MANAGER, koji je prilagođen za UEFI način rada.**

## Instalacija i konfiguracija *PXE* poslužitelja

U ovom koraku instalirat ćemo sve potrebne programe i servise kako bi napravili svoj ***PXE*** poslužitelj.
Sva konfiguracija koju ćemo raditi, prilagođena je za ***RedHat/CentOS/Rocky 7.x*** ili noviji, mada bi sve trebalo raditi i na starijim inačicama. Dakle za ***PXE*** poslužitelj potrebi su nam sljedeći servisi: *DHCP+BOOTP*, *TFTP* i *NFS (ili HTTP/FTP)*.
Kako bi konfiguracija bila što jednostavnija, mi nećemo koristiti kombinaciju ***dhcpd*** + ***xinetd***+ ***tftpd*** već jedan jedini servis koji objedinjuje njihove funkcionalnosti a zove se `dnsmasq`. Dakle on je ***DHCP***, ***BOOTP***, ***TFTP*** i ***DNS*** poslužitelj.

On je primjerice preporučeni i standardno korišteni poslužitelj za navedene protokole na ***OpenStack*** platformi za virtualizaciju.
Instalirajmo `dnsmasq` servis te ***syslinux*** paket unutar kojeg ćemo dobiti potrebne *BOOT MANAGERE/LOADERE*:

```
yum -y install dnsmasq syslinux
```

### Konfiguracija Dnsmasq servisa

Konfiguracijska datoteka *dnsmasqa* je: `/etc/dnsmasq.conf` u kojoj se radi konfiguracija svih navedenih protokola koje on podržava. Naš budući ***PXE*** poslužitelj ima mrežnu karticu `eno50` s IP adresom **`172.255.0.1`** koju ćemo koristiti za **PXE**.
*Syslinux* programski paket instalira sve dostupne *BOOT LOADERE* unutar direktorija: `/usr/share/syslinux/`.

Uredit ćemo datoteku `/etc/dnsmasq.conf` dodavanjem sljedećih redaka na kraj, uz naše opise [**#**] uz svaki redak:

```
interface=eno50                                         # Definicija mrežne kartice
bind-interfaces                                         # Uporaba samo gornje kartice
dhcp-range=172.255.0.10,172.255.0.250,12h             # Definicija opsega IP adresa
dhcp-option=option:router,172.255.0.2      #Podrazumijevani usmjerivač (Default Gateway)
dhcp-option=option:dns-server,8.8.8.8                  # Definicija DNS poslužitelja
dhcp-option=eno50,66,172.255.0.1       # Definicija TFTP poslužitelja (DHCP opcija 66)
dhcp-boot=pxelinux.0          # Definicija BOOT LOADER datoteke na TFTP poslužitelju
enable-tftp                                            # Aktivacija TFTP poslužitelja
tftp-root=/tftpboot          # Definicija početnog direktorija za TFTP poslužitelj
```

Vezano za *DHCP* opcije koje koristimo, moguće je koristiti dvije sintakse poput simboličke:
`dhcp-option=option:router,`**`IP`** ili možemo definirati *DHCP* opciju prema broju opcije (primjerice DHCP opcija `66`):
`dhcp-option=`**`INTF`**`,66,`**`IP`**. Pri tome **`IP`** označava IP adresu, a **`INTF`** mrežnu karticu na koju se primjenjuje.

Pošto je moguće da sustav prepozna radi li se o **BIOS** ili **UEFI** računalu to ćemo i iskoristiti, pa je u slučaju kada odredišno računalo potencijalno koristi **UEFI**, potrebno dodati sljedeće retke u konfiguraciju (broj `7` je identifikator **UEFI** arhitekture):

```
dhcp-match=set:efi-x86_64,option:client-arch,7
dhcp-boot=tag:efi-x86_64,BOOTX64.EFI
```

U slučaju kada želite da se sve poruke *dnsmasq* servisa spremaju u log datoteku: `/var/log/messages`, tada otvorite datoteku `/usr/lib/systemd/system/dnsmasq.service` i umjesto sljedećeg retka:

```
ExecStart=/usr/sbin/dnsmasq -k
```

Dodajte na kraj `-q` pa će sada ovaj redak izgledati ovako:

```
ExecStart=/usr/sbin/dnsmasq -k -q
```

Snimite ovu datoteku te pokrenite proceduru za rekreiranje *systemd* datoteka, s naredbom:

```
systemctl daemon-reload
```

Potom možemo restartati *dnsmasq* servis, koji će sada sve svoje interne poruke snimati u datoteku: `/var/log/messages` .

```
systemctl restart dnsmasq
```

### Priprema direktorija i Boot loadera

Međutim da bismo dobili 64-bitni ***UEFI BOOT LOADER***, potrebno je instalirati dodatni paket zvan ***shim***:

```
yum -y install shim-x64
```



Za druge primjene servisa **`dnsmasq`** pogledajte poglavlje:
**25.8.3.3. Konfiguracija DHCP poslužitelja.**

Sada trebamo kreirati inicijalni direktorij za *TFTP* servis, te stablo poddirektorija i datoteka u njemu:

```
mkdir -p /tftpboot/pxeboot /tftpboot/pxelinux.cfg          # Priprema direktorija
cp  /usr/share/syslinux/pxelinux.0 /tftpboot/              # Kopiranje BIOS BOOT LOADERA
cp /usr/share/syslinux/ldlinux.c32 /tftpboot/             # Modul za BIOS BOOT LOADER
cp /boot/efi/EFI/BOOT/BOOTX64.EFI /tftpboot/              # Kopiranje UEFI BOOT LOADERA
cp /boot/efi/EFI/BOOT/fbx64.efi /tftpboot/                # Modul za UEFI BOOT LOADER
chown -R dnsmasq:dnsmasq /tftpboot/                       # Promjena vlasnika i grupe
chmod 777 /tftpboot                                       # Promjena ovlasti
```



*Ako koristite **SELinux** pazite da vam ne zabrani pristup direktoriju* **`/tftpboot`** *(ili ga isključite).*
*Također, ako koristite **vatrozid** ili ga privremeno isključite ili dodajte pravila za pristup na DHCP, TFTP i NFS servise.*

**Priprema DVD ISO datoteke**

Sada prekopirajmo ***ISO*** datoteku Linux distribucije koju želimo koristiti da se instalira preko ***PXE*** metode instalacije.

```
mkdir /mnt/iso
mount -o loop /install/ISO/Rocky-8.5-x86_64-dvd1.iso /mnt/iso
```

U ovom koraku imamo montiranu ***ISO*** datoteku `Rocky-8.5-x86_64-dvd1.iso` Rocky v.8.5. Linuxa u direktorij: `/mnt/iso`. Potom ćemo kopirati inicijalni RAM disk (`initrd.img`) i linux kernel (`vmlinuz`) u naš *TFTP* direktorij:

```
cp -v /mnt/iso/images/pxeboot/{initrd.img,vmlinuz} /tftpboot/pxeboot/
```

Kreirat ćemo i stablo direktorija u kojem će se nalaziti sve datoteke s ***ISO*** slike Linuxa koji ćemo koristiti i upravo kopirati.

```
mkdir -p /install/rocky /install/kickstart
rsync -avz /mnt/iso/ /install/rocky
```

Dio s kopiranjem (naredba `rsync`) može potrajati jer se cijeli sadržaj ***DVD ISO*** slike mora kopirati u odredišni direktorij.

**Priprema *PXE boot* konfiguracije**

Nakon što se sve kopira, kreirat ćemo datoteku imena: `/tftpboot/pxelinux.cfg/default` i u nju kopirati sljedeće retke konfiguracije. Ovo je dio u kojem se definiraju: ***initrd***, ***kernel*** i ***kickstart*** metoda automatizirane instalacije sustava:

```
default rocky

label rocky
kernel pxeboot/vmlinuz
append initrd=pxeboot/initrd.img ramdisk_size=10000 ks=nfs:
172.255.0.1:/install/kickstart/serveri ksdevice=eno50 nomodeset
```

Ovdje je moguće kreirati i izbornike za više mogućih izvora za instalaciju. Pogledajte primjere konfiguracije **(1161)**.
Sada u ***kickstart*** direktorij `/install/kickstart/` koji smo upravo definirali, moramo ubaciti ***kickstart**** datoteku imena `serveri`.

**Instalacija privremenog računala i dorada njegove kickstart datoteke**

Naime nakon svake standardne instalacije ***Red Hat*** kompatibilne distribucije Linuxa, kreira se *kickstart* datoteka: `/root/anaconda-ks.cfg` koja sadrži sve parametre i opcije koje su se koristile tijekom instalacije tog sustava (računala).
Stoga ćemo prvo instalirati jedno računalo s iste ***ISO*** *image* datoteke (`Rocky-8.5-x86_64-dvd1.iso`) koju smo prethodno koristili, na način koji će nam odgovarati za sva računala koja želimo instalirati s **PXE** metodom: od particioniranja i formatiranja diskova, instalacije potrebnih softverskih paketa i svega drugog, što nam nudi instalacija ovog operativnog sustava.
Nju ćemo kasnije nakon instalacije iskoristiti kao inicijalnu ***kickstart*** datoteku, koju možemo i prilagoditi za naše potrebe.
Dakle datoteku `/root/anaconda-ks.cfg` ćemo prekopirati s novo instaliranog računala na naš ***PXE*** poslužitelj, ali sada s novim nazivom datoteke `/install/kickstart/serveri` jer ćemo upravo nju koristiti kao predložak za buduće ***PXE*** instalacije. U ***kickstart*** datoteci obratite pažnju na sljedeću opciju, odnosno prvo dodajte sljedeći redak, zbog **NFS** poslužitelja:

```
#Instalacija preko NFS poslužitelja (dodati prije sekcije s diskovima)
nfs --server=172.255.0.1 --dir=/install/rocky
```

Dakle s ovim unosom, instalacijskoj proceduri nalažemo da krene s instalacijom novog operativnog sustava preko **NFS** poslužitelja, koji će za tu namjenu eksportirati (preko mreže), direktorij: `/install/rocky` u koji smo prethodno prekopirali cijeli sadržaj ***DVD ISO*** slike instalacije Linuxa.

Sada još trebamo svima dozvoliti prava čitanja svih gore kreiranih datoteka i direktorija, što ćemo napravit sa:

```
chmod -R a+r /install/ /tftpboot/
```

**Instalacija i konfiguracija NFS servisa**

***NFS*** servis koristit ćemo za nastavak ***PXE*** instalacije. Prvo ga instalirajmo na sljedeći način:

```
yum -y install nfs-utils
```

Sada već kreirani i pripremljeni direktorij u kojem su sve datoteke Linuxa za instalaciju (`/install/rocky`) moramo definirati da će se eksportirati preko *NFSa*, dodavanjem sljedećeg retka u datoteku: `/etc/exports`. Osim njega ***PXE*** procedura će u zadnjem koraku morati moći doći do ***PXE*** datoteke u direktoriju: `/install/kickstart`.
Ovdje ćemo još dodati i neke opcije. Kako bi postigli sve navedeno, trebamo dodati sljedeći redak u datoteku `/etc/exports` kako bi eksportiranje gore kreiranog direktorija preko *NFS* protokola bilo omogućeno:

```
/install/rocky *(rw,sync,no_root_squash)
/install/kickstart *(rw,sync,no_root_squash)
```

Sada možemo pokrenuti ***NFS*** servis:

```
systemctl start nfs-server
```

Zatim ga trajno aktivirajmo:

```
systemctl enable nfs-server
```

I potom provjerimo je li gore navedeni direktorij eksportiran preko *NFSa*:

```
showmount -e

Export list for localhost.localdomain:
/install/rocky *
/install/kickstart *
```

Vidimo da je eksportiran jer vidimo oba direktorija (`/install/rocky` i `/install/kickstart`) eksportirana.
Tek sada je **PXE** poslužitelj spreman za uporabu.


U slučaju da vam nešto ne radi, pratite greške koje dobivate i pokušajte ih popraviti. Naime zbog određenih specifičnosti i raznih inačica svih navedenih programa i komponenti, moguća su određena odstupanja. Greške primjerice možete pratiti s: `tail -f /var/log/messages` i korištenjem naredbe `tcpdump` na mrežnom sučelju poslužitelja koji se koristi za ***PXE***.


Ukoliko nakon instalacije operativnog sustava želimo da se sustav ponovno inicijalno konfigurira (iako je možda već konfiguriran), potrebno je napraviti sljedeće.

Naime ako na sustavu u vršnom korijenskom direktoriju (mapi) / postoji datoteka imena: `.unconfigured`
tada će sustav nakon prvog restarta pokrenuti proceduru ponovne konfiguracije sustava odnosno imena računala, mrežnih postavki i slično. Ako to želimo napraviti, dovoljno je kreirati ovu datoteku:

```
touch /.unconfigured
```

Dakle ako ova datoteka postoji na sustavu sustav će odmah pokrenuti proceduru ponovne (re)konfiguracije sustava, kao da je računalo tek instalirano (bilo da se radi o fizičkom ili virtualnom računalu).


Ako se radi o virtualnom računalu od kojeg želimo napraviti predložak za kopiranje (***VM Template***) tada imamo nekoliko opcija nakon instalacije tog virtualnog računala. Dakle želimo očistiti njegovu konfiguraciju (mrežne postavke i drugo).

**Za Red Hat/CentOS do v.7.x.**
Možemo instalirati sljedeći programski paket:

```
yum -y initial-setup
```

I potom pokrenuti naredbu za čišćenje konfiguracije sustava:

```
sys-unconfig
```

Nakon toga virtualno računalo je očišćeno od konfiguracije sustava (mrežne i druge postavke sustava).

**Za Red Hat/CentOS od v.8.x.**
Možemo instalirati sljedeći programski paket:

```
yum -y cloud-init
```

I potom pokrenuti naredbu za čišćenje konfiguracije sustava:

```
virt-sysprep
```

Nakon toga virtualno računalo je očišćeno od konfiguracije sustava (mrežne i druge postavke sustava).

**Izvori informacija:(720),(961),(962),(1149),(1150),(1151),(1152),(1153),(1154),(1155),(1156),(1157),(1158),(1159),(1160), (1161), `man dnsmasq`,`man syslinux`,`man virt-sysprep`,`man sys-unconfig`, RFC 783, RFC 906, RFC 951, RFC 1497, RFC 2132, RFC 2131.**

### 25.8.5. DNS protokol

U svim TCP/IP mrežama, a to su praktično sve današnje računalne mreže, za komunikaciju između računala koriste se isključivo IP adrese kao jedinstveni identifikatori svakog računala ili uređaja na mreži. U starijim i manjim mrežama u kojima nam je bila poznata svaka IP adresa, svakog računala u mreži, ponekad je bilo jednostavnije pristupati im direktno pomoću njihove IP adrese. S obzirom na to kako su IP adrese, poput 192.168.100.234, nama ljudima ne baš previše pamtljive, uvelo se posredovanje u pristupu mrežnim uređajima i računalima, pomoću imena računala, a koje nam je znatno pamtljivije, poput: *server1*, *marko*, *ivan* i slično. I dalje je na nižoj razini mrežne komunikacije sve ostalo na IP adresama, ali smo na višoj razini, odnosno u aplikacijama mogli koristiti imena računala (Engl. *Hostname*) jer nam je to pamtljivije. Kako mrežna komunikacija i dalje koristi IP adrese, potrebno je nekako povezati ime računala s njegovom IP adresom. Prvi i najjednostavniji način povezivanja između IP adrese i imena računala jest ubacivanjem svakog pojedinog imena računala i njegove pripadajuće IP adrese u posebnu tablicu, odnosno datoteku, koja se zove ***host*** datoteka odnosno konkretno datoteka: `/etc/hosts`.

U *Windows* operativnim sustavima ova datoteka se nalazi na lokaciji: ***C:\Windows\System32\drivers\etc\hosts***, dok se na Unix/Linux operativnim sustavima ona nalazi u datoteci: `/etc/hosts`. Naime u svaki novi redak ove datoteke morali bi upisati svako pojedino računalo na mreži, kako bismo mu mogli pristupati pomoću imena, a ne samo pomoću IP adrese.



Vezano za datoteku `/etc/hosts` pogledajte poglavlje:
**25.6. Osnovna konfiguracija mreže i mrežnog podsustava.**

Unosi u datoteci `/etc/hosts` izgledaju otprilike ovako:

```
192.168.100.50 marko
192.168.100.51 ivan
192.168.100.234 server1
```

Sâmo računalo, svaki puta kada bi pristupalo drugom računalu na mreži pomoću imena tog računala, tražilo bi prvo u ovoj datoteci njegovu pripadajuću IP adresu, kojoj bi na kraju i pristupilo. Razvojem i širenjem mreža, a pogotovo razvojem *interneta* i pristupanjem računalima na internetu, ovakav način upisivanjem svakog pojedinog računala i njegove IP adrese je očito postao nemoguć. Stoga se uveo poseban servis odnosno mrežni protokol koji bi svako računalo moralo kontaktirati, kako bismo na osnovu imena računala moglo saznati njegovu IP adresu kojoj u konačnici želimo pristupiti. Ovaj protokol se zove **DNS** (Engl. *Domain Name System*). Dakle DNS servis odnosno poslužitelj se kontaktira svaki puta kada želimo saznati IP adresu nekog računala na mreži, kojemu želimo pristupiti koristeći njegovo ime računala (*hostname*). *Domain Name System* (***DNS***) protokol je hijerarhijski, distribuirani imenični sustav koji barata s imenima odnosno nazivima (engl. *Naming*), a koristi se za razlučivanje imena računala, servisa ili drugih resursa na mreži.
DNS primarno povezuje ime računala s njegovom pripadajućom IP adresom, koja je osnovni element u mrežnoj komunikaciji. Dakle DNS kontaktiramo kada želimo saznati IP adresu na osnovi imena računala odnosno tzv. *FQDN* imena *(Fully qualified domain name*) na mreži.

DNS poslužitelji koje naš sustav može kontaktirati odnosno koristiti, definiraju se u datoteci `/etc/resolv.conf` s ključnom riječi `nameserver`. U ***Windows*** operativnom sustavu DNS poslužitelje u upotrebi dobivamo s naredbom:
`ipconfig /all`, a oni su vidljivi unutar mrežnog sučelja pod nazivom: **DNS Servers**.



Vezano za konfiguraciju **DNS** poslužitelja koje naš Linux sustav koristi, pogledajte poglavlje:
**25.6.2. Konfiguracijska datoteka /etc/resolv.conf.**

Dodatno kako bi bilo lakše pamtiti imena računala: prema njihovoj državi, namjeni, upotrebi, tvrtki, instituciji i drugim parametrima, uvedene su takozvane domene. Domena je hijerarhijska struktura naziva koja počinje od vršne domene `.` hijerarhijski sve prema dolje odnosno prema nižim domenama u hijerarhiji.
Drugim riječima ime domene predstavlja poziciju unutar DNS hijerarhije.
Domena se sastoji od liste svih domena, počevši od vršne domene sve do najdonje u hijerarhiji, odvojenih točkom (`.`). Kod ovakvog hijerarhijskog DNS sustava, raspoređuju se odgovornosti za svaku pojedinu domenu, na takozvani autoritativni DNS poslužitelj koji je odgovoran za svaku određenu (pojedinu) domenu. Dakle svaka domena mora imati svoj autoritativni DNS poslužitelj (ili više njih). Autoritativnim imeničkim poslužiteljima je dodijeljena odgovornost za (svoju) točno određenu domenu, a po redu im se mogu pripisati drugi imenički poslužitelji za njihove pôd domene. Ovaj mehanizam je izgradio DNS kao distribuiran i tolerantan na greške, ali je i pomogao u izbjegavanju potrebe za jednim jedinstvenim registrom za cijeli svijet, koji bi se trebao neprestance konzultirati i ažurirati, što bi bilo praktično (gotovo) neizvedivo.

Odgovornost za održavanje i ažuriranje glavnog zapisa (*master record*) za domene je raširena među mnogim registrima domenskih imena, koji se nadmeću za obavljanje ovog zadatka (među vlasnicima domene). Domene se mogu pomicati iz registra u registar u bilo koje vrijeme. **DNS** imenički poslužitelj može spremati **DNS** zapise za domenska imena, kao što su adresni zapisi (**A** ili **AAA**), za imeničke poslužiteljske (**NS**) zapise te zapis poslužitelja elektroničke pošte (**MX**) i mnoge druge.
DNS imenički poslužitelj odgovara na upite iz svoje baze podataka.



Osnovna funkcionalnost DNS sustava jest prevođenje lako pamtljivih domenskih imena u numeričke IP adrese.

**DNS koristi u većini slučajeva** UDP **protokol za transport, na portu broj 53.**
DNS komunikacija je prema principu: klijent ←→ poslužitelj.
U nekim slučajevima se koristi **TCP** protokol za transport:

- Kada je odgovor od strane poslužitelja veći od 512 *bajta.*
- Kada se radi takozvani transfer zone (Engl. *zone transfer).*
- Kada DNS klijent koji inače komunicira s DNS poslužiteljem preko *UDP* protokola, nije dobio odgovor, za nekoliko sekundi (ovisi o implementaciji: 3-5 sekundi), on mora ponoviti DNS upit, korištenjem TCP protokola.

***DNS*** protokol je inicijalno definiran u RFC 882 i RFC 883 te kasnije nadopunjen u RFC 1035.

**Izvori informacija: (1136),(1137), RFC 882, RFC 883, RFC1034, RFC 1035.**

Pogledajmo kako izgleda domenska hijerarhija:
**1.** Na vrhu hijerarhije nalazi se korijenska domena koja je označena s točkom `.` a koja se inače ne vidi.
**2.** Postoje takozvane vršne odnosno *Top level* domene, koje su točno definirane i ne mogu se mijenjati.
Pogledajmo tablicu s definiranim vršnim domenama (naveli smo ih samo nekoliko):

| Domena | Tko ju koristi |
|---|---|
| `.com` | Komercijalne organizacije. |
| `.org` | Neprofitne organizacije. |
| `.edu` | Edukacijske organizacije. |
| `.gov` | Državne agencije. |
| `.mil` | Vojne organizacije. |
| `.net` | Organizacije koje se bave mrežama. |

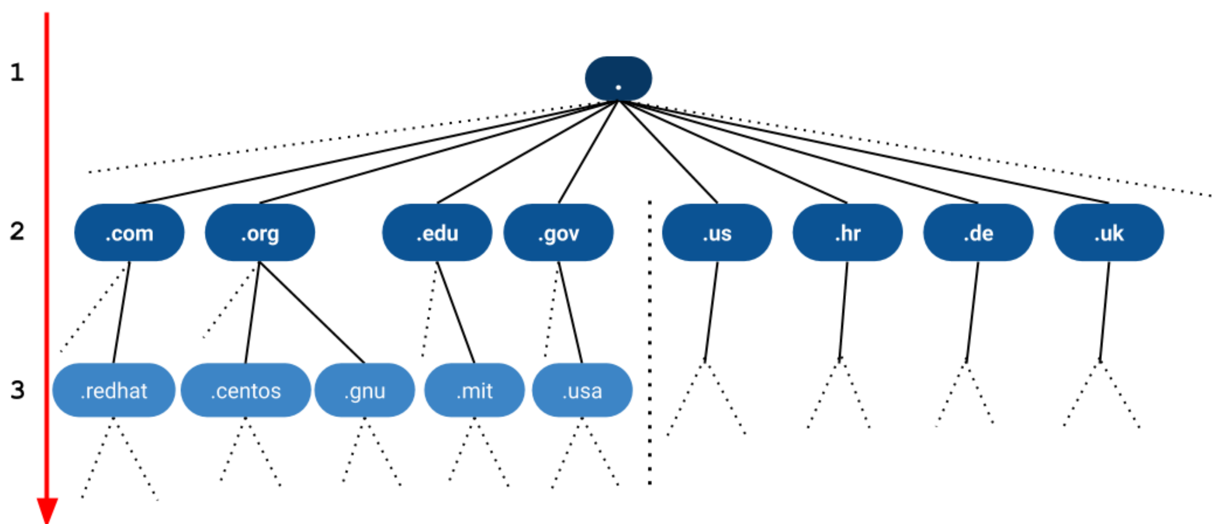
Osim ovih vršnih domena prema namjeni, postoje i niže vršne domene koje predstavljaju države svijeta, poput:

| Domena | Država |
|---|---|
| `.us` | Sjedinjene Američke države (SAD). |
| `.hr` | Hrvatska. |
| `.de` | Njemačka. |
| `.uk` | Velika Britanija. |

**3.** Ispod vršnih domena postoje pôd domene, kao i njihove pôd domene te pôd pôddomene i tako dalje.

Kako to logički izgleda pogledajte na slici 242.

**Slika 242. DNS: pogled na vršne domene i hijerarhijsku strukturu DNSa.**



**Svaki puta kada primjerice preko web preglednika otvaramo stranicu na adresi www.redhat.com događa se sljedeće:**



Inicijalno naše računalo čita datoteku: /etc/nsswitch.conf te pronalazi unos hosts:.
U njemu je definiran redoslijed, kojim se (i što) treba čitati za razlučivanje IP adrese iz imena računala.
Uobičajeno je to prvo definicija files – koja definira čitanje datoteke: /etc/hosts.
Tek potom, ako se u njoj ne može pronači ime računala i njegova pripadajuća IP adresa, kreće se dalje.
Drugi korak pretrage je obično definicija dns – koja upućuje na DNS poslužitelje definirane u datoteci: /etc/resolv.conf.

1. Naš web preglednik tada kontaktira prvi DNS poslužitelj koji mu je konfiguriran (u /etc/resolv.conf) i pita ga tko poznaje IP adresu od: www.redhat.com. Zapravo svaka adresa izgleda ovako (završava s točkom) www.redhat.com.
2. Nama prvi DNS poslužitelj provjerava u svojoj bazi ili međumemoriji* može li on odgovoriti. Ako ne zna odgovor, on kontaktira prvi vršni DNS poslužitelj u hijerarhiji koji je odgovoran za vršnu domenu .com. Ako slučajno on (nama prvi DNS poslužitelj) ne zna IP adresu DNS poslužitelja za vršnu domenu .com on tada mora kontaktirati neki od korijenskih odnosno "***root***" poslužitelja za vršnu domenu . koji definitivno moraju imati IP adrese svih nižih (u hijerarhiji) vršnih poslužitelja.
Dakle oni moraju imati podatke od DNS poslužitelja vršnih domena:

- Vršnih domena poput: **.com**, **.net**, **.org** …
- Kao i za vršne domene za države: **.hr**, **.us**, **.de**, **.uk** …

Naš DNS poslužitelj potom postavlja upit DNS poslužitelju zaduženom za domenu .com te on gleda u svoju bazu i daje nam IP adresu od DNS poslužitelja odgovornog za pôd domenu koju smo tražili, točnije za .redhat domenu.
3. Naš DNS poslužitelj potom šalje upit na DNS poslužitelj od tražene pôd domene .redhat kojega se potom pita koja je IP adresa poslužitelja imena www.redhat.com, a on mu odgovara s traženom IP adresom. Tada nama prvi DNS prema našem web pregledniku daje kao odgovor, traženu IP adresu. Primjerice to je IP adresa: 209.132.183.105.
4. Sada kada naš web preglednik zna kako je adresi računala www.redhat.com pripadajuća IP adresa: 209.132.183.105 on se na nju konačno i spaja. Naravno, cijeli ovaj proces može trajati neko vrijeme.



Za više detalja o ovoj komunikaciji, pogledajte i poglavlje:
**25.8.5.4. Vrste DNS upita.**



Važno je znati kako *DNS* poslužitelji svaki puta kada saznaju za nove domene i vršne *DNS* poslužitelje odgovorne za njih, taj podatak čuvaju u svojoj privremenoj memoriji (*cache*) sve dok ne istekne vremenski period koji je definiran za svaku pojedinu domenu (na svakom DNS poslužitelju zasebno). U praksi se radi o vremenima od obično nekoliko sati. Stoga, ako dođe do neke promijene u određenoj domeni, potrebno je prilično dugo vremena (nekada i čak 24.h.) da toga budu svjesni svi DNS poslužitelji u hijerarhiji. Ovo osvježavanje odnosno njegovo namjerno kašnjenje se zove propagacija.

U slučaju kada tražimo podatak o računalu u našoj mreži kao primjerice: server1.nasadomena, naš DNS poslužitelj, ako ga imamo na lokalnoj mreži, odnosno onaj koji je odgovoran za tu domenu, vratit će nam IP adresu od traženog računala. Odgovor će biti IP adresa, poput: 192.168.100.100, a koja se u pozadini i koristiti za komunikaciju.

**Sintaksa domenskih imena**
Domensko ime se sastoji od jednog ili više dijelova, koji se tehnički zovu oznake, a koji su dogovoreno ulančani i odvojeni točkama, poput: www.redhat.com, a pri tome:

- Najdesnija oznaka (.com) predstavlja vršnu domenu.
- Svaka oznaka od desno prema lijevo predstavlja pôddomenu. Ovdje je oznaka redhat pôd domena, domene .com
- Nadalje www je zatim pôd domena (u ovom slučaju je to oznaka računala, ali trenutno to zanemarite) vršne domene redhat

Postoje i određena ograničenja i pravila *DNS* standarda:

- Svaka oznaka može imati do 63 znaka.
- Potpuno domensko ime ne može prelaziti 253 znaka u svojem tekstualnom prikazu. U unutarnjem binarnom DNS prikazu maksimalna dužina zahtijeva je 253 okteta.
- DNS imena tehnički se mogu sastojati od bilo kojeg znaka koji se može predstaviti oktetom. Kakogod, dopuštena formulacija domenskih imena u DNS korijenskoj zoni, te u većini ostalih pôddomena, koristi željeni oblik i skup znakova. Znakovi dopušteni u oznakama su podskup ASCII skupa znakova te uključuju znakove **a** pa sve do **z**, **A** do **Z**, **0** do **9**, te minus. ovo pravilo je poznato kao **LDH** pravilo (*letter, digits, hyphen*). Domenska imena su neovisna o veličini znaka odnosno na veliko ili malo slovo. Oznake ne mogu započeti ili završiti crticom (minusom). Postoji i dodatno pravilo da vršne domene ne smiju biti sâmo brojčani znakovi.
- Ime računala (*hostname*) je ime koje ima pridruženu barem jednu IP adresu. Na primjer domenska imena www.redhat.com i redhat.com su također imena računala, ali com domena **nije**.

Vratimo se na trenutak na vršnu korijensku domenu (.) te pogledajmo kako pomoću naredbe dig dobiti listu vršnih (.) korijenskih *DNS* poslužitelja. Skratili smo ispis na njih sâmo nekoliko (standardno ih postoji 13 i to: od **A** do **M**):

```
dig NS .

;; ANSWER SECTION:
.                        76458    IN       NS       a.root-servers.net.
.                        76458    IN       NS       b.root-servers.net.
.                        76458    IN       NS       c.root-servers.net.
.                        76458    IN       NS       d.root-servers.net.
.                        76458    IN       NS       e.root-servers.net.
.                        76458    IN       NS       f.root-servers.net.
...
;; ADDITIONAL SECTION:
a.root-servers.net.      76531    IN       A        198.41.0.4
b.root-servers.net.      76504    IN       A        199.9.14.201
c.root-servers.net.      76818    IN       A        192.33.4.12
d.root-servers.net.      2427     IN       A        199.7.91.13
e.root-servers.net.      2426     IN       A        192.203.230.10
f.root-servers.net.      76458    IN       A        192.5.5.241
...
```

Dakle vidimo listu svih vršnih korijenskih DNS poslužitelja; onih s točkom (.) u nazivu. Dakle onih koji su po hijerarhiji iznad svih vršnih domena. Oni su zbog sigurnosnih, kao i razloga zalihosti (redundancije), ali i latencije, raspoređeni po kontinentima te regijama unutar njih. Oni su vidljivi na slici 242, na vrhu hijerarhije. Njihove IP adrese su obično posebne ***Anycast*** IP adrese.



DNS zapisi odnosno unosi koje je naše računalo dobilo od DNS poslužitelja se u ***Windows*** operativnom sustavu čuvaju u posebnoj predmemoriji, koja se koristi za sve aplikacije i za koju se brine poseban servis ***dnscache***.
Ova predmemorija se prema potrebi ručno može očistiti s naredbom: `ipconfig /flushdns`.



Međutim ***Linux***, ako ne koristimo neki od DNS servisa ili servisa koji ove zapise spremaju u međumemoriju poput: nscd, bind, dnsmasq, systemd-resolved i drugih, **ne zapisuje** nigdje, te se spremanje ovih unosa tada prepušta sâmim aplikacijama*.

Dakle ako nemamo instaliran neki od navedenih DNS servisa, pronađeni DNS unosi se ne pohranjuju u lokalnu međumemoriju, već se svaki puta (ako to sâma aplikacija ne napravi), dohvaćaju s interneta (od DNS poslužitelja).

Prema potrebi, sve unose u DNS međumemoriji možemo obrisati; ako recimo koristimo nscd servis (***RH/CentOS 6+***), sa:

```
service nscd restart
```

Odnosno ako koristimo systemd-resolved (***RH/CentOS 8+ ili Fedora/Ubuntu Linux***), DNS unose možemo obrisati s naredbom:

```
systemd-resolve --flush-caches
```

Ako pak koristimo servis dnsmasq, DNS unose možemo obrisati sa:

```
systemctl restart dnsmasq
```

i tako dalje, sve ovisno o konkretnom servisu.

Za brisanje DNS međumemorije unutar web preglednika, za ***Chrome*** preglednik, u njegovo adresno polje upišite: **chrome://net-internals/#dns** te odaberite: „***Clear host cache***" i tako smo očistili DNS međumemoriju unutar web preglednika. Drugi web preglednici koriste malo drugačije opcije, pa ih sâmi pronađite i proučite.



*Brisati DNS unose u međumemoriji ponekada trebamo u slučajevima kada ih na silu želimo osvježiti.*

**Izvori informacija: (1133),(1136),(1137), `man nscd`, `man systemd-resolved`, `man bind`, `man dnsmasq`, `man dig`, RFC1034, RFC 1035.**

### 25.8.5.1. DNS detaljnije

DNS poslužiteljskih softvera postoji cijeli nîz, a neki od poznatijih i široko rasprostranjenih su:

- `BIND` - otvorenog je kôda, razvijen na sveučilištu *Berkeley*, 1980, u razvoju i upotrebi je i danas.
- `Unbound` - razvijen kao visoko performantan, u programskom jeziku ***C***, otvorenog je kôda.
- `DNSMasq` - višenamjenski poslužitelj, optimiziran za *embedded*/minijaturne sustave, on osim *DNS* protokola podržava i *TFTP* te *DHCP*/*BOOTP*. Također je otvorenog kôda.
- `NSD` - također je otvorenog kôda i široko je u upotrebi.

**Pogledajmo i oblik DNS poruke, kako je definirano prema standardu RFC 1035.**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Message ID** | | | | | | | |
| QR | OPCODE | AA | TC | RD | RA | **Z** | RCODE |
| **QDCOUNT** | | | | | | | |
| **ANCOUNT** | | | | | | | |
| **NSCOUNT** | | | | | | | |
| **ARCOUNT** | | | | | | | |
| DNS poruka (upit ili odgovor) | | | | | | | |

Opis polja:

- `Message ID` - identificira transakciju.
- `QR` - *Query* - *Response bit* koji može biti:
  - 0 - označava upit (*Query*).
  - 1 - označava odgovor (*Response*).
- `OPCODE` - identificira vrstu upita/odgovora:
  - 0 - označava standardni upit (*QUERY*).
  - 1 - označava inverzni upit (*IQUERY*).
  - 2 - označava upit o DNS statusu (*STATUS*).
- `Flags` odnosno zastavice su sljedeća polja
  - `AA` - Autoritativni odgovor (može biti samo kao odgovor).
  - `TC` - *TrunCation* - poruka je skraćena zbog prevelike dužine.
  - `RD` - *Recursion Desired* - ova opcija može biti unutar zahtjeva (*request*) i unutar odgovora (*response*), ako su rekurzivni upiti/odgovori podržani.
  - `RA` - *Recursion Available* - rekurzija je podržana (rekurzivni upiti/odgovori).
- `Z` - rezervirano za buduću upotrebu
- `RCODE` - ovdje se nalaze odgovori na upit a oni mogu biti:
  - 0 - Nema greške.
  - 1 - *Format error* - poslužitelj ne može interpretirati upit.
  - 2 - *Server failure* - poslužitelj ne može odgovoriti zbog neke greške na njemu.
  - 3 - *Name Error* - samo odgovor: domena ne postoji.
  - 4 - *Not Implemented* - poslužitelj ne podržava ovu funkcionalnost.
  - 5 - *Refused* - poslužitelj odbija dati odgovor.
  - 6 - *YXDomain* - domena postoji iako ne bi trebala.
  - 7 - *YXRRSet* - DNS zapis (*RR*) postoji iako ne bi trebao.
  - 8 - *NXRRSet* -DNS zapis (*RR*) koji bi morao postojati ne postoji.
  - 9 - *NotAuth* - DNS poslužitelj nije autoritativan za zonu (domenu).
  - 10 - *NotZone* - ime se ne nalazi u zoni. …
- `QDCOUNT` - označava broj unosa u "*Question*" sekciji.
- `ADCOUNT` - označava broj unosa u "*Answer*" sekciji.
- `NSCOUNT` - označava broj unosa u "*Authority*" sekciji.
- `ARCOUNT` - označava broj unosa u "*Additional*" sekciji.

**Izvor informacija: RFC 1035.**

### 25.8.5.2. Autoritativni DNS poslužitelj

Autoritativni imenični (DNS) poslužitelj (Engl. *Authoritative name server*) je imenični DNS poslužitelj koji je izričito odgovoran za određenu domenu. Autoritativni DNS poslužitelj može raditi sâmo u dva načina rada:

- Kao glavni poslužitelj odnosno **MASTER**. U teoriji i prema standardu je moguće imati i više *MASTER* poslužitelja.
- Kao pomoćni poslužitelj odnosno **SLAVE** poslužitelj.

Pri tome je **MASTER** poslužitelj onaj koji pohranjuje originalne (Engl. *Master*) kopije svih svojih zonskih unosa (Engl. *Zone records*) odnosno svojih domena. Dok je **SLAVE** poslužitelj onaj koji može preko posebnih mehanizama za ažuriranje zonskih unosa (domena), uskladiti svoje stanje s **MASTER** poslužiteljem.

Dakle promijene koje se zapisuju se mogu zapisivati i mijenjati samo na **MASTER** poslužitelju, dok ih **SLAVE** poslužitelj može sâmo ažurirati s **MASTER**-a, pomoću posebne DNS metode. Prema pravilima, svaka DNS zona *MORA* biti dodijeljena minimalno jednom autoritativnom imeničnom poslužitelju (DNS) i to minimalno jednom **MASTER** poslužitelju.

Ova lista DNS autoritativnih poslužitelja za svaku pojedinu *domenu* pohranjena je u vršnoj domenskoj zoni i to kod vršnog odnosno onog DNS poslužitelja koji je u hijerarhiji iznad naše domene i to unutar njegovih ***NS*** zapisa (Engl. ***NS*** *Record*). Autoritativni DNS poslužitelj indicira svoj status za određenu zonu unutar DNS poruke, postavljajući zastavicu unutar polja sa zastavicama. Ova zastavica je u takozvanoj poziciji **AA** (*Authoritative Answer*).

**Izvor informacija: (1137), RFC1034, RFC 1035.**

### 25.8.5.3. Rekurzivni predmemorijski (*caching*) DNS

U teoriji autoritativni imenski poslužitelji su dovoljni za funkcioniranje Interneta i svake druge mreže koja ih koristi.
Ali sâmo s upotrebom autoritativnih imenskih poslužitelja svaki DNS upit mora početi s rekurzivnim upitima u korijenskoj zoni (`.`) *DNS* hijerarhije. Dodatno i svaki DNS softver mora imati implementiranu metodu rekurzivnih upita.

Kako bi se poboljšala učinkovitost i smanjio promet preko interneta te povećala učinkovitost krajnjih korisničkih aplikacija, dodana je i podrška za predmemorijski (Engl. *cache*) rad DNS poslužitelja. Ovakvi DNS poslužitelji spremaju rezultate DNS upita na određeno vrijeme.

Točnije ovi DNS zapisi se čuvaju u predmemoriji sve dok je vrijeme života određenog DNS zapisa zadovoljavajuće, prema polju koje se nalazi u konfiguraciji svakog zapisa (Engl. *time-to-live* polje). Obično predmemorijski (*caching*) DNS poslužitelji imaju implementiranu i *rekurzivnu* i *caching* metodu rada. Ovakvim radom se ubrzalo dohvaćanje DNS zapisa, stoga što se rezultat svakog upita DNS klijenta (pr. našeg računala) preko ovakvog DNS poslužitelja čuva unutar DNS poslužitelja te se kod sljedećeg istog upita, bilo kojeg drugog ili istog DNS klijenta, zapis se trenutno dohvaća iz predmemorije ovog DNS poslužitelja, bez potrebe za kontaktiranjem cijelog nîza vanjskih autoritativnih DNS poslužitelja.

Ovakav rad je standardno implementiran u mnoge DNS servise, a poglavito u one koji su implementirani u pristupne mrežne uređaje odnosno usmjerivače (*Routere*) na našoj lokalnoj mreži.

Pogledajmo listu često korištenih DNS poslužitelja otvorenog kôda (koji podržavaju i upotrebu predmemorije [*cache*]):

| Ime DNS poslužitelja | Komentar |
|---|---|
| **`bind`** | Besplatni softverski proizvod (otvorenog kôda) i distribuira se s većinom Unix i Linux distribucije, gdje se najčešće naziva i ***named***. On je najrasprostranjeniji DNS poslužitelj. Povijesno gledano, BIND je prošao tri velike revizije, svaka sa značajno različitom arhitekturom: BIND4, BIND8 i BIND9. |
| **`dnsmasq`** | Također je otvorenog kôda i zauzima vrlo malo prostora te je jednostavan za konfiguriranje. Dizajniran za pružanje DNS, ali i opcionalno DHCP i TFTP usluga maloj mreži. |
| **`nsd`** | Isto je otvorenog kôda, razvijen od tvrtke ***NLNet Labs***. NSD je često testni poslužitelj za DNSSEC. Nove značajke DNSSEC protokola često se prototipiraju pomoću baze programskog kôda NSD-a. NSD se vrlo često koristi za domene najviše razine i upravlja s tri korijenska poslužitelja imena DNS-a. |
| **`unbound`** | On je DNS poslužitelj otvorenog kôda, dizajniran za visoke performanse. Objavljen je 20. svibnja 2008. (verzija 1.0.0) kao besplatni softver licenciran pod BSD licencom od strane *NLnet Labsa*. Instalira se kao dio osnovnog sustava u ***FreeBSD Unixu*** počevši od inačice 10.0, te u NetBSD-u s inačicom 8.0. Također je dostupan u ***OpenBSD*** od inačice 5.6. |

**Izvor informacija: (1137), RFC1034, RFC 1035.**

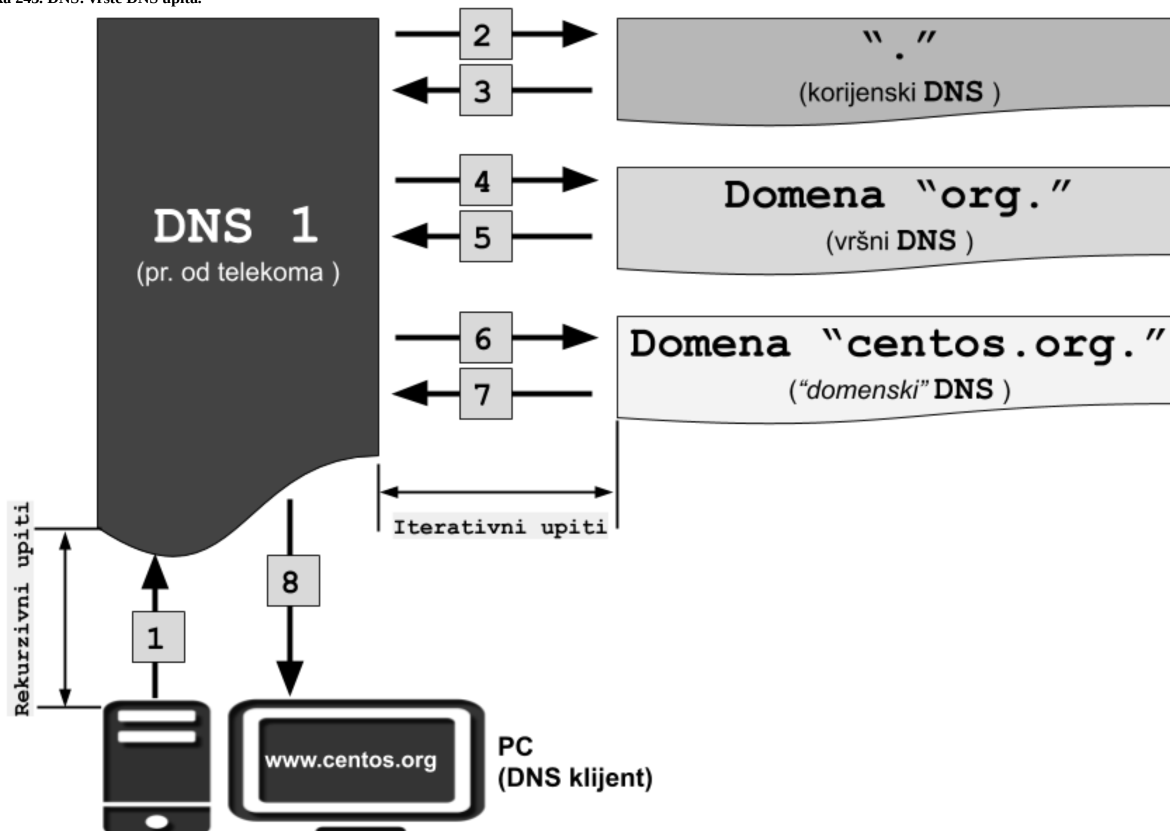
### 25.8.5.4. Vrste DNS upita

Postoji nekoliko vrsta *DNS* upita prema *DNS* poslužiteljima:

- Ne rekurzivni upit odnosno ***Non-recursive query***.
- Rekurzivni upit odnosno ***Recursive query***.
- Iterativni upit odnosno ***Iterative query***.

Kroz primjer na slici 243 pogledajmo što se događa u slučaju kada naš web preglednik želi posjetiti web stranicu odnosno adresu poslužitelja `http://www.centos.org/` kako smo uvodno objasnili u cjelini: **25.8.5**.

**Slika 243. DNS: vrste DNS upita.**




U ovom procesu spajanja web preglednika na navedenu web stranicu (adresu) se događa sljedeće, kako je vidljivo na slici 243:

1. DNS klijent (naše **PC** računalo) kontaktira nama prvi (primarni) DNS poslužitelj (koji smo dobili od našeg telekoma) s rekurzivnim upitom (*recursive query*) za http://www.centos.org/. Pošto je ovo rekurzivni upit DNS poslužitelj kojem šaljemo upit može odgovoriti sâmo na dva načina:
   a. **Poslati nam točan odgovor odnosno dati nam konkretnu IP adresu kao odgovor na naš upit.**
   b. **Poslati nam poruku o grešci.**
2. Naime kada je navedeni primarni **DNS 1** poslužitelj zaprimio naš upit, on provjerava svoju predmemoriju (*cache*) i svoje zone. Pošto on nije odgovoran za ovu domenu, a nema podatak u svojoj predmemoriji (*recimo da je to ovdje slučaj*) on šalje iterativni DNS upit (*iterative query*) na jedan od vršnih korijenskih (*root*) DNS poslužitelja (.) s istim upitom za http://www.centos.org/.
3. Korijenski DNS poslužitelj (.) ne zna točan odgovor, ali poznaje odnosno ima popise svih vršnih DNS poslužitelja, pa tako i onoga za vršnu domenu **.org**. On potom vraća odgovor za vršnu domenu (**.org.**) odnosno daje nam IP adresu DNS poslužitelja koji je odgovoran za domenu **.org**.
4. **DNS 1** poslužitelj je dobio IP adresu DNS poslužitelja za vršnu domenu **.org.** i potom šalje iterativni upit za http://www.centos.org/ na *DNS* poslužitelj koji je autoritet za domenu **.org.** u nadi kako će dobiti konačan odgovor.
5. Vršni DNS poslužitelj koji je autoritet za domenu **.org.** ne zna konačan odgovor, ali zna koji DNS poslužitelj je autoritet za traženu pôd domenu odnosno za domenu ***centos.org***, te nam vraća IP adresu tog DNS poslužitelja.
6. **DNS 1** poslužitelj sada kada ima potrebnu IP adresu DNS poslužitelja za domenu ***centos.org.***, kontaktira ga, pošto je on autoritet za domenu ***centos.org***, te njemu šalje iterativni upit za http://www.centos.org/.
7. DNS poslužitelj koji je autoritet za domenu ***centos.org.*** ima podatak o http://www.centos.org/ te vraća odgovor na **DNS 1** poslužitelj odnosno daje mu traženu IP adresu.
8. **DNS 1** poslužitelj sada ima IP adresu od http://www.centos.org/ te vraća rekurzivni DNS odgovor u kojem nama odnosno našem računalu – **PC**, daje tu IP adresu te sprema rezultate ovog upita u svoju predmemoriju (*cache*).
9. Sada naše računalo (**PC**) ima IP adresu od http://www.centos.org/ te se spaja na nju odnosno na web poslužitelj na toj IP adresi. Naše računalo sada sprema taj unos: ako ima pokrenut DNS servis ili u međumemoriju web preglednika.

**Sumirano možemo reći sljedeće:**

- Ne rekurzivna metoda odnosno ***Non-recursive query*** se odnosi na DNS upit na autoritativni DNS poslužitelj, zadužen za traženu domenu. Na ne rekurzivni upit može nam odgovoriti i predmemorijski (*caching*) poslužitelj, ako ima korektan unos u svojoj predmemoriji, bez potrebe za kontaktiranjem autoritativnog poslužitelja za traženu domenu. To je moguće, ako je njegov unos u predmemoriji vremenski valjan odnosno nije mu isteklo vrijeme života (*time-to-live*) za konkretan zapis.
- Rekurzivni upit odnosno ***Recursive query*** je onaj kod kojega DNS klijent šalje rekurzivni upit DNS poslužitelju, koji tada može proslijediti taj upit na drugi DNS poslužitelj. Tada se DNS poslužitelj ponaša kao DNS klijent jer on u svoje ime šalje DNS upit prema vršnom DNS poslužitelju u naše ime. Tipičan primjer je kućni usmjerivač (*router*) koji šalje rekurzivni upit DNS poslužitelju našeg telekoma (ili onome koji smo već konfigurirali za upotrebu). Očekivani odgovor na rekurzivni upit može biti: konačan odgovor ili greška: *NXDOMAIN (Non Existent Domain*).
- Kod **iterativnog** upita, DNS klijent koji može biti i DNS poslužitelj koji šalje upit drugom DNS poslužitelju, može slati upite na više DNS poslužitelja. Svakim odgovorom s vršnih DNS poslužitelja će se naš DNS poslužitelj uputiti na sljedeći DNS u hijerarhiji niže, na koje će on slati upite, sve dok ne dođe do konačnog DNS poslužitelja od kojega će dobiti završni odgovor.

**Što se događa tijekom otvaranja odnosno učitavanja svake moderne web stranice; primjerice https://www.centos.org?**
Svaka moderna web stranica sadrži veliki broj različitih *objekata*, od kojih se određeni broj njih uopće ne mora nalaziti na našem web poslužitelju, na kojem se nalazi i sâma web stranica; odnosno čak ne mora pripadati našoj DNS domeni. Pod objektima o kojima ovdje govorimo uglavnom mislimo na web poveznice, slike, animacije, video materijale te druge načine referenciranja na (udaljeni) sadržaj. Dakle prije nego naš web preglednik uopće pokuša pristupiti web stranici, on prvo mora od *DNS* poslužitelja saznati IP adresu tražene (željene) stranice odnosno domene, za koju on zapravo traži **DNS A** zapis/unos.
O ponašanju operativnog sustava ovisi hoće li naše računalo slati DNS upite (i s koliko ponavljanja) za traženu domenu na sve nama konfigurirane DNS poslužitelje istovremeno, ili sâmo na prvi primarno definirani, pa tek, ako se od njega ne dobije odgovor, na sljedeće definirane u nizu. U Linuxu je standardno kontaktiranje prvo primarnog DNS poslužitelja (osim ako koristimo lokalno instalirani DNS servis). Tek nakon dobivene IP adrese, stranici se može po prvi puta pristupiti. Kako se HTML stranica učitava (u tom se trenutku ne nužno i prikazuje), ako se unutar nje nalaze bilo kakvi objekti s drugih domena, za svaki od tih objekata pojedinačno se ponovno mora od DNS poslužitelja saznati na kojim IP adresama se oni nalaze.



***Napredni DNS poslužitelji mogu i filtrirati (zabranjivati) određene domene; pr. reklamne, nepoželjnog sadržaja, lažnih vijesti, … . Ovakvi DNS poslužitelji (pr. AdGuard) održavaju listu stotina tisuća ili milijuna potencijalno nepoželjnih domena koje mogu blokirati (naravno, ako ih koristimo). Oni nam u pravilu za svaku blokiranu domenu vraćaju IP adresu: `0.0.0.0` koja našem računalu nalaže da niti ne pokuša uspostaviti konekciju na tu adresu, kako je definirano u RFC 5735, sekcija: 3. te RFC3513, sekcija 2.5.2. Napredni DNS poslužitelji koriste anycast IP adrese na internetu.***

Na kraju, tek kada su sve IP adrese poznate; odnosno kada smo ih saznali od DNS poslužitelja, web stranica se stvarno i počinje učitavati, odnosno počinje se dohvaćati njen sadržaj, a koji se vjerojatno i nalazi raspršen u više desetaka domena.
Sa sljedećim nîzom naredbi pogledajmo na koliko raznih domena se nalazi raštrkan sadržaj gore navedene stranice:

```
wget -qO- https://www.centos.org | grep -Eoi '<a [^>]+>' | grep -Eo 'href="[^\"]+"' |
grep -Eo '(http|https)://[^/"]+'
```

Naime svaka poveznica u HTML dokumentu zahtjeva DNS razlučivanje domene u IP adresu. Ako se sadržaj dohvaća s već naučene domene (domena→IP adresa) tada se potpuno DNS razlučivanje ne radi ponovno jer se zapis dohvaća iz lokalne međumemorije računala (lokalna DNS međumemorija sustava ili web preglednika) ili s prvog trenutno aktivnog DNS poslužitelja koji koristimo, a koji odmah prepoznaje pripadajuće domene-IP adrese jer ih čuva u svojoj priručnoj memoriji.
Pogledajmo i naredbu (u jednom retku) koja će nam prebrojati sve pojedinačne domene koje su definirane na navedenoj stranici:

```
wget -qO- https://www.centos.org | grep -Eoi '<a [^>]+>' | grep -Eo 'href="[^\"]+"' |
grep -Eo '(http|https)://[^/"]+' | sort |uniq |wc -l
85
```

Ovdje ih konkretno ima **85**, što je prosječan broj za većinu web stranica.



U slučajevima kada je naš usmjerivač (router) od strane Linuxa odabran kao primarni odnosno aktivni DNS poslužitelj, on na sebe preuzima zadaću provjere svih razlučivanja IP adresa, od domenskih imena. Odnosno na njega (odabrani DNS) se nadalje šalju svi DNS upiti.

Zamislimo okruženje u kojem svaki korisnik otvara deset ovakvih stranica te, ako imamo deset računala na mreži s istom količinom otvorenih web stranica, to znači do 8.500 DNS upita, što je procesorski i memorijski zahtjevno za usmjerivač.
S druge strane upotreba brzog lokalnog DNS poslužitelja u usmjerivaču čini ga dobrim odabirom, ako je dovoljno snažan, jer se ponovljeni DNS upiti s drugih računala mogu dohvaćati iz njegove priručne memorije, a ne ponovno s interneta. S time se smanjuje broj DNS upita prema vanjskim DNS poslužiteljima (na internetu) koji mogu biti limitirajući faktor, ako u sve uračunamo i ostali promet kao i sve ostale otvorene konekcije prema vanjskim poslužiteljima na internetu. Nadalje, pitanje je i brzine odziva DNS poslužitelja koji koristimo.

Naime nisu svi DNS poslužitelji iste brzine, odnosno ne odgovaraju svi jednako brzo ili do njih eventualno možemo imati kašnjenje u mreži, odnosno latenciju, koju nazivamo ***DNS latencija***. Stoga ponekad upotreba DNS poslužitelja koji smo dobili od strane telekoma čiju uslugu koristimo, ipak nije optimalan izbor.



Latencije DNS poslužitelja su svjesni i veliki igrači poput tvrtki ***Google***, ***CloudFlare*** i drugih, te nam oni nude svoje DNS poslužitelje na korištenje, a koji vrlo često znaju biti brži od navedenih DNS poslužitelja telekoma. ***Google*** nam daje nekoliko DNS poslužitelja: **8.8.8.8** te **8.8.4.4** i **4.4.4.4** dok nam tvrtka ***CloudFlare*** nudi DNS poslužitelje **1.1.1.1** i **1.0.0.1**. Naravno ovdje postoje i mnogi drugi DNS poslužitelji od drugih velikih i/ili specijaliziranih tvrtki poput ***AdGuard*** i sličnih.

**Dodatne informacije o domenama.**
Mnoge informacije o domenama se pri kupnji domene moraju dostaviti autoritetu za Domene i Internet adrese (*Internet Corporation for Assigned Names and Numbers* - **IANA**). Stoga ih je i moguće pretraživati. Konkretno s naredbom `whois`. Prvo instalirajmo ovaj program odnosno naredbu `whois`, na sljedeći način:

```
yum -y install whois
```

Sada probajmo doznati više informacija, primjerice o domeni: ***centos.org*** (informacije koje ćemo dobiti pogledajte sami):

```
whois centos.org
```

Informacije koje ćemo ovdje dobiti su za fizičku ili pravnu osobu vlasnika domene: ime i adresa, registracijski broj (*IANA ID*), datume kreiranja i isteka domene te adrese DNS poslužitelja zaduženih za konkretnu domenu, kao i druge detalje.

**Izvori informacija: (1135),(1136), `man whois`, RFC1034, RFC 3513, RFC 5735.**

## 25.8.5.5. DNS zapisi (*DNS records*)

DNS zapisi (Engl. *DNS records*) unutar zonske datoteke (Engl. *Zone files*) se koriste za razlučivanje domene ili nekog drugog resursa i njegove pripadajuće IP adrese, na DNS poslužitelju. Postoji više vrsta DNS zapisa:

- Oni koji definiraju sâmu domenu te oni koji se povezuju s imenom računala.
- Oni koji pokazuju na DNS poslužitelje.
- Oni koji pokazuju na e-mail poslužitelje i/ili oni koji pokazuju na druge resurse.

Dalje u tekstu obradit ćemo nekoliko osnovnih zapisa. Primjeri koji slijede bit će odrađeni korištenjem naredbi `nslookup` i `dig`. Obje naredbe u ***Redhat/CentOS*** linuxu dolaze u paketu `bind-utils`. Naredba `nslookup` postoji i u operativnom sustavu *Windows*.

**Izvor informacija: (1137).**

### 25.8.5.5.1. Autoritativni i neautoritativni odgovori

Za bilo koji zahtjev koji pošaljemo prema DNS poslužiteljima, uvijek možemo dobiti dvije vrste poruka neovisno o vrsti zapisa koji tražimo, a to su autoritativni ili neautoritativni odgovori.

**Neautoritativni odgovor** dobivamo od DNS poslužitelja koji nam može ponuditi odgovor jer ga: ili ima u svojoj predmemoriji (*cache*) ili je on u naše ime postavljao iterativne upite te je u konačnici doznao odgovor od vršnih DNS poslužitelja.
**Autoritativni odgovor** dobivamo sâmo od DNS poslužitelja koji je autoritet za domenu za koju tražimo upit.

Primjer ćemo dati na upitu za **A** zapis (o kojem ćemo pričati ubrzo). Dakle pošaljimo DNS upit za sve **A** zapise (`-type=a`) za domenu `centos.org` pomoću naredbe `nslookup` na sljedeći način:

```
nslookup -type=a centos.org

Server:         192.168.1.1
Address:        192.168.1.1#53

Non-authoritative answer:
Name:   centos.org
Address: 81.171.33.201
```

To znači da smo dobili odgovor od našeg usmjerivača koji ima IP adresu: 192.168.1.1, a koji je imao već spreman odgovor u međumemoriji, te nam odgovorio kako je za domenu `centos.org` zadužena IP adresa: `81.171.33.201`. Ali nam on kaže i kako on nije autoritativni DNS za tu domenu i to s porukom: `Non-authoritative answer`. Ako zbog nekog razloga baš želimo autoritativan odgovor, moramo upit poslati na DNS poslužitelj koji je autoritet za domenu `centos.org`.

Točnije moramo poslati DNS upit na DNS koji je zadužen za tu domenu. Konkretno je za domenu `centos.org` zadužen DNS poslužitelj (to smo saznali iz NS zapisa u sljedećim poglavljima): `ns1.centos.org`.

Pogledajmo sada ovakav upit direktno na navedeni DNS poslužitelj koji je autoritativan za konkretnu domenu:

```
nslookup -type=a centos.org ns1.centos.org

Server:         ns1.centos.org
Address:        8.43.84.215#53
Name:    centos.org
Address: 81.171.33.202
```

Vidljivo je kako smo dobili odgovor **bez** poruke `Non-authoritative answer` što znači kako je odgovor autoritativan. Kako pronaći koji DNS poslužitelj je zadužen (autoritativan) za koju domenu, vidjet ćete kasnije u tekstu.

**Izvor informacija: (1137).**

### 25.8.5.5.2. A zapis

**A** zapis je osnovni DNS zapis koji se koristi za povezivanje domene ili pôd domene i njene pripadajuće IP adrese. Dakle ovaj zapis je praktično pokazivač za razlučivanje domene ili punog imena računala odnosno imena računala s njegovom domenom (Engl. *Fully Qualified Domain Name*), na njegovu pripadajuću IP adresu. Jedina dodatna opcija u ovom unosu je ***TTL*** (*time-to-live*) polje odnosno definicija vremena nakon kojeg će drugi DNS poslužitelji koji su dohvatili ovaj unos s ovog autoritativnog DNS poslužitelja za konkretnu domenu i drže ga u svojoj predmemoriji, morati isti osvježiti. Dakle ***TTL*** možemo promatrati kao vrijeme života konkretnog unosa u predmemoriji DNS poslužitelja. Pogledajmo primjer **A** zapisa za: `centos.org`

```
centos.org.          303      IN       A        81.171.33.202
centos.org.          303      IN       A        35.178.203.231
centos.org.          303      IN       A        81.171.33.201
```

Opis polja (od lijevo na desno):

- `centos.org.` – ovo je domena odnosno ime računala.
- `303` – ovo je vrijeme života ovog unosa u sekundama; konkretno 303s= 5 minuta (dakle ovo je **ttl** vrijednost).
- `IN` – ovo je oznaka kategorije: `IN` je standardna kategorija i znači INternet (*class*).
- `A` – je oznaka označava kako je ovo `A` zapis (*rr - Resource record*).
- `81.171.33.202` – ovo je pripadajuća IP adresa za ovo računalo odnosno domenu (konkretno prva vidljiva u nîzu).

Potražimo i sâmi **DNS A** zapise za domenu **`centos.org`**, ali sada s programom `dig` na sljedeći način:

```
dig centos.org A
```

Isto možemo dobiti i s naredbom `nslookup` i opcijom `type` uz navođenje vrste zapisa `a` odnosno konkretno: `type=a`

```
nslookup -type=a centos.org
```

U svakom slučaju ovdje imamo višestruke **A** unose zbog zalihosti odnosno redundancije, što za (jednostavnije) domene obično nije slučaj. U ovakvom slučaju, kada dobijemo višestruke **A** unose odnosno višestruke IP adrese za određenu (istu) domenu, web preglednici od sustava odabiru sâmo jednu od njih i to obično onu prvu na listi (ovisno o implementaciji), kojoj u konačnici i pristupaju. U slučaju da je odabrana IP adresa nedostupna, web preglednik će potom koristiti prvu sljedeću IP adresu koja je navedena odnosno dobivena.



***Sa svakim novim DNS upitom, DNS poslužitelj će nam odgovoriti tako da će raditi permutaciju navedene liste IP adresa, tako da će svaki puta neka druga od IP adresa iz liste biti postavljena kao prva u listi odnosno ona koja će u konačnici i biti odabrana. S time se postiže osnovna raspodjela (Load Balancing) opterećenja odnosno prometa prema odredišnim IP adresama DNS poslužitelja prema tzv. Round Robin[(912)] mehanizmu raspodjele unutar DNS protokola.***

Naime moguće je da određena domena ima više od jednog poslužitelja zbog zalihosti, a također je dodatno moguće da se iza svake od tih IP adresa krije ***Load Balancer*** iza kojega se opet nalazi cijeli nîz poslužitelja. Pogledajte poglavlje o ***VRRP*** protokolu i to dio o **keepalived** servisu:**23.4.5. Redundancija na OSI sloju tri (OSI 3) i *VRRP* protokol.**

.

**Izvori informacija: (912), `man nslookup`, `man dig`, RFC 1034, RFC 1035.**

### 25.8.5.5.3. AAAA zapis

**AAAA** zapis je identičan **A** zapisu, ali u njemu se ne koriste IPv4 već IPv6 IP adrese.

**Izvori informacija: (1138), RFC 1034 i RFC 1035, RFC 3596.**

### 25.8.5.5.4. PTR zapis

**PTR** zapis (Engl. *Pointer record*) je praktično suprotan od **A** zapisa. Naime **A** zapis nam govori koja je IP adresa određene domene, dok nam **PTR** zapis govori upravo suprotno. Dakle pomoću njega možemo saznati koja domena stoji iza neke IP adrese; točnije koji poslužitelj odnosno njegovo FQDN ime. Ova funkcionalnost se koristi često kod e-mail poslužitelja za zaštitu od neželjenih e-mail poruka (*spam*), kako bi se provjerilo je li pristigla pošta (*e-mail*) stvarno došla s e-mail poslužitelja koji je zadužen za određenu domenu, prema IP adresi domene. **PTR** zapis se stoga definira od strane vlasnika IP adrese određenog poslužitelja na domeni ili vlasnika domene. Upit s kojim iz IP adrese dobivam ime domene, zove se reverzni DNS upit (Engl. *Reverse DNS lookup*). Pogledajmo primjer *PTR* zapisa:

```
122.110.20.10.in-addr.arpa. IN PTR server1.domena.com
```

Vidljivo je nekoliko stvari:

- IP adresa je s lijeve strane i upisana je unatrag u odnosu na **A** zapis (zapis predstavlja IP: `10.20.110.122`).
- IP adresa završava s: *`in-addr.arpa.`*
- FQDN ime poslužitelja (s domenom) je s desne strane (`server1.domena.com`), a može biti povezana u drugoj domeni od one na kojoj se izvorno koristi; npr. donja IP* adresa je registrirana kao: **ns3.centos.org** u **centos.org**.

Do ***PTR*** zapisa možemo doći pomoću naredbe `dig`; primjerice za IP adresu: **88.208.217.170*** bi to dobili sa:

```
dig -x 88.208.217.170
170.217.208.88.in-addr.arpa. 86337 IN   PTR   server88-208-217-170.live-servers.net.
```

**Izvori informacija: (1138), RFC 1034 i RFC 1035.**

### 25.8.5.5.5. CNAME zapis

**CNAME** zapis (Engl. *Canonical name*) se koristi kada želimo napraviti poveznicu s postojeće domene na neku pôddomenu ili jednog imena računala na drugo. Dakle **CNAME** koristimo kada želimo asocirati postojeću domenu s postojećim **A** zapisom na novu pôddomenu ili s postojećeg računala (*hosta)* na drugo. **CNAME** je definiran u RFC 1034 te u RFC 2181 - poglavlje 10. **CNAME** se ne koristi za asociranje nove pôd domene na cijelo stablo postojeće domene (za to pogledajte **DNAME** zapis).

Kao primjer ćemo uzeti domenu `centos.org` za koju postoji **A** zapis:

```
centos.org.          3600    IN     A      81.171.33.202
```

Sada ako želimo kreirati pôddomenu imena: `http://www.centos.org/` koja će pokazivati na vršnu domenu `centos.org`. To će izgledati ovako:

```
www.centos.org.         3600    IN   CNAME       centos.org
```

Isto tako smo mogli kreirati i ime računala (*hostname)* unutar iste domene (kada bi bili vlasnici ove domene). Neka to bude *hostname* (ime računala): `server1` kako bi nam bilo razumljivije. Pogledajmo sada, kako bi izgledao ovakav unos:

```
server1.centos.org.     3600  IN       CNAME       centos.org
```

Dodatno je vidljivo kako i **CNAME** ima svoju **TTL** (*time-to-live*) vrijednost.

U oba gore navedena primjera, adrese/*hostname*/domene: `http://www.centos.org/` i `server1.centos.org` pokazuju na isti **A** unos `centos.org` koji nas na kraju dovodi do iste IP adrese: `81.171.33.202`

Ograničenja **CNAME** zapisa su:

- **CNAME** zapis mora uvijek pokazivati na domenu, a nikada na IP adresu poput **A** zapisa.
- **CNAME** zapis može pokazivati na drugi **CNAME**, ali to treba izbjegavati zbog lošije efikasnosti.
- **MX** i **NS** zapisi nikada ne smiju pokazivati na **CNAME** zapis, već na domenu (na **A** zapis).

Domene koje se koriste za e-mail ne bi trebale koristiti CNAME zapise zbog različitog ponašanja ovisno o inačici i vrsti e-mail poslužitelja.

**Izvori informacija: RFC 1034, RFC 1035 i RFC 2181.**

### 25.8.5.5.6. DNAME zapis

**DNAME** zapis (Engl. *Delegation Name record*) je definiran u **RFC 6672**. **DNAME** zapis kreira pokazivač za cijelo stablo novog domenskog stabla, za razliku od **CNAME** koji se koristi za jedan unos, a ne za sve pôd domene postojeće domene za koju je on pokazivač. Pogledajmo primjer:

```
server1.domena-1.com. 14400   DNAME   www.domena-2.com.
www.domena-2.com.     14400   A       148.17.100.23
```

Ako želimo pristupiti na: `http://www.domena-2.com` dobiti ćemo IP adresu `148.17.100.23` iz **A** zapisa.

Ali ako želimo pristupiti na `server1.domena-1.com` iako nemamo **A** zapis, pošto **DNAME** može pokazivati na cijelo stablo novog domenskog stabla te je `domena-1.com` sada povezana s `domena-2.com` tada i `server1` koji je vezan za `domena-1.com` sada ima vezu i s drugom domenom `domena-2.com` pa će tako odgovor biti IP adresa iz druge domene: `148.17.100.23`.



I za **DNAME** zapise postoji ***TTL*** vrijeme.

**Izvori informacija: RFC 6672.**

### 25.8.5.5.7. MX zapis

**MX** zapisi (*Mail Exchange record*) se koriste kako bi se unutar domene definirali poslužitelji elektroničke pošte koji su zaduženi za tu domenu. Moguće je imati više ovih zapisa odnosno više poslužitelja elektroničke pošte unutar domene, svaki sa svojim prioritetom koji se može definirati. Pomoću ovog zapisa servisi elektroničke pošte mogu saznati koji poslužitelj elektroničke pošte je potrebno kontaktirati za određenu domenu prilikom slanja elektroničke pošte (e-mailova).
**MX** zapis je opisan u **RFC 974**. Za točnu proceduru kako klijent elektroničke pošte odnosno ***SMTP*** sustav može pronaći poslužitelj elektroničke pošte zadužen za određenu domenu, pogledajte **RFC 5321 - stranica 69**.



Pogledajte i poglavlje:
**25.8.8. Elektronička pošta (e-mail).**

Pogledajmo listu ***MX*** zapisa s domene: **centos.org** s programom dig na sljedeći način:

```
dig centos.org MX +noall +answer
```

Odgovor smo filtrirali s tipom zapisa ***MX*** i prekidačima +noall i +answer. Dobit ćemo listu poslužitelja e-pošte (ako ih ima više) odnosno minimalno sâmo jedan definiran za domenu **centos.org**. To je konkretno sljedeći poslužitelj e-pošte:

```
centos.org.             3600    IN      MX      10 mail.centos.org.
```

Predzadnji broj **10** označava koji poslužitelj e-pošte ima veći prioritet (0 je najveći), u slučaju kada imamo definirano više poslužitelja e-pošte, što je čest slučaj kod većih domena koje trebaju zalihost. Vrijednost 3600 je TTL (*time-to-live*) vrijednost.

**Izvori informacija: RFC 974, RFC 1035, RFC 5321, RFC 7505.**

---

### 25.8.5.5.8. NS zapis

**NS** zapis (Engl. *Name Server*) je zadužen za povezivanje domene s autoritativnim DNS poslužiteljima za tu domenu.
Dakle on nam daje listu autoritativnih *DNS* poslužitelja zaduženih za određenu domenu.
Pogledajmo listu **NS** zapisa odnosno listu DNS poslužitelja zaduženih (autoritativnih) za domenu centos.org

```
dig centos.org NS +noall +answer
centos.org.             14400    IN      NS      ns3.centos.org.
centos.org.             14400    IN      NS      ns4.centos.org.
centos.org.             14400    IN      NS      ns1.centos.org.
```

Vidljivo je kako su za domenu centos.org zaduženi sljedeći DNS poslužitelji (što znamo jer imaju NS oznaku):

- ns1.centos.org i ns3.centos.org te ns4.centos.org

U drugom stupcu broj: 14400 (14400s=240min=4 sata) predstavlja *TTL* vrijednost (vrijeme života zapisa).

**Izvori informacija: man dig, RFC 1034, RFC 1035.**

---

### 25.8.5.5.9. SOA zapis

**SOA** (Engl. *Start of authority*) zapis predstavlja autoritativne informacije o DNS zoni odnosno domeni, a koja uključuje:

- Primarni autoritativni DNS poslužitelj zadužen za zonu (domenu).
- E-mail adresu vlasnika (administratora) domene (zone).
- Revizijski broj domene i druge razno razne brojače i vremenske oznake (*tajmere)* definirane za domenu, koji definiraju vremena potrebna za osvježavanje domene, vremena za ponavljanje upita na domenu i drugih.

Prema definiciji dozvoljen je sâmo jedan ***SOA*** domenski zapis (*Resource record* (***RR***)) unutar svake zone (domene) i on mora biti prvi zapis u zoni (domeni). Dakle ***SOA*** zapis sadrži informacije nužne za rad domene.
Nakon ovog zapisa slijede svi ostali zapisi (***A***, ***CNAME***, ***AAAA***, ***NS***, ***MX***, ***TXT***, ***DNSKEY***, …).

**_SOA_ zapis je najvažniji dio _zonske_ datoteke.**

***SOA*** zapis za željenu domenu možemo provjeriti na dva načina. Provjerit ćemo domenu redhat.com i to direktno od autoritativnog DNS poslužitelja za tu domenu ns1.redhat.com i to samo SOA zapise.
**1.** To možemo vidjeti pomoću naredbe nslookup

```
nslookup -type=soa centos.org ns1.centos.org
```

Odgovor je:

```
Server:         ns1.centos.org
Address:        8.43.84.215#53

centos.org
        origin = ns1.centos.org
        mail addr = hostmaster.centos.org
        serial = 202009
        refresh = 28800
        retry = 7200
        expire = 2400000
        minimum = 3600
```

**2.** Napravimo isto i pomoću naredbe `dig`

```
dig @ns1.centos.org centos.org SOA +noall +answer
```

Odgovor je :

```
centos.org. 14400 IN SOA ns1.centos.org. hostmaster.centos.org. 202009 28800 7200 2400000 3600
```

Objašnjenje polja slijedi (od lijeva na desno):

- Polje **TTL** (ovdje je to `14400` sekundi) je vrijeme koje DNS klijenti smiju čuvati ovaj zapis u predmemoriji (*cache*). Ovo se odnosi i na druge DNS poslužitelje koji ionako pretražuju druge domene kao DNS klijenti te mogu čuvati lokalno ove unose, točno koliko vremena je definirano ovdje.
  **→ SLAVE DNS poslužitelj se uopće ne obazire na ovo polje**.
- `origin` - on predstavlja autoritativni DNS poslužitelj za ovu domenu (onaj koji sadrži *zonsku* datoteku).
- `mail addr` - ovo je adresa elektroničke pošte (e-mail) administratora ove domene.
- `serial` - *nije serijski broj* već revizijski broj, koji se povećava svaki puta kada dođe do nekih promjena u domeni odnosno u zoni. On je prema tome indikator promjena: svaka promjena bilo kojeg unosa u domeni (***A***, ***CNAME***, ***PTR***, …) ili dodavanje ili brisanje nekog od njih, mijenja ovaj broj. Ovdje je to vrijednost: `202009`.
- `refresh` - ovo je vrijeme u sekundama kada **SLAVE** DNS poslužitelj(i) ove domene moraju provjeriti **MASTER** DNS poslužitelj ove domene: je li došlo do nekih promjena. Dakle svakih `28800` sekundi (u ovom slučaju) **SLAVE** DNS ove domene mora provjeriti **MASTER** DNS i provjeriti je li se povećao `serial` broj što indicira kako je došlo do nekih promjena.
- `retry` - u slučajevima kada je **SLAVE** DNS poslužitelj pokušao kontaktirati **MASTER**a i nije uspio, ponovno mora pokušati tek nakon vremena definiranog u ovom polju. U ovom slučaju je to nakon `7200` sekundi.
- `expire` - ovo je vrijeme koje će **SLAVE** DNS poslužitelj čuvati domenske zapise (zone datoteku) lokalno.
- `minimum` je standardno minimalno vrijeme koje će **SLAVE** DNS poslužitelj čuvati lokalnu zonsku datoteku (cijelu domenu). Ovdje je to postavljeno na `3600` sekundi.

**Izvori informacija: RFC 1996, RFC 1035 (sekcija 3.3.13), RFC 2308, RFC 2136, RIPE-203.**

#### 25.8.5.5.10. Drugi zapisi

Postoji i cijeli (dugački) nîz drugih zapisa, poput:

- ***TXT*** zapisa - za dodavanja bio kakvog teksta - za različite namjene.
- ***SRV*** (*Service locator*) zapisa - za pridruživanje servisa određenoj domeni (Pr. HTTP protokol preko TCP transportnog protokola na portu 80 (web servis)). Navedenih servisa se može definirati veliki broj, poput:
  - _http - HTTP
  - _ftp - FTP
  - _ldap - LDAP
  - A kao protokoli za transport mogu biti označeni:
    - `_tcp` - TCP protokol.
    - `_udp` – UDP protokol.
- ***NAPTR*** (*Naming Authority Pointer*) zapisa - za povezivanje brojeva telefona, kao i mnogi drugi zapisi

**Izvori informacija: RFC 1034, RFC 1035.**

### 25.8.5.6. Zone i zonske datoteke

*Zonske* (domenske) datoteke sadrže DNS zapise (Engl. *Resource Records*) koji opisuju domenu ili pôddomenu. Format *zonskih* datoteka je definiran u **RFC 1035** standardu. Zonska datoteka mora sadržavati minimalno sljedeće podatke:

- Podatke koji definiraju ovu zonu (domenu) i njene parametre (**SOA** zapis). *SOA* (*Start of Authority*) definira i daje autoritativne informacije o domeni, e-mail adresi domenskog administratora, serijski broj domene i druge parametre sâme domene.
- Autoritativne podatke za sva računala u ovoj zoni (domeni): obično su to **A** (IPv4) ili **AAA** (IPv6) zapisi odnosno zapisi koji razlučuju ime računala (domene) u IP adresu [*hostname-to-IP]*.
- Druge podatke koji su važni za zonu (domenu), poput popisa DNS poslužitelja (**NS** zapisi), email poslužitelja (**MX** zapisi) i slično.
- U slučaju kada imamo i pôddomenu, potrebno je definirati DNS poslužitelje zadužene za te pôddomene (**NS** zapisi).
  - Dodatno za pôddomene potrebni su i **A** ili **AAA** zapisi računala (*hostova*) unutar pôddomene.

**DNS** koristi *TCP* kao transportni protokol na portu **53** kod transfera zona (*zone transfer*). Pošto je konzistencija podataka vrlo važna, kod transfera zona, a koja se događa između DNS poslužitelja, uvijek se koristi *TCP* za kao transportni protokol za razliku od normalne DNS komunikacije koja u pravilu koristi *UDP* protokol.

#### 25.8.5.6.1. Zone update

Prema dizajnu DNS sustava, promijene unutar zone (domene) propagiraju se prema drugim DNS poslužiteljima pomoću kompletnog transfera zone (Engl. *Zone transfer*) (**AXFR**). Povećanjem zona i sâmim time količine podataka koje treba prenijeti uvedena je i inkrementalna metoda prijenosa zona (**IXFR**) te *notify* poruka za osvježavanje tijekom prijenosa.

#### 25.8.5.6.2. Full Zone Update (AXFR)

Prema izvornoj specifikaciji DNS protokola (**RFC 1034** i **RFC 1035**) definirano je kako **SLAVE** DNS poslužitelj može povući domenu (zonu) s **MASTER** DNS poslužitelja. Vremenski intervali između ovih povlačenja zone (domene), a koje **SLAVE** poslužitelj mora redoviti raditi, je definiran u **SOA** zapisu domene i to pod opcijom `refresh`.
Proces povlačenja (kopiranja) zone (domene) se postiže tako što **SLAVE** DNS šalje upit na **MASTER** DNS i od njega traži **SOA** zapis. Ako je u **SOA** zapisu unutar opcije `serial` vrijednost promijenjena, što znači kako je došlo do nekih promjena, **SLAVE** DNS traži od **MASTER**a transfer zone (domene) pomoću **AXFR** zahtjeva. Dakle u slučaju kada se radi neka promjena unutar domenskih zapisa, mora se promijeniti (povećati) i vrijednost opcije `serial`.
Za transfer zone (**AXFR**) koristi se isključivo **TCP** protokol na portu **53**. Osim standardne metode transfera cijele zone, koju odrađuju DNS poslužitelji za potrebe izrade sigurnosne kopije, transfer cijele zone (domene) moguće je napraviti i ručno upotrebom naredbe `dig`. Jedino što je potrebno su prava (ovlasti) na to. Naime potrebno je na DNS poslužitelju s kojega ćemo raditi izradu sigurnosne kopije (*backup*) dozvoliti izradu sigurnosne kopije za IP adresu računala s kojega ćemo to raditi. Konkretna konfiguracija dodavanja ovih prava ovisi o implementaciji, a obično je to neka opcija poput "*Custom External Servers*". U primjeru u nastavku ćemo pretpostaviti kako imamo sva potrebna prava pristupa za dohvaćanje svih zapisa.
Prekopirajmo zonu (domenu), a u primjeru će to biti neka naša zona (domena) imena `local.org`

```
dig  local.org @192.168.100.254 AXFR > local.org.export
```

Opis navedene naredbe:

- Zona (domena) koju kopiramo je `local.org`
- Autoritativni DNS poslužitelj s kojega kopiramo zonu je `192.168.100.254`
- `AXFR` označava kako radimo transfer cijele zone.
- Datoteka u koju snimamo zonu (domenu) se zove `local.org.export`

Pogledajmo ovu datoteku:

```
cat local.org.export
```

Sadržaj ove datoteke je sljedeći:

```
; <<>> DiG 9.9.4-RedHat-9.9.4-18.el7 <<>> local.org @192.168.100.254 AXFR
;; global options: +cmd
local.org.              7200    IN      SOA     ns.example.com.
hostmaster.example.com. 1 10800 3600 6048
local.org.              7200    IN      NS      ns.server1.addr.
local.org.              7300    IN      NS      ns.server2.addr.
local.org.              7200    IN      SOA     ns.example.com.
hostmaster.example.com. 1 10800 3600 6048
local.org.              7200    IN      NS      ns.server1.addr.
local.org.              7300    IN      NS      ns.server2.addr.
;; Query time: 7 msec
;; SERVER: 192.168.100.254#53(192.168.100.254)
;; WHEN: Fri Aug 25 23:47:06 UTC 2017
;; XFR size: 2 records (messages 2, bytes 324)
```

Napomena:

- Format zapisa je takav u kojem nakon svih DNS zapisa slijede zadnja tri zapisa, koja su zapravo kopirana prva tri zapisa i to, ako je sve prošlo kako treba.
- `;` i `;;` su linije s komentarima, koje se ionako ignoriraju.

Dakle u našem slučaju imali smo samo tri zapisa:

1. Prva tri reda: **SOA**, pa prvi i drugi **NS** zapis za domenu `local.org`
2. Potom slijede ponovljena ista ta tri prva reda, kao potvrda kako je sve prošlo u redu.

**Izvori informacija: RFC 1034, RFC 1035.**

### 25.8.5.6.3. Incremental Zone Update (IXFR)

Tijekom prijenosa velikih zonskih datoteka odnosno domena s velikim brojem unosa, potrebno je i puno vremena i mrežne propusnosti kako bi se ova procedura odradila. U slučajevima kada se unutar zone (domene) mijenjalo vrlo malo unosa, potpuni transfer zone (***AXFR***) postaje nepotreban i neefikasan.
Prema novom standardu (**RFC 1995**) postoji mogućnost inkrementalnog prijenosa zone (**IXFR**) dakle prijenosa u kojemu se prenose samo oni zapisi koji su se mijenjali.

Procedura je identična kao i za ***AXFR***:

- **SLAVE** traži **SOA** zapis od **MASTER**-a svakih ***x*** sekundi, definiranih u polju **refresh** iz **SOA** zapisa.
- Ako se vrijednost *refresh* povećala u odnosu na onu koju **SLAVE** ima, **SLAVE** prema **MASTER**-u šalje zahtjev za transfer zone (domene) te mu indicira, kako traži *IXFR* način transfera. Ako i **MASTER** i **SLAVE** podržavaju **IXFR** način transfera (prijenosa), kreće se s *IXFR* metodom, a ako jedan od njih ne podržava ovakav transfer, nastavlja se s normalnim odnosno potpunim *AXFR* transferom.

Za **IXFR** prijenos zone, koristi se također **TCP** protokol na portu **53**.
**IXFR** metoda se, ako je podržava DNS poslužitelj, omogućava:

- Ili unutar samog DNS poslužitelja, kao parametar.
- Ili unutar konfiguracijske datoteke za DNS poslužitelj.

---

### 25.8.5.6.4. Notify (NOTIFY)

Prema standardu **RFC 1912** definiran je interval osvježavanja podataka o zoni (domeni), koji je definiran u **SOA** zapisu, kao opcija `refresh`. To znači kako u najgorem slučaju, ako je primjerice `refresh` interval 12 sati, bilo kakve promijene neće biti propagirane odnosno vidljive upravo 12 sati. To je zato što niti jedan drugi DNS poslužitelj, a poglavito **SLAVE** DNS poslužitelji neće biti dužni osvježiti svoje stanje za ovu domenu navedenih 12. sati. Vrlo često to i nije dobro jer se promjene na aktivnijim domenama mogu događati puno brže. Prema novijem standardu **RFC 1996** moguće je da **MASTER** šalje posebnu **NOTIFY** poruku prema svim **SLAVE** DNS poslužiteljima, kada je došlo do neke promjene, iako se ona dogodila unutar vremenskih okvira `refresh` opcije.

Ako **SLAVE** DNS podržava ovu opciju on će nakon što je primio *NOTIFY* poruku zatražiti zadnji **SOA** zapis od svog **MASTER** DNS poslužitelja. Kako je došlo do promjene i `serial` unutar *SOA* zapisa se povećao, **SLAVE** će zatražiti *zone transfer* na jedan od dva načina:

- **AXFR** odnosno puni prijenos zone.
- **IXFR** odnosno inkrementalni prijenos zone.

**NOTIFY** ima nekoliko opcija i parametara, koji se konfiguriraju na DNS poslužitelju.

**Izvori informacija: RFC 1034, RFC 1035, RFC 1912, RFC 1996.**

### 25.8.5.7. Izgled DNS paketa na mreži

## Pogledajmo jedan DNS upit na razini mrežnog paketa

Upit koji ćemo poslati je upotrebom naredbe `nslookup` na slijedeći način. Pri tome šaljemo klasični DNS upit:

```
nslookup www.opensource-osijek.org
```

S gornjom naredbom šaljemo DNS paket na mrežu. Navedeni DNS paket prema ***OSI*** slojevima izgleda ovako:

```
(1) Frame 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0
(2) Ethernet II, Src: (2c:d4:44:b3:5f:b6), Dst: (d4:ca:6d:34:83:23)
(3) Internet Protocol Version 4, Src: 192.168.1.20, Dst: 192.168.1.1
(4) User Datagram Protocol, Src Port: 62928, Dst Port: 53
     Domain Name System (query)
       Transaction ID: 0x0002
       Flags: 0x0100 Standard query
       Questions: 1
       Answer RRs: 0
       Authority RRs: 0
       Additional RRs: 0
       Queries: www.opensource-osijek.org.lab.os: type A, class IN
```

Vidimo da se na OSI sloju dva **(2)** s naše MAC adrese `2c:d4:44:b3:5f:b6` paket šalje na MAC dresu našeg usmjerivača `d4:ca:6d:34:83:23`.
Na OSI sloju tri **(3)** vidimo da je naša IP adresa s koje šaljemo paket: `192.168.1.20` te da ga šaljemo prema IP adresi usmjerivača: `192.168.1.1`.
Dok na OSI sloju četiri **(4)** odnosno na UDP transportnom sloju vidimo da se poruka šalje na odredišni UDP port: `53` koji predstavlja DNS servis. I sada slijedi OSI sloj koji predstavlja DNS protokol u kojem to i vidimo (`Domain Name System`) te vidimo da se radi o upitu (`query`) što se dekodira iz zastavice: `Flags: 0x0100` kao: `Standard query`.
Slijedi broj upita koji se šalje (`Questions: 1`) i konačno vidimo upit:
`www.opensource-osijek.org.lab.os` te da se radi o **A** vrsti zapisa (`type A`) koji se traži.

Nakon tog upita očekujemo odgovor od našeg DNS *poslužitelja* (usmjerivača konkretno) u kojem ćemo dobiti pripadajuću IP adresu od računala: `www.opensource-osijek.org`.

## Sada pogledajmo i DNS odgovor na razini mrežnog paketa

DNS odgovor koji smo dobili je sljedeći:

```
(1)Frame 95: 165 bytes on wire (1320 bits), 165 bytes captured (1320 bits)
(2)Ethernet II, Src: d4:ca:6d:34:83:23, Dst: 2c:d4:44:b3:5f:b6
(3)Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.20
(4)User Datagram Protocol, Src Port: 53, Dst Port: 62928
     Domain Name System (response)
         Transaction ID: 0x0004
         Flags: 0x8180 Standard query response, No error
         Questions: 1
         Answer RRs: 2
         Authority RRs: 1
         Additional RRs: 0
            Queries
              www.opensource-osijek.org: type A, class IN
            Answers
              www.opensource-osijek.org: type A, class IN, addr 134.209.226.211
              www.opensource-osijek.org: type A, class IN, addr 157.230.103.136
           [Request In: 92]
      [Time: 0.039332000 seconds]
```

U odgovoru više nećemo gledati na OSI slojeve: **1**, **2**, **3** i **4**.
Ovdje iznad OSI sloja četiri (**4**) u dijelu koji je zadužen za DNS protokol vidimo sljedeća važna polja:
I ovdje vidimo koji je bio naš (klijentov) DNS upit, pod `Queries`: `www.opensource-osijek.org`.
Te konačno dobivamo sljedeći odgovor (`Answers`):

```
www.opensource-osijek.org: type A, class IN, addr 134.209.226.211
www.opensource-osijek.org: type A, class IN, addr 157.230.103.136
```

To znači kako su za stranicu (računalo) odnosno domenu: `www.opensource-osijek.org`, odgovorne dvije IP adrese: `134.209.226.211` i `157.230.103.136` (*u trenutku pisanja teksta*).
Dvije IP adrese su zbog zalihosti (redundancije) odnosno jer je tako upisano u **DNS A** zapisima za ovu domenu.



Podsjetite se **DNS A** zapisa u poglavlju:
**25.8.5.5.2. A zapis.**

**Izvori informacija:** `man nslookup`, `man dig`, **RFC 1034**, **RFC 1035**, **RFC 1995**, **RFC 1912**, **RFC 1996.**

## 25.8.6. Telnet i SSH protokoli

***Telnet*** kao i noviji ***SSH*** protokol omogućavaju nam pristup u naredbeni redak (*shell*) odnosno ljusku udaljenog računala (obično poslužitelja). Većina operativnih sustava ima svoju implementaciju *telnet* protokola za udaljeni pristup, ali zbog njegovih poznatih sigurnosnih problema u današnje vrijeme, većinom se prešlo na upotrebu ***SSH*** protokola.
Prvo ćemo se ukratko upoznati s *telnet* protokolom.

### 25.8.6.1. Telnet

Kao što smo rekli ***telnet*** omogućava pristup naredbenom retku (ljusci) udaljenih računala odnosno pristup na virtualni terminal udaljenog *telnet* poslužiteljskog mrežnog servisa.

Telnet također radi prema principu: klijent ←→ poslužitelj.
**Telnet kao transportni protokol koristi TCP na portu 23.**

Telnet se kao protokol aktivno razvijao od 1969 godine. U to vrijeme najviše se koristio u akademskoj zajednici, ali najveću primjenu doživio je početkom 1990. godina sve većim korištenjem interneta. S obzirom na to da su se ljudi sve više spajali na udaljena računala ili poslužitelje pomoću *telneta*, sve više je bilo i pokušaja zlouporabe. Stoga se sve više pažnje počelo posvećivati sigurnosti. Zbog sigurnosnih problema i to najviše činjenice što se sâv promet preko *telnet* protokola prima i šalje u čistom tekstualnom ne kriptiranom formatu, uključujući i ime korisničkog računa i pripadajuće mu lozinke.

*Telnet* se stoga počeo zamjenjivati s puno sigurnijim *SSH* protokolom.
Pošto *telnet* radi prema principu: klijent ←→ poslužitelj, na strani poslužitelja potrebno je pokrenuti *telnet* poslužitelj, a na strani klijenta, neki od *telnet* klijenata, poput programa PuTTY (za *Windows*) ili drugih sličnih programa.

---

### 25.8.6.2. SSH

*SSH* se također koristi za udaljeni pristup naredbenom retku (ljusci). ***Secure Shell*** odnosno *SSH* je mrežni protokol, koji koristi kriptiranje (šifriranje) podataka koji se primaju i šalju, za razliku od *telnet* protokola koji nije siguran za komunikaciju pogotovo preko interneta, jer sve podatke šalje u izvornom obliku odnosno nezaštićeno to jest ne kriptirano (ne šifrirano).

*SSH* također radi prema principu: klijent ←→ poslužitelj.
**SSH kao transportni protokol koristi TCP, na portu 22.**

Postoje inačice jedan i dva, a poznate su kao: *SSH-1* i *SSH-2* od kojih se inačica dva najviše koristi. SSH je zamišljen kao zamjena za *telnet*. SSH koristi (pouzdane) kriptografske algoritme za kriptiranje podataka koji prolaze između klijenta i poslužitelja. SSH koristi takozvani sustav *javnog ključa* (Engl. *Public Key*) koji se često naziva i *asimetričnom kriptografijom*.

Princip rada kriptiranja (šifriranja) sustavom: javni + privatni ključ, je takav da:
**1.** Obje strane u komunikaciji kreiraju svoj pâr: javni te pripadajući privatni ključ. Privatni ključ se čuva i ne daje nikome.
**2.** Jedna strana kriptira podatke s javnim ključem od druge strane, koji mu druga strana daje prije početka komunikacije.
**3.** Druga strana može dekriptirati sadržaj sâmo sa svojim privatnim ključem (koji je pâr od javnog ključa iz koraka **2.**).

Kod SSH protokola osnovni način rada je za svaku stranu u komunikaciji: kreiranje pâra javni + privatni ključ.
Tada se ti ključevi koriste za kriptiranje veze uz dodatnu potrebu da se korisnik identificira s kombinacijom: korisničko ime i pripadajuća lozinka. Pri tome su korisničko ime i lozinka od sustava (od Linuxa), a ne posebno kreirani od strane SSH servisa.

Ovo je najčešća metoda rada. Važno je razumjeti kako se pâr: javni + privatni SSH ključ kreiraju odnosno koriste za svakog korisnika to jest za svaki korisnički račun zasebno, i to u pravilu u njegovom kućnom direktoriju, i to u posebnom `.ssh` poddirektoriju unutar njega. Primjerice za korisnika korisničkog imena `ivan` bi tada to bio vršni direktorij (mapa) imena: `/home/ivan/.ssh/`. Unutar njega se trebaju nalaziti svi ključevi i druge datoteke vezane za SSH protokol.
Tako je standardno ime datoteke koja sadrži privatni ***RSA*** ključ: `id_rsa`, dok je one koja sadrži javni ključ: `id_rsa.pub`.

Druga metoda rada je također upotrebom pâra ključeva: javni + privatni na obje strane (klijent i poslužitelj), ali u svrhu autentikacije. Javni ključ (engl. *Public key*) se tada prvo mora kopirati s klijenta na poslužitelj, a poslužiteljev javni ključ na klijentsku stranu. **Privatni ključevi se nikada ne razmjenjuju!**

Autentikacija se tada odrađuje primjenom javnih ključeva, iako sve i dalje radi na isti način odnosno; kriptiranje javnim ključem, a dekriptiranje privatnim ključem. Privatni ključevi se nikada niti na koji način ne kopiraju preko mreže u procesu autentikacije niti u bilo kojem drugom procesu rada. Ovakva metoda autentikacije omogućava spajanje na drugi sustav bez potrebe za unošenjem kombinacije: korisničko ime i lozinka, već se veza automatski ostvaruje u pozadini.

Pri ovoj metodi autentikacije, javni ključevi susjeda (u komunikaciji) se spremaju u datoteku imena: `authorized_keys`, pa tako svaka strana ima pohranjene javne ključeve druge strane.
Tijekom prvog otvaranja veze, lokalno se pohranjuje javni ključ koji se veže za *IP* adresu ili ime računala *(hostname)* udaljene strane, koji služi za provjeru identiteta druge strane. Međutim u slučaju kada se promijeni javni ključ koji se veže za *IP/ime računala,* to znači da je došlo do nekih potencijalno sigurnosno problematičnih stvari te se ta konekcija neće ostvariti.



Za primjere ovakve konfiguracije/autentikacije pogledajte poglavlje:
**26.2.1.3. Konfiguracija SSH klijenta**.

*SSH* inačica dva (SSH v.2.) se danas najviše koristi jer je donijela:

- Poboljšanja u procesu kriptiranja i razmjene ključeva (engl. *Diffie–Hellman key exchange*).
- Nove algoritme kriptiranja (AES-128, AES-192, AES-256, …).
- Poboljšanu provjeru integriteta podataka.
- Podršku za PKI certifikate.

### 25.8.6.2.1. SSH komunikacija detaljnije (kriptiranje/dekriptiranje)

Vratimo se nazad na uspostavljanje SSH veze, koju ćemo analizirati detaljnije, kako je vidljivo na slici **243.1**.
U inicijalnom trenutku prvo se čita globalna konfiguracija *SSH* klijenta, definirana u datoteci: `/etc/ssh/ssh_config`.

Potom se SSH klijent spaja na SSH poslužitelj i ostvaruje se inicijalna TCP konekcija. Ako se za ključeve koristi ***RSA*** standard (što je pretpostavljena postavka), prvo se učitava ***RSA*** privatni ključ (nâs kao SSH klijenta). Naš ***RSA*** privatni ključ (dio pâra ***RSA*** ključeva: privatni + javni) se pohranjuje u korisničkom kućnom/početnom direktoriju, tj. u njegovom pôddirektoriju ***.ssh***, što bi za primjerice korisnika **root** bila datoteka: `/root/.ssh/id_rsa`.

U slučaju kada inicijalno nismo kreirali pâr ***RSA*** privatni + javni ključ, već neki od ***DSA*** varijanti, oni se provjeravaju i to redom: `id_dsa`, `id_ecdsa`, `id_ed25519`.

Nakon što se uspostavila TCP veza (***SYN***, ***SYN ACK*** i ***ACK***), SSH poslužitelj šalje SSH klijentu (**slika 243.1: korak 1**) inačicu SSH servisa koju ima (pr. *OpenSSH_7.4p1, OpenSSL 1.0.2k-fips* ), na što SSH klijent odgovara sa svojom inačicom SSH klijenta. Ovdje se provjerava ima li možda druga strana neku staru inačicu SSH servisa ili klijenta, na što može reagirati odbijanjem SSH veze.

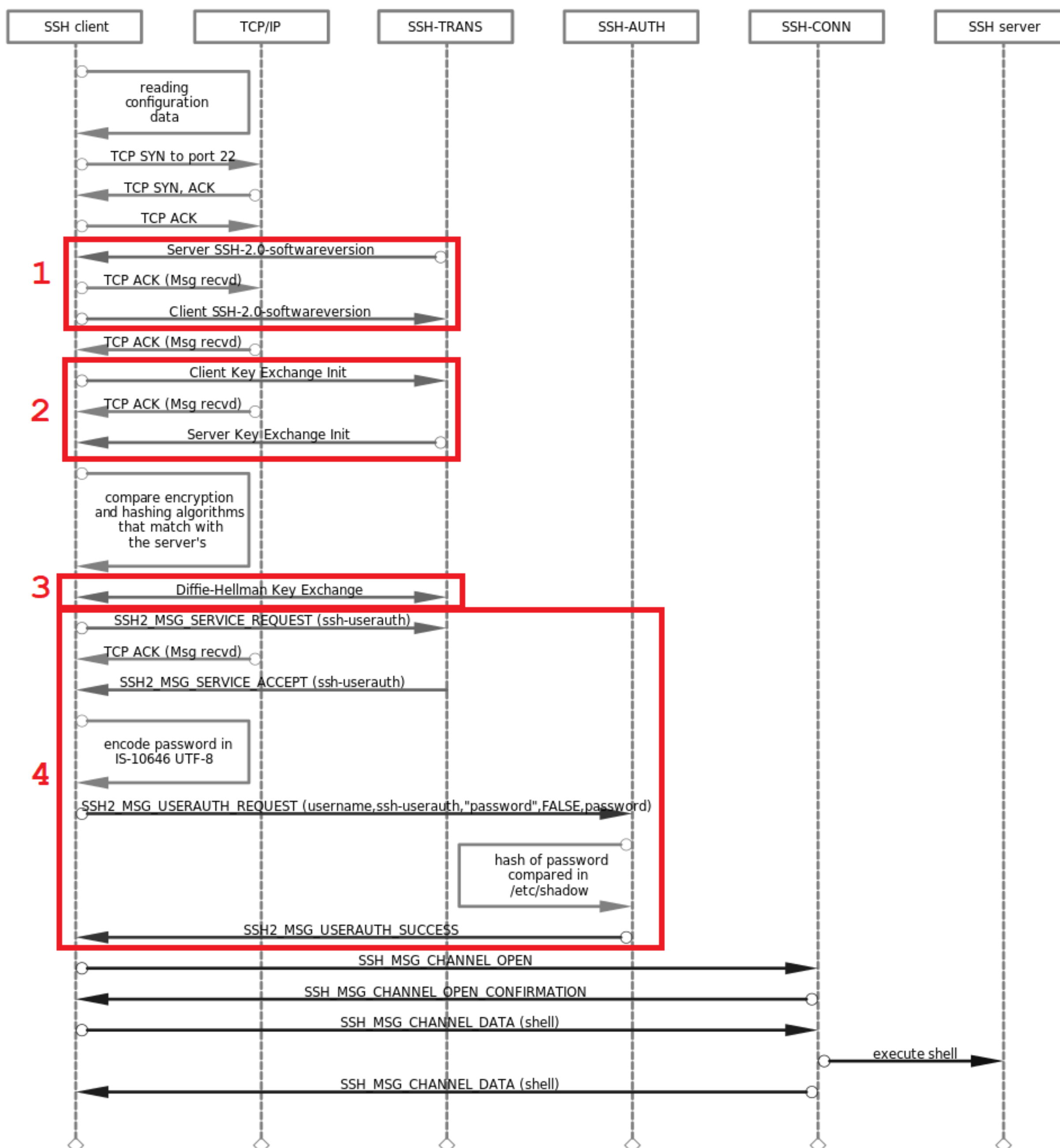
Potom se kao korisničko ime (login) priprema korisničko ime s kojim se želimo spojiti na udaljeni SSH poslužitelj (pr. ***root***). Tada se na obje strane provjerava, nalazi li se unos s javnim ključevima suprotne strane u komunikaciji, u datoteci `known_hosts`, što bi za korisnika **root**, bila datoteka: `/root/.ssh/known_hosts`.

Dakle klijent traži unos od poslužitelja, a poslužitelj od klijenta. Ovim se provjerava je li možda došlo do nekih sigurnosnih proboja u kojima je netko podmetnuo neki drugi ključ. Isto se može dogoditi, ako smo namjerno napravili nove pârove RSA ključeva, pa će tada biti potrebno na drugoj strani obrisati unos one strane koja se promijenila, a pri prvoj novoj konekciji će se upisati novi javni ključ susjeda.
Ključevi suprotne strane se pohranjuju, za svakog klijenta u novom retku ove datoteke. U svakom slučaju, ako postoji razlika između javnog ključa druge strane i onoga zapisanog ovdje, veza će odmah biti prekinuta, kao nesigurna.



Za više detalja o algoritmima i mehanizmima u kriptografiji, pogledajte poglavlje:
**26.4.4. Secure Socket Layer, Transport Layer Security i kriptografija.**

**Slika 243.1. SSH protokol i komunikacija detaljnije.**




**Autor sheme:** Aleksey Valov.

Tek sada se kreće s drugim korakom, vidljivim na slici 243.1, kao korak (**2**).
Ovdje SSH klijent šalje SSH poslužitelju poruku `Key Exchange Init` unutar koje oglašava svoje mogućnosti, poput:

- Koje algoritme za razmjenu ključeva (**kex_algorithms**) podržava, poput: *curve25519-sha256,curve25519-sha256, ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group14-sha256,diffie-hellman-group16-sha512, …* Ove algoritme možemo definirati u opciji `KexAlgorithms`.
- Koje algoritme za sigurno potpisivanje tijekom autentikacije (za provjeru autentičnosti poslužitelja i provjera autentičnosti temeljena na *hostu*), očekujemo od poslužitelja. Ovdje se radi o definiranju algoritama za potpisivanje (***server_host_key_algorithms***) ključeva *hosta*, koje će poslužitelj predložiti i prihvatiti za autentifikaciju *hosta* (računala), poput*: ecdsa-sha2-nistp256-cert-v01, ecdsa-sha2-nistp384-cert-v01, ecdsa-sha2-nistp521-cert-v01, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, ecdsa-sha2-nistp521, ssh-ed25.* Ove algoritme možemo definirati u opciji `HostKeyAlgorithms`, unutar konfiguracije našeg SSH klijenta.
- Slijede algoritmi za simetrično kriptiranje (šifriranje) sesije, koje mi kao klijent nudimo poslužitelju (***encryption_algorithms*****_client_to_server** *[ciphers]*), poput: *aes256-gcm,chacha20-poly1305,aes256-ctr,aes256-cbc,aes128-gcm, aes128-ctr, aes128-cbc, …* Ove algoritme možemo definirati u opciji `Cipher`, unutar konfiguracije našeg SSH klijenta. Zatim slijede algoritmi za simetrično šifriranje sesije koje očekujemo od SSH poslužitelja (***encryption_algorithms*****_server_to_client**), poput: *aes256-gcm, chacha20-poly1305, aes256-ctr, aes256-cbc, aes128-gcm, aes128-ctr, aes128-cbc, …* ***Pri tome primjerice AES i ChaCha označavaju standard za kriptiranje, 256 veličina ključa, a: CBC, GCM, CTR, (Poly1305) i drugi označavaju na koji način i kako će se raditi kriptiranje (šifriranje).***

- Potom šaljemo algoritme koji se koriste za provjeru integriteta podataka tzv. ***MAC*** (*Message Authentication Code*), a koje mi podržavamo (***mac_algorithms_client_to_server***), poput: *hmac-sha2-256-etm,hmac-sha1-etm,umac-128-etm,hmac-sha2-512-etm,hmac-sha2-256,hmac-sha1,umac-128,hmac-sha2-512, …* Ove algoritme možemo definirati u opciji `MACs`, unutar konfiguracije našeg SSH klijenta. I ovdje potom slijede algoritmi koji se koriste za provjeru integriteta podataka, a koje očekujemo od SSH poslužitelja *(**mac_algorithms_server_to_client**)*.
- I na kraju slijede algoritmi za komprimiranje podataka, koje mi podržavamo i/ili očekujemo od SSH poslužitelja, primjerice: *none, zlib, …*.

Nakon svega ovoga što smo poslali na SSH poslužitelj, on nam odgovora, s vrlo sličnim odgovorom **`Key Exchange Init`**, sa svim algoritmima koje on podržava. Ovdje se dogovaraju kriptografski algoritmi koji će se koristiti za obavljanje razmjene ključeva, te za šifriranje (kriptiranje) podataka koji će prolaziti kroz SSH komunikacijski kanal.
U ovom koraku, vidljivom na slici 243.1., kao (**3**) dolazi do odabira algoritma za razmjenu ključeva, i to od onih oglašenih i podržanih, definiranih pod: **kex_algorithms** (***KEX algorithms***). Tako se primjerice odabire algoritam: ***curve25519-sha256***.

Budući da SSH obično koristi tzv. ***Elliptic Curve Diffie-Hellman*** (***ECDH***) metodu, razmjena ključeva započinje tako što klijent generira *efemerni* (privremeni) par ključeva (privatni i pridruženi mu javni ključ) te šalje svoj javni ključ u poruci ***SSH_MSG_KEX_ECDH_INIT***. *ECDH* mehanizam omogućuje dvjema stranama, od kojih svaka ima par javno-privatnih ***EC*** (*Elliptic-curve*) ključeva, da uspostave zajedničku tajnu/ključ (*Shared secret*) preko nesigurnog kanala. Ovaj zajednički ključ odnosno zajednička tajna, može se izravno koristiti kao ključ ili za izvođenje drugog ključa.

Ovaj ključ ili izvedeni ključ tada se koriste za šifriranje naknadne komunikacije pomoću šifre simetričnog ključa.
Važno je naglasiti da je ovaj pâr ključeva *efemeran,* što znači da će se upotrijebiti sâmo tijekom razmjene ključeva, a kasnije će se odbaciti. To čini vrstu napada u kojoj napadač pasivno snima šifrirani promet s nadom da će nekada u budućnosti ukrasti privatni ključ, nemogućom. Ovo svojstvo naziva se tajnost prema naprijed (*forward secrecy*). Kada SSH poslužitelj zaprimi ***SSH_MSG_KEX_ECDH_INIT*** poruku, i on generira *efemerni* (privremeni) par ključeva (privatni i pridruženi javni ključ), on šalje javni ključ poslužitelja u poruci ***SSH_MSG_KEX_ECDH_INIT*** prema SSH klijentu. Sada se odabiru konačni algoritmi koji će se kasnije koristiti za kriptiranje podataka u prijenosu.
Dalje se kompleksnim kriptografskim metodama, upotrebom gore navedenih algoritama kriptira komunikacijski kanal, od kojih je jedan od njih naš i poslužiteljev ***RSA*** javni ključ, pomoću kojih konačno i dolazi do razmjene podataka.

U konačnici u cijelom procesu obje strane trebaju generirati šest ključeva:
dva ključa za šifriranje, dva inicijalizacijska vektora (***IV***) i dva za integritet podataka koji se šalju:

- Ključevi za šifriranje koriste se za osiguravanje povjerljivosti podataka i koriste se sa simetričnim sustavom za šifriranje (***encryption_algorithms*** *[ciphers]*) i dešifriranje podataka.
- Ključevi za integritet podataka obično se koriste s kôdom za provjeru autentičnosti poruke (***MAC***) (***mac_algorithms***) kako bi se osiguralo da eventualni napadač ne manipulira podacima u prijenosu.
- Vektori inicijalizacije (***IV***) obično su slučajni brojevi koji se koriste kao ulaz za simetričnu šifru. Svrha ***IV*** je osigurati da ista poruka šifrirana dva puta, ne rezultira istim šifriranim tekstom.

Dodatni sigurnosni parametar ugrađen u SSH protokol v.2 je značajka koja omogućuje bilo kojoj strani da forsira još jedno pokretanje faze razmjene ključeva, mijenjajući ključeve šifriranja i integriteta za sesiju. Ideja je to raditi povremeno, svako nekoliko sekundi ili nakon što je određeni broj bajtova podataka prošao preko veze. Ovaj parametar se zove engl. ***Rekey***.
Vezano za naš ***RSA*** javni ključ (dio pâra ***RSA*** ključeva: privatni + javni), on se pohranjuje u korisničkom kućnom/početnom direktoriju (mapi), tj. u njegovom pôddirektoriju ***.ssh***, što bi za primjerice korisnika **root** bila datoteka: `/root/.ssh/id_rsa.pub`.

To je ključ koji se šalje drugoj strani i koji druga strana koristi za šifriranje podataka, koji se mogu dešifrirati sâmo s pripadajućim privatnim ključem. Dakle sâmo vlasnik pâra od javnog ključa, odnosno njegovog privatnog ključa može dešifrirati sadržaj šifriran s javnim ključem. U drugom smjeru komunikacije, mi kao klijent koristimo javni ključ poslužitelja za šifriranje, a on svoj privatni ključ za dešifriranje.

Tek kroz taj šifrirani kanal se šalju korisničko ime i lozinka za spajanje na SSH poslužitelj: slika 243.1., korak (**4**).



Za konfiguracijske primjere, pogledajte poglavlje:
**26.2.1.4. Napredna konfiguracija SSH servisa.**

**Izvori informacija: (976),(977), `man 5 ssh_config`, `man 5 sshd_config`, RFC 4251, RFC 4252, RFC 4253, RFC 4254, RFC5656.**

## 25.8.7. NFS

***NFS*** (*Network File System*) je mrežni datotečni sustav koji je 1984. godine razvila tvrtka ***Sun Microsystems.*** On omogućava korisniku na klijentskom računalu pristup datotekama preko računalne mreže, kao da su datoteke dostupne na lokalnom računalu. *NFS* kao i mnogi drugi protokoli temelji se na sustavu ***Remote Procedure Call*** (ONC RPC). *NFS* se razvijao i nadograđivao s godinama, pa je tako inačica dva definirana u RFC 1094, a inačica tri u RFC 1813 te trenutna inačica četiri u RFC 3010 te RFC 3530 (2003.g.) kao i RFC 7530 (2015.g) godine.

Ovdje su uvedene mnoge optimizacije i poboljšanja, a zatim su napravljene i dvije nadogradnje:
- NFS v.4.1 definiran je u RFC 5661 (2010).
- NFS v.4.2 definiran je u RFC 7862 (2016).

Zadnja inačica (v.4.2) uvela je i značajne optimizacije poput: funkcije kloniranja i kopiranja na poslužiteljskoj strani, *sparse* datoteke, rezervaciju prostora i mnoge druge.

Standardni scenarij rada *NFS* protokola (v.3.) je sljedeći:
1. Poslužitelj implementira i pokreće *NFS* servisni (*daemon)* proces, koji se prema zadanim postavkama pokreće kao `nfsd` preko kojeg njegovi podaci odnosno datoteke i direktoriji (mape) bivaju dostupni *NFS* klijentima.
2. Administrator *NFS* poslužitelja određuje što uopće učiniti dostupnim (koji dijeljeni direktorij) tako da se izvoze (*eksportiraju*) imena i parametri željenog direktorija (mape), obično pomoću konfiguracijske datoteke `/etc/exports` i naredbe `exportfs`.
3. Administracija sigurnosti *NFS* poslužitelja provjerava može li prepoznati i odobriti klijente koji se preko *NFS* protokola žele spojiti na *NFS* poslužitelj.
4. Konfiguracija mrežne komponente poslužitelja provjerava mogu li klijenti pregovarati s *NFS* poslužiteljem.
5. *NFS* klijent zahtijeva pristup izvezenim (eksportiranim) podacima, obično izdavanjem naredbe za montiranje. Klijent traži poslužitelj (`rpcbind`) koji port koristi *NFS* poslužitelj, klijent se povezuje s *NFS* poslužiteljem (na `nfsd`), a `nfsd` tada prosljeđuje zahtjev prema `mountd` servisu (*daemonu*) klijenta.
6. Ako je sve dobro prošlo, korisnici računala klijenta mogu vidjeti i komunicirati s montiranim mrežnim datotečnim sustavom na *NFS* poslužitelju unutar dopuštenih parametara.

Procedura automatskog povezivanja (*mountanja*) *NFS* poslužitelja može se dodati i u datoteku `/etc/fstab` (o tome kasnije).



**Napomene**:
- *NFS* promet preko mreže nije kriptiran, ali ga je moguće *tunelirati* preko protokola za enkripciju.
- Za razliku od ***Sambe*** (Windows *SMB/CIFS share*) *NFS* nema standardno postavljenu autentifikaciju korisnika. Pristup klijentu inicijalno je ograničen IP adresom ili nazivom računala klijenta, što je moguće naknadno posebno definirati.
- *NFS* očekuje da su identifikacijski brojevi korisnika (***UID***) i/ili korisničkih grupa (***GID***) jednaki i na klijentu i na poslužitelju. Međutim moguće je to sve i promijeniti.
- Kod *NFSa*, *NFS* klijent i *NFS* poslužitelj mogu imati potpuno različite operativne sustave.
- *NFS* radi poprilično brže od ***Windows*** SMB/CIFS protokola, pošto je specijaliziran sâmo i isključivo za razmjenu datoteka i njihovih atributa, a ne i drugih sadržaja i opcija.

**Koje sve NFS servise imamo na sustavu?**
Sa strane *NFS* poslužitelja u *NFS* serverskom softverskom paketu standardno dobivamo nekoliko servisa odnosno *daemona*:
- `portmap` - ovo je servis koji povezuje (*mapira*) *NFS* pozive s *NFS* klijenata na RPC servis. U NFS v.4. on više nije potreban odnosno više se ne koristi.
- `nfs` - ovo je servis koji prevodi udaljeni zahtjev za pristup dijeljenom direktoriju (i datotekama) u upit prema lokalnom diskovnom sustavu *NFS* poslužitelja.
- `rpc.mountd` - ovo je servis koji je u konačnici zadužen za *montiranje* i *demontiranje* datotečnog sustava.

**Koje su sve važne datoteke za NFS?**
U upotrebi *NFS* sustava imamo nekoliko konfiguracijskih datoteka koje bi trebali upoznati, a to su:
- `/etc/exports` - ovo je datoteka na *NFS* poslužitelju u kojoj su definirani direktoriji (mape) koje će se eksportirati preko *NFS* sustava. Dakle ovdje se definiraju lokalni direktoriji na *NFS* poslužitelju kojima će **NFS** klijenti moći pristupati preko *NFS* mrežnog sustava.
- `/etc/fstab` - ovo je datoteka na *NFS* klijentu u kojoj se može definirati udaljeni *NFS* direktorij koji će se montirati automatski tijekom svakog restarta (*NFS*) klijentovog računala.
- `/etc/sysconfig/nfs` - ovo je datoteka na strani *NFS* poslužitelja u kojoj se definiraju osnovni parametri rada *NFS* servisa pri njegovoj inicijalizaciji s posebnim opcijama i parametrima koje ne možemo definirati drugdje na strani *NFS* poslužitelja.

**Izvori informacija: `man 5 exports`,`man 5 fstab`,`man 5 nfs`,`man 5 nfs.conf`, RFC 1094, RFC 1813, RFC 3010, RFC 3530, RFC 5661, RFC 7530, RFC 7862.**

### 25.8.7.1. Instalacija i konfiguracija

Za instalaciju *NFS* klijenta i *NFS* poslužitelja potrebno je instalirati slijedeće *NFS* softverske pakete i na *NFS* klijent i *NFS* poslužitelj:

```
yum install -y nfs-utils nfs-utils-lib
```

← Ovdje će se vjerojatno automatski instalirati i nekoliko dodatnih softverskih paketa u pozadini.

***Dalje ćemo se bazirati na NFS sustavu inačice 4.x.***

**Na NFS poslužitelju je potrebno napraviti sljedeće:**
Recimo da želimo kreirati novi direktorij `/NFS-share` koji ćemo dijeliti preko mreže i *NFSa*. Kreirajmo ovaj direktorij:

```
mkdir /NFS-share
```

Sada isti direktorij moramo definirati da će se eksportirati preko *NFSa* dodavanjem sljedećeg retka u datoteku: `/etc/exports`. Ovdje ćemo još dodati i neke opcije koje ćemo potom objasniti. Dakle trebamo dodati i sljedeći redak u gore navedenu datoteku kako bi eksportiranje gore kreiranog direktorija bilo omogućeno:

```
/NFS-share *(rw,sync,no_root_squash)
```

Pogledajmo što smo sve ovdje definirali:

- `/NFS-share` - ovo je direktorij (mapa) koji dijelimo preko mreže, pomoću *NFS* sustava.
- `*` - ovo je oznaka mreže ili IP adrese klijenata kojima dopuštamo pristup preko *NFS* sustava. Ovdje `*` znači da se pristup dopušta sa svih mreža i svih IP adresa. Primjerice mogli smo umjesto `*` definirati samo IP adresu našeg *NFS* klijenta, poput: `192.168.1.100`.
- `rw` - ovo označava kako dajemo prava pristupa: *Read* (čitanje) i *Write* (pisanje) ovom direktoriju. Mogli smo primjerice staviti `ro` što bi značilo samo *Read only* (sâmo čitanje) bez prava mijenjanja/zapisivanja.
- `sync` - ovo znači kako definiramo sinkroni način zapisivanja, u kojemu će se bilo koja operacija zapisivanja u ovaj *NFS* mrežni direktorij odraditi sâmo i isključivo tako da tek kada se podaci stvarno snime na *NFS* poslužitelj, odnosno na njegov datotečni sustav, operacija zapisivanja će biti završena. Odnosno sve dok se od datotečnog sustava poslužitelja ne dobije potvrda da je određena datoteka stvarno snimljena na površinu diska, operacija zapisivanja preko *NFS* sustava nije završila uspješno. Ovo je najsigurnija metoda, ali i najsporija jer se međumemorija za zapisivanje na razini datotečnog sustava neće koristiti.
- `no_root_squash` - ovo je naredba koja dozvoljava i *`root`* korisniku da može preko *NFS* mrežnog dijeljenog diska imati pravo pristupa.
- `no_subtree_check` - ovo je naredba koja *NFS* poslužitelju nalaže da ne mora prolaziti kroz cijelo stablo direktorija i pod direktorija počevši od vršnog u *NFS* strukturi, gledajući na datoteke, njihova imena i ovlasti, u trenutku kada klijent mijenja ime neke datoteke ili kada ju je druga strana promijenila. *NFS* na kojem se imena datoteka mogu često mijenjati, bolje je da imaju postavljenu opciju: `no_subtree_check`. Suprotnost je opcija `subtree_check` kod koje se nalaže suprotna aktivnost koja može usporiti sustav.
- Za NFS inačice 4, imamo i sljedeću opciju **na strani NFS poslužitelja**:
  - `fsid=X` ovdje se postavljaju *file deskriptori* i njegovi atributi kojima se za točno određeni dijeljeni NFS direktorij postavlja točno određeni broj odnosno oznaka uređaja (MAJOR:MINOR brojevi) da budu točno definirani (kao broj `X`), a umjesto broja koji proizlazi iz većeg (*Major*) i manjeg (*Minor*) broja blok uređaja na montiranom datotečnom sustavu. Vrijednost `0` ima posebno značenje kada se koristi s NFSv4. NFSv4 ima koncept korijenskog direktorija ukupnog izvezenog datotečnog sustava preko *NFS*-a. Izvozna točka koja se izvozi s `fsid=0` koristi se kao identifikator za ovaj korijenski direktorij.
    To bi u datoteci: `/etc/exports` izgledalo ovako:

```
/eksport/  *(rw,sync,fsid=0)   #→ ovo bi bio navedeni vršni NFS direktorij (/eksport)
```

A tada bi drugi pod direktorij za ***NFS4*** bio primjerice:

```
/eksport/NFS-share *(rw,sync,no_root_squash) #→ ovo bi bio prvi pod direktorij ispod vršnog NFS dijeljenog direktorija
```

Međutim trenutno ovu opciju nećemo koristiti, pa idemo dalje.

**Zamislim da je IP adresa NFS poslužitelja: 192.168.1.1**
Za ***RedHat/CentOS*** 6.x potrebno je pokrenuti NFS servise na sljedeći način:

```
service rpcbind start
service nfs start
```

Odnosno za trajno pokretanje (tijekom pokretanja sustava):

```
chkconfig nfs on
chkconfig nfslock on
chkconfig rpcbind on
```

Međutim za ***RedHat/CentOS*** 7.x/8.x potrebno je pokrenuti ***NFS*** servise na sljedeći način:

```
systemctl start nfs-server.service
```

Odnosno za trajno pokretanje (za ***RedHat/CentOS*** 7.x/8.x/9.x), tijekom svakog pokretanja sustava, potrebno je napraviti:

```
systemctl enable nfs-server
```

**Na NFS klijentu** (IP: 192.168.1.100) **je potrebno napraviti sljedeće:**
Recimo da se želimo spojiti na *NFS* poslužitelj na IP adresi: 192.168.1.1 i prvo želimo vidjeti što nam on se sve nudi preko *NFS* sustava. Stoga pokrenimo naredbu `showmount` na sljedeći način:

```
showmount -e 192.168.1.1

Export list for 192.168.1.1:
/NFS-share *
```

Vidimo kako nam se preko NFS sustava na *NFS* poslužitelju nudi mrežno dijeljeni direktorij `/NFS-share`

Sada na klijentu moramo napraviti direktorij u koji ćemo *montirati NFS* sa *NFS* poslužitelja. Stoga kreirajmo direktorij `/NFS`

```
mkdir /NFS
```

Potom možemo napraviti montiranje *NFS* točke: *NFS* mrežni direktorij `/NFS-share` u naš lokalni direktorij `/NFS`

```
mount -t nfs 192.168.1.1:/NFS-share /NFS
```

I nakon toga ćemo na našem NFS klijentu imati *montiran* direktorij `/NFS` koji je zapravo *NFS* dijeljeni mrežni direktorij s *NFS* poslužitelja. Pogledajmo što smo sada dobili (s naredbom `mount`):

```
mount | grep nfs

192.168.1.1:/NFS-share on /NFS type nfs4
rw,relatime,vers=4,rsize=131072,wsize=131072,namlen=255,hard,proto=tcp,timeo=600,retr
ans=2,sec=sys,clientaddr=192.168.1.100,minorversion=0,local_lock=none,
addr=192.168.1.1
```

Dakle vidimo kako je naš *NFS* uredno *montiran* kao *NFS* v.4 i to s dodatnim parametrima (standardno)

## Datoteka: `/etc/fstab`

U slučaju kada želimo da nam se automatski montira udaljeni *NFS* mrežni dijeljeni direktorij to najjednostavnije možemo napraviti dodavanjem unosa u datoteku `/etc/fstab`. Mi ćemo u našem slučaju dodati sljedeći redak:

```
192.168.1.1:/NFS-share     /NFS    nfs    defaults    0 0
```

Pogledajmo i nekoliko opcija koje možemo koristiti ovdje, a koje bi mijenjale ključnu riječ `defaults`.
Ove opcije se mogu navoditi u nîzu, odvojene znakom zareza (`,`). Pogledajmo neke od njih:

- Prava zapisivanja ili čitanja na NFS poslužitelj i druge ovlasti:
  - `ro` - označava sâmo prava čitanja, ali ne i zapisivanja.
  - `rw` - označava i pravo zapisivanja i čitanja.
  - `noexec` -definira nemogućnost da korisnik na strani *NFS* klijenta ima pravo izvršavanja bilo koje datoteke unutar ovog *NFS* dijeljenog mrežnog direktorija. Ove opcije se mogu definirati i na strani *NFS* poslužitelja.
  - `nosuid` - definira nemogućnost da korisnik na strani *NFS* klijenta ima pravo postavljanja dodatnih posebnih ovlasti poput: *set-user-identifier* i *set-group-identifier*. Ove opcije se mogu definirati i na strani *NFS* poslužitelja.
- Stanje rada u slučaju privremene nedostupnosti *NFS* poslužitelja:
  - `hard` - označava da korisnik na strani *NFS* klijenta NE MOŽE prekinuti proces ponovnog pokušavanja spajanja na *NFS* poslužitelj koji je privremeno nedostupan, sve dok opcija (`intr`) nije isto aktivna. Opcija `intr` znači kako u slučaju kada je *NFS* poslužitelj nedostupan *NFS* zahtjevi se mogu prekinuti, ako je *NFS* poslužitelj trajno nedostupan.
  - `soft` - označava kako će se u slučaju kada je *NFS* poslužitelj nedostupan pa ne potvrđuje prijem *NFS* paketa, koristiti mehanizmi pokušaja ponovnog slanja točno određeni broj puta kao i čekanja na to slanje, definiranim u opcijama `timeo` i `retrans`. Ove dvije opcije se prema tome mogu koristiti sâmo, ako imamo postavljenu opciju `soft`.
- Metoda zapisivanja na disk (na strani *NFS* poslužitelja):
  - `sync` - je standardna metoda spajanja odnosno korištenja *NFS*-a koju smo objasnili. Ovo je najsigurnija metoda zapisivanja podataka na disk, na *NFS* poslužitelju bez upotrebe međumemorije, ali i najsporija.
  - `async` - ovo je metoda u kojoj se tijekom zapisivanja od *NFS* poslužitelja ne traži potvrda da su podaci stvarno zapisani na površinu diska, kao kod metode `sync` već, čim je *NFS* poslužitelj dobio sve što mu treba za zapisivanje, vraća *NTP* klijentu da je procedura zapisivanja prošla uredno. Ovo potencijalno, u slučaju naglog nestanka el. energije znači gubitak podataka koji su ostali u *RAM* memoriji (međumemoriji za operacije zapisivanja na disk), a nisu se stigli zapisati na površinu diska.
- *Timeout* i *retransmisije*:
  - `timeo=`**`XY`** - ovo je vrijeme u milisekundama (**XY**) koje će *NFS* klijent čekati prije nego dobije potvrdu od *NFS* poslužitelja (ACK), a prije nego li ponovno pošalje mrežni paket koji nije potvrđen (za TCP).
  - `retrans=`**`X`** označava koliko puta (**X**) će se ponoviti paket koji nije potvrđen nakon isteka `timeo` vremena.
- Sigurnosne postavke:
  - `sec=` ovdje može biti postavljeno:
    - `sys` označava standardne postavke u kojima se koriste lokalne linux/unix ***UID*** i ***GID*** identifikatori za korisnike i korisničke grupe.
    - Može se koristiti i (*krb5*, *krb5i*, *krb5p* …), za ***Kerberos*** autentikaciju.

- Postavljanje veličine bloka podataka koja će se koristiti kod primanja ili slanja podataka preko *NFS* protokola, što može uzrokovati dodatno fragmentiranje mrežnih paketa (usporenje) ili povećati veličinu mrežnog paketa te sâmim time smanjiti broj mrežnih paketa na mreži, ako ne dolazi do fragmentacije, što ovisi i o postavkama mreže.
  Ovim promjenama se stoga može ili ubrzati ili usporiti *NFS* komunikacija. Standardno je najveća vrijednost 65.535 (ovisno o TCP postavkama *NFS* klijenta i poslužitelja i mreže između njih), ali može biti i znatno veća, opet ovisno o TCP postavkama. Ovdje možemo definirati sljedeće:
  - `rsize=XY` - ovdje se definira veličina podataka za primanje (u bajtima) koja će se pokušati koristiti za pojedini mrežni paket.
  - `wsize=XY` - ovdje se definira veličina podataka za slanje (u bajtima) koja će se pokušati koristiti za svaki pojedini mrežni paket.
- Postavke za atribute pristupanja datotekama preko *NFS* dijeljenog mrežnog direktorija. Naime u Unix/Linux operativnim sustavima, svaki puta kada pristupamo određenoj datoteci (čak i za sâmo čitanje) zapisuje se vrijeme kada je toj datoteci zadnji puta pristupano, što vrijedi i za pristup određenom direktoriju kao i vremenu kada je datoteka promijenjena. To možemo vidjeti s Linux naredbom `stat`. Slična statistika se vodi i za datoteke i direktorije preko *NFS* sustava preko mreže, što može pogoršati performanse jer se kod svakog novog pristupa nekoj datoteci preko *NFS*-a mora za tu datoteku promijeniti i ovo vrijeme (preko *NFS*-a preko mreže naravno). To često nije potrebno i sâmo usporava pristup datotekama preko mreže odnosno preko *NFS* sustava. Ovdje imamo opcije:
  - `atime` - označava da će se za svako pristupanje svakoj pojedinog datoteci zapisati vrijeme pristupa dok `noatime` - naznačava kako se neće zapisivati vremena pristupa datotekama. Dakle `noatime` znači kako se nikakve statistike vremena UOPĆE NEĆE zapisivati, pa opcije dolje više nemaju značaj.
  - `relatime` - označava kako želimo da se tijekom svakog pristupa svakoj pojedinoj datoteci osvježi vrijeme u koje joj se pristupalo, ako ono odstupa od zadnjeg `atime` vremena. Dok `norelatime` naznačavan da ne želimo da se ovo vrijeme stalno zapisuje (za svaku datoteku).
  - `diratime` - označava kako će se voditi i vremenske statistike ulaska u direktorije dok `nodiratime` označava kako se one neće voditi. Dakle ovdje se radi o razini direktorija, a ne datoteka.
- Inačica *NFS* protokola koji se koristi:
  - `vers=` - XY pri tome XY označava inačicu *NFSa*, koja može biti: `2`,`3`,`4` ili `4.0`, `4.1`, … ili se može koristiti format:
  - `vers=X` - pa potom oznaka pod inačice `minorversion=Y`, što bi za recimo v.4.1 bilo sljedeće: `vers=4 minorversion=1`

Većina drugih opcija je opisana u sljedećem dokumentu:
https://www.centos.org/docs/5/html/Deployment_Guide-en-US/s1-nfs-client-config-options.html


Za više detalja o datoteci `/etc/fstab` pogledajte poglavlje: **13.7.3 Ograničenje zauzeća prostora na disku** (*quota*)

Ako smo mijenjali datoteku `/etc/fstab` tada moramo pozvati sljedeću naredbu kako bi se montirali odnosno aktivirali i novi unosi u ovoj datoteci. To ćemo postići upotrebom naredbe `mount` na sljedeći način:

```
mount -a
```

U slučaju kada želimo maknuti odnosno demontirati konkretan *NFS* (pr. direktorij `/NFS`) to možemo napraviti na sljedeći način; pod pretpostavkom da se **ne** nalazimo unutar direktorija (mape) koji je montiran:

```
umount /NFS
```

Praćenje performansi *NFS* sustava možemo raditi sa sljedećim naredbama:

- `nfsiostat` - (dolazi u softverskom paketu `nfs-utils`), a daje nam razne statistike rad *NFS* sustava.
- `nfsstat` - (dolazi u softverskom paketu `nfs-utils`) i daje nam razne statistike rad *NFS* sustava.
- `nfsiostat-sysstat` - (dolazi u softverskom paketu `sysstat`) daje nam razne statistike rada *NFS* sustava.

Pogledajmo ***NFS*** statistike koje ćemo dobiti sa pokretanjem sljedeće naredbe (razlomili smo ispis na dva reda):

```
nfsiostat-sysstat
Filesystem: rkB_nor/s wkB_nor/s  rkB_dir/s wkB_dir/s rkB_svr/s wkB_svr/s ops/s rops/s  wops/s
192.168.1.254:/mnt/storage/

              0.00      0.01      0.00       0.00      0.00      0.01     0.19   0.00   0.00
```

Slične statistike možemo dobiti i sa sljedećom naredbom:

```
nfsiostat
192.178.1.254:/mnt/storage/ mounted on /home:

   op/s         rpc bklog
   0.19    0.00
read:           ops/s    kB/s   kB/op         retrans    avg     RTT (ms)    avg exe (ms)
                 0.000   0.001 133.149        0 (0.0%)   15.628  15.674
write:          ops/s    kB/s   kB/op         retrans    avg    RTT (ms)    avg exe (ms)
                 0.000   0.085 460.415        0 (0.0%)   36.950 188.693
```

Odnosno s naredbom `nfsstat` možemo vidjeti detalje o *NFS* operacijama, poput pr. *read/write/open/commit/access/*, … .

**Izvori informacija: (451),(452),(453),(454),(455),(456),(457),(458), `man nfsstat`, `man nfsiostat`, `man nfsiostat-sysstat`, `man 5 exports`, `man 5 fstab`, `man 5 nfs`, `man 5 nfs.conf`.**

## 25.8.8. Elektronička pošta (e-mail)

Elektronička pošta odnosno e-pošta (Engl. *e-mail*) je metoda razmjene poruka ("pošte") između ljudi koji koriste elektroničke uređaje. E-pošta je zamišljena kao elektronička (digitalna) inačica ili pandan pošte, u vrijeme kada je klasična pošta značila samo fizičku poštu. E-pošta je kasnije postala sveprisutan komunikacijski medij, do te mjere da se u trenutnoj upotrebi adresa e-pošte često tretira kao osnovni i neophodan dio mnogih procesa u poslovanju, trgovini, obrazovanju, zabavi i drugim sferama svakodnevnog života u cijelom svijetu.

Razvoj e-pošte započeo je 1960-ih, ali u početku su korisnici mogli slati e-poštu samo drugim korisnicima na istom računalu. Povijest modernih internetskih usluga e-pošte seže do ranog ***ARPANET***-a (preteče interneta), sa standardom za kodiranje poruka e-pošte objavljenom još 1973 godine i definiranom u **RFC 561**. Prva poruka e-pošte poslana ranih 1970-ih slična je osnovnoj e-pošti poslanoj danas. *Ray Tomlinson* je zaslužan kao izumitelj elektroničke pošte, on je 1971. godine razvio prvi sustav koji je mogao slati poštu između korisnika na različitim računalima diljem ***ARPANET*** računalne mreže, koristeći znak **@** za povezivanje korisničkog imena s odredišnim poslužiteljem.

Sredinom 1970-ih, ovaj je oblik bio prepoznat kao e-pošta. Međutim sve do 1990-tih elektronička pošta se šire koristila samo u poslovnom svijetu, unutar vlada i vladinih agencija, industriji i vojnoj sferi, ali i dalje nije široko prihvaćena, sve do većeg razvoja i širenja interneta sredinom 1990-tih. Današnji sustavi e-pošte temelje se na modelu spremanja i prosljeđivanja poruka (e-pošte). Poslužitelji e-pošte prihvaćaju, prosljeđuju, isporučuju i pohranjuju poruke: neki ih čuvaju samo do preuzimanja, a neki sve dok ih klijent ne obriše. Pri tome niti korisnici ni njihova računala ne moraju biti istovremeno *online*; već se moraju povezati, obično s poslužiteljem e-pošte ili sučeljem web-pošte, kako bismo slali ili primali
poruke ili ih preuzimali.

Izvorno se uporabom elektroničke pošte moglo komunicirati samo uporabom *ASCII* teksta, ali je e-pošta vrlo brzo proširena takozvanim ***MIME*** (Engl. *Multipurpose Internet Mail Extensions*) standardom za dodavanje privitaka multimedijskog ili drugog sadržaja, poput dokumenata, slika, glazbe, videa i slično, uz mogućnost za prijenos teksta u drugim skupovima znakova osim *ASCII*. ***MIME*** standard uveo je specifikacije skupa znakova i dva načina kodiranja za prijenos sadržaja kako bi se omogućio prijenos podataka koji nisu *ASCII*: takozvani ***Quoted-printable***, uglavnom za 7-bitni sadržaj s nekoliko znakova izvan tog raspona i ***base64*** kodiranja za proizvoljne binarne podatke (priloge i dodatke). Potom su uvedene ***8BITMIME*** (**RFC 1652**) i ***BINARY*** ekstenzije, ali neki sustavi za prijenos e-pošte ih ne podržavaju.

Ovakva komunikacija definirana je u **RFC 5322**. U nekim zemljama softver za e-poštu krši ***RFC 532***2 slanjem sirovog teksta koji nije *ASCII*, uz upotrebu nekoliko shema kodiranja istovremeno; kao rezultat, prema zadanim postavkama, poruke na jeziku koji nije latinična abeceda pojavljuju se u nečitljivom obliku. Jedina iznimka je slučajnost, ako pošiljatelj i primatelj koriste istu shemu kodiranja. Stoga se za međunarodne skupove znakova često koristi ***Unicode*** (*Unicode Transformation Formats):* obično **UTF-8** uz **base64** standard kodiranja za prijenos (binarnih) podataka to jest priloga (datoteka) ili umetnutih sadržaja (za HTML format e-pošte).

Međutim da bi potpuna internacionalizacija bila moguća, uvedeni su: **RFC 6530**, **RFC 6531**, **RFC 6532** i **RFC 6533**.



Za više detalja o ***MIME*** standardu, pogledajte poglavlje**:**
**4.7. Format datoteka (*MIME type*).**

Za više informacija o kodiranju i dekodiranju, pogledajte poglavlje:
**10.3.2.1. Kodiranje i dekodiranje.**

Svaka poruka (elektronička pošta) mora biti oblikovana na način da se sastoji od zaglavlja i tijela poruke.
Zaglavlja elektroničke pošte definirana su u **RFC 4021**. Svaka elektronička pošta mora imati sljedeća zaglavlja:

- **From:** odnosno definiciju pošiljatelja elektroničke pošte.
- **To:** odnosno definiciju primatelja elektroničke pošte.
- **Subject:** odnosno naslov to jest predmet elektroničke pošte.

Moguća su i druga zaglavlja, ali su opcionalna; poput: ***Date:*** (datuma slanja), ***cc:*** (primatelja kopija poruke [*carbon copy*]), ***Bcc:*** (primatelja kopija poruke, koji ne vide kome je sve poslana kopija: ***cc*** i ***bcc*** [*blind carbon copy*]), ali i mnoge druge.

Nakon zaglavlja, svaka elektronička pošta mora imati i neki sadržaj (engl. *content/body* ) odnosno takozvano tijelo poruke.
Većina modernih klijenata e-pošte dopušta korištenje ***običnog teksta*** ili ***HTML-a*** za tijelo poruke, po izboru korisnika.
HTML poruke e-pošte često uključuju automatski generiranu kopiju običnog teksta radi kompatibilnosti.

Prednosti HTML-a uključuju mogućnost uključivanja umetnutih poveznica i slika, izdvajanja prethodnih poruka u blok navodnika, prirodnog prelamanja sadržaja na bilo kojem prikazu, korištenja naglaska kao što su podcrtavanje, zadebljani tekst i kurziv te promjenu stilova fonta. Nedostaci uključuju povećanu veličinu e-pošte, probleme oko privatnosti u vezi sa sigurnosnim propustima, zlouporabu HTML e-pošte kao vektora za takozvane *phishing* napade i širenje zlonamjernog softvera.

Unutar elektroničke pošte, moguće je dodati i prilog (engl. *Attachment*), koji može biti bilo koja datoteka, koju možemo ubaciti u elektroničku poštu. Tipični prilozi uključuju dokumente poput *Microsoft Word*, *Libre Office Writer*, *PDF* dokumenata, skeniranih slika papirnatih dokumenata i slično. U načelu ne postoji tehničko ograničenje veličine ili broja datoteka koje šaljemo u prilogu. Međutim, u praksi, klijenti e-pošte, poslužitelji i davatelji internetskih usluga implementiraju različita ograničenja na veličinu datoteka ili kompletne e-pošte, obično na 25 MB ili čak i manje.

Nakon što su 1995. godine okončana ograničenja za prijenos komercijalnog prometa putem Interneta, u široku primjenu ušli su **SMTP**, **POP3** i **IMAP** protokoli za korištenje elektroničke pošte.

Pri tome se **SMTP** (*Simple Mail Transfer Protocol*) obično koristi kao komunikacijski protokol za prijenos elektroničke pošte. Naime klijenti za elektroničku poštu koriste **SMTP** za slanje poruka e-pošte, korištenjem sljedećih TCP portova:

- Porta **25** (standardni port).
- Novijih portova: **465** ili **587**. Portovi **465** ili **587** obično se koriste za ***TLS*** ili ***SSL*** (kriptiranu) komunikaciju.

**SMTP** protokol je definiran u **RFC 788**, te proširen s **RFC 1869**, te **RFC 2476**, **RFC 2554** i **RFC 8314**.

S druge strane, za preuzimanje e-pošte iz elektroničkog poštanskog sandučića najčešće se koriste protokoli **POP3** ili **IMAP**.
Pri tome **POP3** (*Post Office Protocol*) koristi sljedeće TCP portove za komunikaciju:

- Port **109** za ***POP3*** inačice dva.
- Port **110** za ***POP3*** inačice tri.
- Port **995** za ***POP3*** preko kriptiranog TLS/SSL kanala.

Ako se koristi **IMAP** (*Internet Message Access Protocol*), definiran u **RFC 9051**, on može koristiti sljedeće TCP portove:

- Standardni port **143** ili port **993** za kriptiranu komunikaciju preko TLS/SSL.

Za čitanje i slanje elektroničke pošte koristimo programe koje zovemo klijenti e-pošte.
Ti programi mogu biti tekstualni (npr. *mail*, *mutt*, *elm*, *alpine*) ili grafički (npr. *Mozilla Thunderbird*, *Microsoft Outlook*).

Svaki od korisnika elektroničke pošte ima svoju elektroničku adresu koja ima oblik: **ime_pošiljatelja@ime_domene**, kao što su primjerice: **ivanhorvat@tvrtka.hr** ili **ivan.horvat@tvrtka.com**.

Dakle svi znakovi prije znaka **@** označavaju jedinstveno ime pošiljatelja, a poslije znaka **@**, domenu, kako je vidljivo dolje:

**ivan.horvat@tvrtka.com**

*Lokalni dio* *Domena*

Definicija i pravila oko adrese elektroničke pošte, definirana su u **RFC 5322** te proširena u **RFC 6854**.
***Lokalni dio*** adrese elektroničke pošte predstavlja ime korisnika (korisničkog računa).

Ovdje se mogu koristiti:

- Latinična slova abecede `A-Z` ili `a-z`.
- Brojevi `0-9` i ispisivi znakovi: `` !#$%&'*+-/=?^_`{|}~ ``.
- Točka (`.`), ali znak točke (`.`) ne smije biti prvi ili zadnji znak.

***Domena*** je DNS domena primatelja i/ili pošiljatelja elektroničke pošte u kojoj se mogu koristiti:

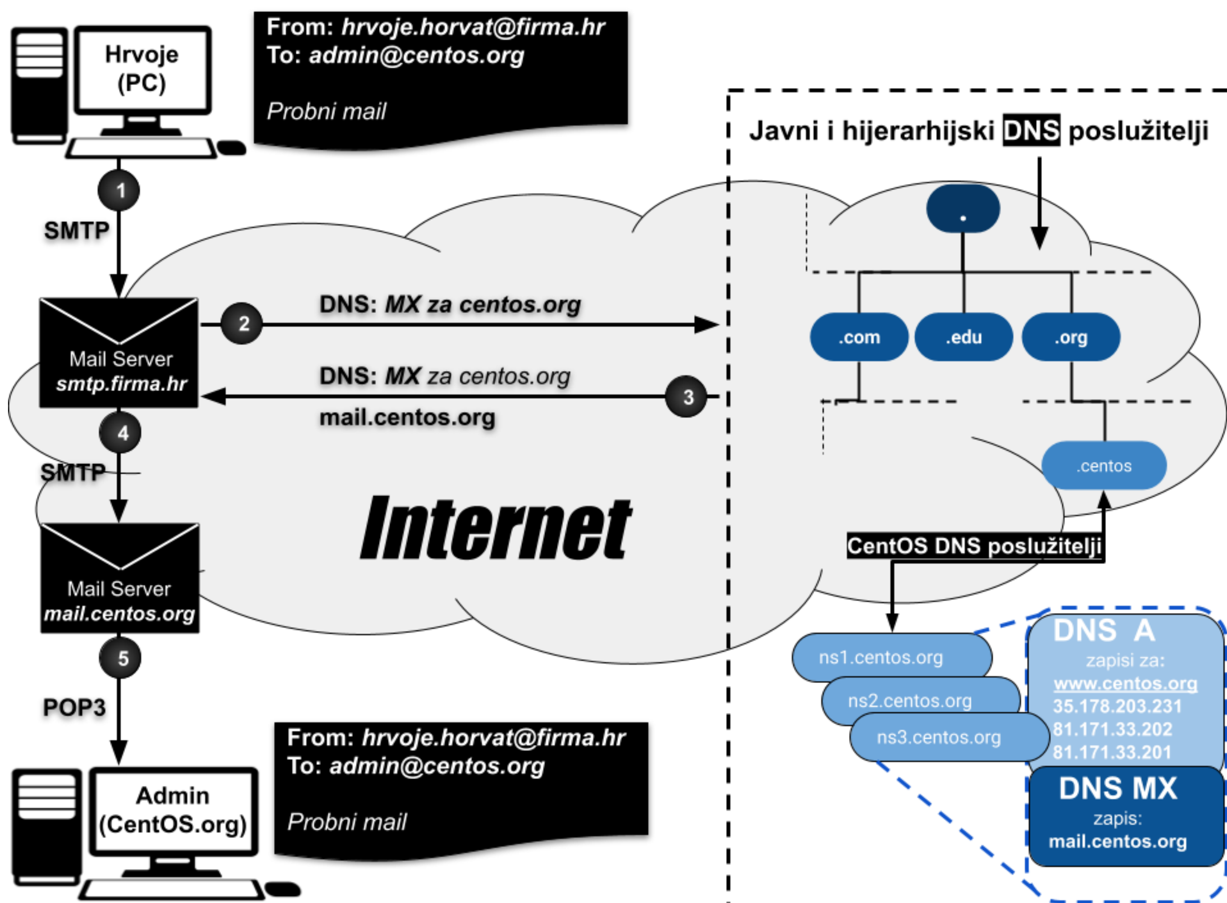
- Latinična slova abecede `A-Z` ili `a-z` i brojevi `0-9` s time da vršne domene ne mogu biti brojčane.
- Točka (`.`), ali znak točke (`.`) ne smije biti prvi ili zadnji znak te znak minus (`-`), koji ne smije biti prvi ili zadnji znak.



Za više detalja o domenama i domenskom sustavu, pogledajte poglavlja:
**25.8.5. DNS protokol.**
**25.8.5.5.7. MX zapis.**

Pogledajmo shemu komunikacije, pri slanju elektroničke pošte, upotrebom **SMTP** protokola, kako je vidljivo na slici 243.A.

**Slika 243.A. Slanje elektroničke pošte**




U primjeru na slici 243.A, vidimo sljedeće:

1. Na našem računalu (**PC**), klijent elektroničke pošte je konfiguriran na način da za slanje pošte koristi poslužitelj elektroničke pošte na adresi: smtp.firma.hr. Mi tada šaljemo e-poštu na adresu. admin@centos.org (izmišljena adresa). Ova e-pošta se automatski pomoću ***SMTP*** protokola, šalje na navedeni poslužitelj elektroničke pošte.
   Prema logici rada poslužitelj elektroničke pošte na adresi: smtp.firma.hr je **MSA** i **MTA** (pogledajte opise dolje).

2. Navedeni poslužitelj elektroničke pošte smtp.firma.hr (**MSA/MTA**) provjerava odredišnu adresu pošte (admin@centos.org) to jest njenu domenu. Pošto je njena domena: **centos.org**, on kontaktira ***DNS*** poslužitelje koji su mu konfigurirani na sustavu, slanjem ***DNS*** upita. U ***DNS*** upitu koji on šalje, od ***DNS*** poslužitelja odgovornog za domenu **centos.org**, traži se informacija za ovu domenu, koji je poslužitelj elektroničke pošte odgovoran za nju, kako je definirano u **RFC 5321 - stranica 69**.

3. Od odgovornog ***DNS*** poslužitelja za domenu: **centos.org**, dobivamo, da je poslužitelj elektroničke pošte za ovu domenu, poslužitelj na adresi: mail.centos.org, za koji dobivamo i njegovu pripadajuću IP adresu.

4. Tek sada naš poslužitelj elektroničke pošte smtp.firma.hr (**MSA/MTA**) šalje pomoću ***SMTP*** protokola, našu elektroničku poštu na odredišni poslužitelj elektroničke pošte, unutar domene na kojoj se nalazi korisnik kojem i šaljemo e-poštu. Dakle on šalje našu elektroničku poštu na poslužitelj: mail.centos.org to jest na njegovu pripadajuću IP adresu.

5. Pošta je zaprimljena od strane poslužitelja elektroničke pošte mail.centos.org, te ju on sprema u memoriju (obično na disk), te čeka da se s druge strane s klijentom elektroničke pošte, netko logira na sustav, s korisničkim računom i pripadajućom lozinkom, za korisnika **admin**. Tada taj klijent elektroničke pošte (**MDA**) pomoću ***POP3*** ili ***IMAP*** protokola, u bilo kojem trenutku može dohvatiti novo pristiglu e-poštu s poslužitelja. Dakle ovom klijentu e-pošte je zadani poslužitelj elektroničke pošte, njegov domenski poslužitelj elektroničke pošte mail.centos.org, s kojeg prima i na kojeg šalje svoju e-poštu. Pri tome se za primanje koriste ***POP3*** ili ***IMAP***, a za slanje ***SMTP*** protokol.

**Važno je razumjeti nekoliko osnovnih pojmova**:

- Klijent e-pošte (*E-mail client*), čitač e-pošte ili, formalnije, korisnički agent za poruke **MUA** (*Mail user agent*) ili korisnički agent e-pošte, računalni je program koji se koristi za pristup i upravljanje korisničkom e-poštom.
- Agent za slanje poruka to jest ***MSA*** (*Mail Submission Agent*) ili agent za slanje pošte je računalni program ili softverski servis koji prima poruke elektroničke pošte od korisničkog agenta za poštu (***MUA***) i surađuje s agentom za prijenos pošte (**MTA**) za dostavu pošte. Većina, ako ne i svi ***MTA***-ovi obavljaju i funkciju ***MSA***.
- Agent za prijenos pošte to jest ***MTA*** (*Mail Transport Agent*) prenosi poruke e-pošte između računala koristeći ***SMTP*** protokol. Prijenos poruke (e-pošte) može uključivati nekoliko MTA-ova u tranzitu, dok poruka odlazi na željeno odredište. ***Mail Transport Agenti* odnosno poslužitelji e-pošte su primjerice: *Dovecot*, *Exim*, *Postfix* i *Sendmail*.**
- ***MTA*** poziva agenta za isporuku pošte to jest ***MDA*** (*Mail Delivery Agent*) kako bi arhivirao dolaznu e-poštu u poštanski sandučić odgovarajućeg korisnika. U mnogim slučajevima, **MDA** je zapravo takozvani agent lokalne dostave (***LDA***), kao što su programi `mail` ili `procmail`. Bilo koji program koji obrađuje poruku (e-poštu) za isporuku, do točke u kojoj je može pročitati aplikacija klijenta e-pošte može se smatrati ***MDA***-om.
- Korisnički agent pošte to jest ***MUA*** (*Mail User Agent*) je sinonim za aplikaciju klijenta e-pošte. ***MUA*** je program koji, u najmanju ruku, omogućuje korisniku čitanje i sastavljanje poruka e-pošte. **MUA**-i mogu biti grafički, kao što su ***Thunderbird*** i ***Microsoft Outlook***, ili imati jednostavna tekstualna sučelja, kao što su `mail`, `procmail` ili `mutt`. ***MUA*** programi odrađuju sljedeće zadatke:
    - Dohvaćanje poruka pomoću **POP3** ili **IMAP** protokola.
    - Postavljanje poštanskih sandučića (e-pošte) za pohranu poruka.
    - Slanje odlaznih poruka na **MTA** sustav.

**Izvori informacija: (1128),(1129),(1130),(1131),(1132),(1133), RFC 561, RFC 788, RFC 1652, RFC 1869, RFC 2476, RFC 2554, RFC 4021, RFC 5321, RFC 5322, RFC 6530, RFC 6531, RFC 6532, RFC 6533, RFC 6854, RFC 8314, RFC 9051.**

---

## 25.8.8.1. Konfiguracija i rad s elektroničkom poštom

Za konfiguraciju i rad s elektroničkom poštom, koristit ćemo tekstualni program za rad s elektroničkom poštom, naziva `mail`. Dakle on je klijent za elektroničku poštu koji radi u tekstualnom (*CLI*) načinu rada.
Program `mail` dolazi u programskom paketu ***mailx***, jer je u današnje vrijeme on implementiran u novijem programu `mailx` te se pozivanjem programa `mail` zapravo poziva program `mailx`. On nam daje mogućnost korištenja: ***SMTP***, ***POP3*** i ***IMAP*** protokola za rad s elektroničkom poštom. Prvo ga instalirajmo:

```
yum install -y mailx
```

Sada ga možemo konfigurirati, a njegova konfiguracija se nalazi u datoteci: `/etc/mail.rc`.
Uredit ćemo ovu datoteku i u nju dodati sljedeću konfiguraciju varijabli, koju ćemo potom objasniti:

```
set smtp=mail.t-com.hr:25
set from="root@opensource-osijek.org"
```

Konkretno, mi koristimo postavke za *T-com* ***SMTP*** poslužitelja (`mail.t-com.hr` na portu `25`), koji ne traži autentikaciju, u slučaju slanja elektroničke pošte na njega.

Međutim, u slučaju kada naš ***SMTP*** poslužitelj zahtjeva autentikaciju, trebali bi dodati sljedeće retke:

```
set smtp-auth=login
set smtp-auth-user=KORISNIK@DOMENA
set smtp-auth-password=LOZINKA
```

- `smtp-auth=login` - naznačava ***SMTP*** poslužitelju da se želimo autenticirati.
- `smtp-auth-user` - definira koji je naš korisnički račun za e-poštu (pr. `hrvoje.horvat@firma.hr`)
- `smtp-auth-password` - definira lozinku našeg korisničkog računa.

U slučaju kada ***SMTP*** poslužitelj traži i TLS/SSL enkripciju, ovdje imamo cijeli niz opcija, koje su opisane u manualu naredbe mail (**`man mail`**), a koje sada nećemo detaljnije objašnjavati i istraživati (to prepuštamo vama).
Sada probajmo poslati kratku elektroničku poštu na (*izmišljenu adresu*): `proba.mail@gmail.com`, iz jednog retka:

```
echo "Test" | mail -s "Proba" proba.mail@gmail.com
```

Inače se elektronička pošta s programom `mail` šalje pozivanjem programa s navođenjem e-pošte primatelja, primjerice:

```
mail proba.mail@gmail.com
```

Nakon toga upisujemo naslov to jest predmet slanja (`Subject:`) te pišemo sadržaj e-pošte.

```
Subject: Test
Probna poruka
```

Kada smo završili, potrebno je stisnuti kombinaciju tipki **CTRL D**, kako bi e-pošta bila poslana na odredište; u ovom primjeru na: `proba.mail@gmail.com`.



Poštanski pretinci se u pravilu za svakog korisnika snimaju u vršni direktorij (mapu): `/var/mail/`

Pogledajmo i neke od opcija pri slanju elektroničke pošte, upotrebom programa `mail`:

- `-a FILE` – dodavanje privitka (datoteke imena **FILE**) uz elektroničku poštu.
- `-b XX` – *Blind carbon copy* (**BCC**) slanje tzv. slijepe kopije na odredišnu elektroničku poštu **XX**.
- `-c XX` – *Carbon copy* (**CC**) slanje kopije na odredišnu elektroničku poštu **XX**.

Snimanje mrežnih paketa koji koriste ***SMTP*** protokol (port 25), na sučelju `enp0s3`, možemo napraviti na sljedeći način:

```
tcpdump -i enp0s3 port 25 -s0 -vv
```

## Pogledajmo mrežnu komunikaciju prilikom slanja e-pošte upotrebom SMTP protokola.

**SMTP** poslužitelj (**MTA**) je `mail.t-com.hr` na TCP portu `25`.
Izvorišna e-pošta je `root@opensource-osijek.org`.
Naše računalo klijent e-pošte ima IP adresu: `192.168.1.129`.
Odredišna e-pošta je: `proba.mail@gmail.com` .

Komunikacije teće na sljedeći način:

1. Naše računalo traži koju IP adresu ima `mail.t-com.hr` slanjem **DNS** upita na **DNS** poslužitelj.
2. **DNS** poslužitelj odgovora i daje nam **IP** adresu od poslužitelja `mail.t-com.hr`.
3. Naše računalo ostvaruje **TCP** vezu prema `mail.t-com.hr` (s **TCP SYN**, **TCP SYN ACK** i **TCP ACK**), na port **25**.
4. Od `mail.t-com.hr` dobivamo odgovor preko **TCP** protokola s porta **25**, dakle sada kao **SMTP** poruku:
   **SMTP**: *Response code*: **Service Ready (220)**, *Response parameter*: **ESMTP Rock and Roll**.
5. Mi (`192.168.1.129`) odgovaramo sa **SMTP**: *Command*: **Hello**, *Request parameter*: **localhost**.
6. Dobivamo **SMTP** odgovor: *Response code*: **Requested mail action okay, completed (250)**.
7. Mi šaljemo **SMTP**: *Command*: **Mail,** *Request parameter:* ***FROM:*** `root@opensource-osijek.org`.
8. Dobivamo potvrdu da je mail koji smo naznačili kao izvorišni (iz prethodnog koraka) dobar:
   **SMTP**: *Response code*: **Requested mail action okay, completed (250).**



*Većina poslužitelja elektroničke pošte provjerava izvorišnu mail adresu to jest primarno njenu domenu te ako je ona uredna i ako se eventualno ne nalazi na crnoj listi, komunikacija se nastavlja. U protivnom nam poslužitelj vrača grešku.*

9. Mi šaljemo **SMTP**: *Command*: **RCPT,** *Request parameter:* ***TO:*** `proba.mail@gmail.com`.
10. Dobivamo potvrdu da je mail koji smo naznačili kao odredišni (iz prethodnog koraka) dobar (radi se provjera valjanosti):
    **SMTP**: *Response code*: **Requested mail action okay, completed (250).**
11. Mi šaljemo **SMTP**: *Command*: **DATA.**
12. Zatim slijede podaci koje šaljemo **SMTP** poslužitelju, unutar e-pošte, a to su zaglavlje (e-pošte: izvor/odredište) i sadržaj:

```
From: root@opensource-osijek.org\r\n
To: proba.mail@gmail.com\r\n
Subject: Proba\r\n
Message-ID: <61dd5281.eBL5hsmjwPOs7bmd%root@opensource-osijek.org>\r\n
User-Agent: Heirloom mailx 12.5 7/5/10\r\n
MIME-Version: 1.0\r\n
Content-Type: text/plain; charset=us-ascii\r\n
Content-Transfer-Encoding: 7bit\r\n
\r\n
Test\r\n
```

✓ U ispisu iz koraka **12** dodatno vidimo i ***MIME*** inačicu (`MIME-Version: 1.0`) i vrstu sadržaja (`Content-Type: text/plain`), kao i način kodiranja (`Content-Transfer-Encoding: 7bit`).
Pošto smo u sadržaju (tijelu) poslali samo tekstualnu poruku (`Test`), odmah ju i vidimo ovdje. Da smo imali i prilog (datoteke), one bi bile posebno kodirane; obično s **base64** standardom za kodiranje.

13. Od **SMTP** poslužitelja (`mail.t-com.hr`) dobivamo **SMTP** poruku:
    *Response code*: **Requested mail action okay, completed (250)** koja potvrđuje našu prethodnu poruku (iz broja **10**).
14. Da je trebalo još nešto prenijeti od podataka unutar sadržaja e-pošte ili priloga (nekih datoteka), to bi se prenosilo u narednim porukama. Pošto je naša poruka kratka i sve je već preneseno, mi **SMTP** poslužitelju šaljemo poruku da smo završili s prijenosom s **SMTP** porukom: *Command*: **QUIT**.
15. Od **SMTP** poslužitelja (`mail.t-com.hr`) dobivamo **SMTP** poruku da se potvrđuje zatvaranje komunikacije:
    *Response code:* **Service closing transmission channel (221)**.
16. Nakon toga slijedi zatvaranje veze na razini **TCP** protokola, s obije strane (obostrani **TCP FIN ACK**).

Pogledajmo i kako izgledaju navedeni paketi snimljeni u programu ***tcpdump*** te otvoreni i filtrirani s programom ***Wireshark***:

**Slika 243.B. Slanje elektroničke pošte SMTP protokolom (snimka mrežnih paketa)**

*LAN

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr == 192.168.1.129ü

| No. | Time | Source | Destination | Protocol | Lengt | Info |
|---|---|---|---|---|---|---|
| 40 | 2022-01-11 09:48:48,239320 | 192.168.1.129 | 192.168.1.1 | DNS | 73 | Standard query 0xfb59 A mail.t-com.hr |
| 41 | 2022-01-11 09:48:48,239338 | 192.168.1.129 | 192.168.1.1 | DNS | 73 | Standard query 0x805b AAAA mail.t-com.hr |
| 42 | 2022-01-11 09:48:48,246259 | 192.168.1.1 | 192.168.1.129 | DNS | 123 | Standard query response 0x805b AAAA mail.t-com.hr SOA dn |
| 43 | 2022-01-11 09:48:48,253281 | 192.168.1.1 | 192.168.1.129 | DNS | 234 | Standard query response 0xfb59 A mail.t-com.hr A 195.29. |
| 44 | 2022-01-11 09:48:48,253723 | 192.168.1.129 | mail.t-com.hr | TCP | 74 | 55266 → 25 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PER |
| 45 | 2022-01-11 09:48:48,266154 | mail.t-com.hr | 192.168.1.129 | TCP | 74 | 25 → 55266 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=146 |
| 46 | 2022-01-11 09:48:48,266419 | 192.168.1.129 | mail.t-com.hr | TCP | 66 | 55266 → 25 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=17617 |
| 47 | 2022-01-11 09:48:48,279944 | mail.t-com.hr | 192.168.1.129 | SMTP | 106 | S: 220 ls266.t-com.hr ESMTP Rock and Roll |
| 48 | 2022-01-11 09:48:48,280195 | 192.168.1.129 | mail.t-com.hr | TCP | 66 | 55266 → 25 [ACK] Seq=1 Ack=41 Win=64256 Len=0 TSval=1761 |
| 49 | 2022-01-11 09:48:48,280550 | 192.168.1.129 | mail.t-com.hr | SMTP | 82 | C: HELO localhost |
| 50 | 2022-01-11 09:48:48,291956 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [ACK] Seq=41 Ack=17 Win=5888 Len=0 TSval=1978 |
| 51 | 2022-01-11 09:48:48,291956 | mail.t-com.hr | 192.168.1.129 | SMTP | 86 | S: 250 ls266.t-com.hr |
| 52 | 2022-01-11 09:48:48,292215 | 192.168.1.129 | mail.t-com.hr | TCP | 66 | 55266 → 25 [ACK] Seq=17 Ack=61 Win=64256 Len=0 TSval=176 |
| 53 | 2022-01-11 09:48:48,292332 | 192.168.1.129 | mail.t-com.hr | SMTP | 106 | C: MAIL FROM:<root@opensource-osijek.org> |
| 54 | 2022-01-11 09:48:48,303932 | mail.t-com.hr | 192.168.1.129 | SMTP | 80 | S: 250 2.1.0 Ok |
| 55 | 2022-01-11 09:48:48,304184 | 192.168.1.129 | mail.t-com.hr | TCP | 66 | 55266 → 25 [ACK] Seq=57 Ack=75 Win=64256 Len=0 TSval=176 |
| 56 | 2022-01-11 09:48:48,304330 | 192.168.1.129 | mail.t-com.hr | SMTP | 102 | C: RCPT TO: <proba.mail@gmail.com> |
| 57 | 2022-01-11 09:48:48,356097 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [ACK] Seq=75 Ack=93 Win=5888 Len=0 TSval=1978 |
| 58 | 2022-01-11 09:48:49,158035 | mail.t-com.hr | 192.168.1.129 | SMTP | 80 | S: 250 2.1.5 Ok |
| 59 | 2022-01-11 09:48:49,158369 | 192.168.1.129 | mail.t-com.hr | SMTP | 72 | C: DATA |
| 60 | 2022-01-11 09:48:49,168700 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [ACK] Seq=89 Ack=99 Win=5888 Len=0 TSval=1978 |
| 61 | 2022-01-11 09:48:49,172052 | mail.t-com.hr | 192.168.1.129 | SMTP | 103 | S: 354 End data with <CR><LF>.<CR><LF> |
| 62 | 2022-01-11 09:48:49,172404 | 192.168.1.129 | mail.t-com.hr | SMTP | 397 | C: DATA fragment, 331 bytes |
| 64 | 2022-01-11 09:48:49,223036 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [ACK] Seq=126 Ack=430 Win=6912 Len=0 TSval=19 |
| 65 | 2022-01-11 09:48:49,223336 | 192.168.1.129 | mail.t-com.hr | SMTP/IMF | 69 | from: root@opensource-osijek.org, subject: Proba, (text |
| 66 | 2022-01-11 09:48:49,233973 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [ACK] Seq=126 Ack=433 Win=6912 Len=0 TSval=19 |
| 69 | 2022-01-11 09:48:49,306064 | mail.t-com.hr | 192.168.1.129 | SMTP | 103 | S: 250 2.0.0 Ok: queued as 65C8112023E |
| 70 | 2022-01-11 09:48:49,306476 | 192.168.1.129 | mail.t-com.hr | SMTP | 72 | C: QUIT |
| 71 | 2022-01-11 09:48:49,318792 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [ACK] Seq=163 Ack=439 Win=6912 Len=0 TSval=19 |
| 72 | 2022-01-11 09:48:49,318792 | mail.t-com.hr | 192.168.1.129 | SMTP | 81 | S: 221 2.0.0 Bye |
| 73 | 2022-01-11 09:48:49,319131 | 192.168.1.129 | mail.t-com.hr | TCP | 66 | 55266 → 25 [FIN, ACK] Seq=439 Ack=178 Win=64256 Len=0 TS |
| 74 | 2022-01-11 09:48:49,319235 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [FIN, ACK] Seq=178 Ack=439 Win=6912 Len=0 TSv |
| 75 | 2022-01-11 09:48:49,319457 | 192.168.1.129 | mail.t-com.hr | TCP | 66 | 55266 → 25 [ACK] Seq=440 Ack=179 Win=64256 Len=0 TSval=1 |
| 76 | 2022-01-11 09:48:49,332014 | mail.t-com.hr | 192.168.1.129 | TCP | 66 | 25 → 55266 [ACK] Seq=179 Ack=440 Win=6912 Len=0 TSval=19 |

Frame 72: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface \Device\NPF {2E3856FF-0E88-4736-B676-4F142E26AE18}, id 0

"◆" was unexpected in this context. | Packets: 242 · Displayed: 36 (14.9%) · Dropped: 0 (0.0%) | Profile: Default



U slučaju kada vam komunikacija prema poslužitelju elektroničke pošte ne radi, potrebno je pratiti mrežne pakete u komunikaciji i otkriti u kojem koraku imate problem i s kojom greškom.

Za primanje e-pošte, potrebna je dodatna konfiguracija datoteke `/etc/mail.rc`, ovisno o tome koristi li se vršni ***POP3*** ili ***IMAP*** poslužitelj elektroničke pošte i s kojim postavkama to jest parametrima rada.

Provjeriti imamo li pristiglu elektroničku poštu možemo, samo pozivanjem programa:

```
mail
```



Osim tekstualnih klijenata e-pošte, poput programa `mail`, `procmail` ili `mutt`, postoje i grafički programi poput ***Thunderbird*** ili ***Microsoft Outlook***. Program `mail` dostupan je i na drugim operativnim sustavima poput: ***Sun Solaris***, ***Open Unix***, ***FreeBSD***, ***HP-UX*** i drugima.

**Izvori informacija: (1128),(1129),(1134), `man mail`, `man mailx`, `man tcpdump`, RFC 5321.**

## 25.8.9. (R)syslog servis i usluga

Primarna namjena *syslog* servisa je prikupljanje poruka sustava i spremanja istih u takozvane dnevničke (*log)* datoteke. Međutim *Syslog* je servis i protokol za slanje i primanje poruka (obavijesti), o stanju sustava, u određenom formatu i to potencijalno s različitih uređaja na mreži. Obično je to datoteka: `/var/log/messages`, ali i druge datoteke u `/var/log/` direktoriju, poput primjerice: `secure`,`maillog`,`cron`,`spooler` i `boot.log` datoteka, na računalu na kojem je ovaj servis i pokrenut.
Međutim njegova sekundarna namjena je prihvaćanje poruka s drugih računala ili mrežnih uređaja na centraliziranom mjestu.
Navedene poruke uključuju vremenske oznake, poruke događaja, važnosti, IP adrese domaćina, dijagnostiku i još mnogo toga.
U smislu ugrađenih razina važnosti, može se komunicirati u rasponu između razina: (**0**) hitne situacije, (**1**) upozorenja, (**2**) nestabilnost sustava, (**3**) greške, (**4**) upozorenja, (**5**) obavijesti, (**6**) informacije i (**7**) otklanjanje pogrešaka (engl. *Debugging*).
*Syslog* format i protokol definiran je u: **RFC 3164** te proširen s **RFC 5424**.
Prema načinu rada syslog mrežni protokol funkcionira prema modelu: **klijent → poslužitelj**.
***Syslog protokol standardno koristi UDP za transport na portu 514, ali novije implementacije koriste i TCP na portu 6514***.
*Syslog* servis koristi se osim Linuxa i na većini Unixa, a u novije vrijeme, zamijenjen je s ***rsyslog*** servisom. ***Rsyslog*** servis osim klasične podrške za ***UDP***, podržava i upotrebu ***TCP*** transportnog protokola. *Rsyslog* možemo instalirati na sljedeći način:

```
yum -y install rsyslog
```

Nakon instalacije, potrebno ga je pokrenuti i aktivirati. što ćemo postići sa sljedeće dvije naredbe (za ***RedHat/CentOS 7+***):

```
systemctl start rsyslog
systemctl enable rsyslog
```

Nakon što se pokrenuo, on će primarno kreirati *log* datoteku: `/var/log/messages` u koju će početi zapisivati *log* poruke.
Njegova konfiguracijska datoteka je: `/etc/rsyslog.conf` u kojoj su definirana sva pravila ponašanja *(r)syslog* servisa.

**Mrežni syslog poslužitelj**
U slučaju kada na našem **rsyslog (**`rsyslogd`**)** servisu želimo omogućiti i prihvat *syslog* poruka s drugih računala i uređaja na mreži, imamo dvije opcije. Ako želimo omogućiti prihvat **UDP** *syslog* poruka, tada moramo odkomentirati sljedeće retke:

```
module(load="imudp") # needs to be done just once
input(type="imudp" port="514")
```

← ispred svakog ovog retka ne smije biti znak **#** to jest moramo ga obrisati. Uređujemo datoteku: `/etc/rsyslog.conf`.
Ako pak želimo omogućiti prihvat **TCP** *syslog* poruka, tada moramo odkomentirati sljedeće retke (ili gornje ili ove dolje):

```
module(load="imtcp") # needs to be done just once
input(type="imtcp" port="514")
```

← Niti ispred svakog ovog retka ne smije biti znak **#**. Pri tome ponovno uređujemo datoteku: `/etc/rsyslog.conf`.
S navedenim smo omogućili ili upotrebu ***UDP porta 514*** ili ***TCP porta 514*** za prihvati *syslog* poruka preko mreže.
Sada moramo restartati naš *rsyslog* servis:

```
systemctl restart rsyslog
```

I na kraju provjerimo imamo li otvoren port 514:

```
netstat -tunap | grep 514

udp        0      0 0.0.0.0:514            0.0.0.0:*                 1678/rsyslogd
udp6       0      0 :::514                 :::*                      1678/rsyslogd
```

Vidimo da imamo otvoren **UDP** port **514** (za IPv4 [***udp***] i za IPv6 [***udp6***]).
Sada je naš poslužitelj spreman za prihvat *syslog* poruka preko mreže, konkretno upotrebom **UDP** protokola na portu **514**.

**Mrežni syslog klijent**
U slučaju kada na našem **rsyslog (**`rsyslogd`**)** servisu želimo omogućiti slanje *syslog* poruka na centralni *syslog* poslužitelj na mreži, tada u našoj konfiguracijskoj datoteci `/etc/rsyslog.conf`, moramo na kraj dodati sljedeće retke konfiguracije.
Naime ako želimo sve naše *syslog* poruke slati na centralni mrežni *syslog* poslužitelj pomoću **UDP** protokola, ako je *syslog* mrežni poslužitelj na IP adresi: **192.168.1.100**, tada trebamo dodati:

```
*.* @192.168.1.100:514
```

Odnosno ako želimo slati *syslog* poruke upotrebom **TCP** protokola na udaljeni poslužitelj, na IP adresi: **192.168.1.100**:

```
*.* @@192.168.1.100:514
```

I potom restartajmo naš (u ovom primjeru klijentski) *rsyslog* servis, koji će sada sve poruke snimati lokalno, ali i slati na mrežu.

```
systemctl restart rsyslog
```



U oba slučaja, ako imate vatrozid (*firewall*), ne zaboravite dodati pravila za pristup na ***TCP*** ili ***UDP*** port ***514***!

Ako u stvarnom vremenu želite pratiti sve *syslog* poruke, to možete napraviti (na klijentu ili poslužitelju) na sljedeći način:

```
tail -f /var/log/messages
```

**Izvori informacija: (690),(1181),(1182),(1183),(1184),** **`man rsyslogd`**,**`man rsyslog.conf`**, **RFC 3164**, **RFC 5424.**

# 26. Mrežni servisi i usluge

Mrežni servisi ili prema linux terminologiji *daemon-i* zaduženi su za pojedine mrežne protokole. U pravilu je jedan servis zadužen za jedan protokol ili eventualno skupinu protokola. Dalje u tekstu objasnit ćemo njih samo nekolicinu.

## 26.1. Network Manager

Servis *NetworkManager* zadužen je za kontrolu i konfiguraciju mrežnih kartica i mrežnih sučelja. On je zadužen i za praćenje stanja mrežnih kartica odnosno mrežnih sučelja (pr. jesu li aktivne ili nisu) te za njihovo aktiviranje prema potrebi.
*NetworkManager* se sastoji od servisa koji mora biti stalno aktivan (i pokrenut) te potencijalno GNOME *apleta* unutar grafičkog sučelja, ako ga koristimo, a koji prikazuje stanje mrežnih kartica te sâmog grafičkog konfiguracijskog alata.
Iz grafičkog konfiguracijskog alata se radi sva konfiguracija mrežnih sučelja. On podržava razne vrste mrežnih sučelja:

- *Ethernet i Wireless,* Mobilnih kartica (*3G*, *4G*, 5G), kao i: *DSL, PPPoE* i drugih.

Što se tiče sâme konfiguracije mrežnih sučelja, u njemu je moguće definirati sve IP parametre:

- Statički ili dinamički (DHCP):
  - IP adresu i pripadajuću masku mreže te podrazumijevani usmjerivač*/pristupnik (Engl. Default Gateway).*
  - DNS poslužitelje.
- VPN konekcije i druge parametre rada mrežnih sučelja.

Osim toga *NetworkManager* ima ***API*** sučelje preko ***D-Bus*** sustava koji mu omogućava provjeru, nadzor i konfiguraciju mrežnih sučelja. *NetworkManager* je standardno instaliran od inačice **RedHat/CentOS 6.x**.
U slučaju kada nije instaliran, a stvarno vam je potreban, moguće ga je instalirati na sljedeći način:

```
yum -y install NetworkManager
```

Kao i svaki servis, potrebno je da bude pokrenut automatski tijekom svakog pokretanja operativnog sustava.
To ćemo osigurati sljedećom naredbom (za ***RedHat***/***CentOS*** 6.x):

```
chkconfig NetworkManager on
```



I na kraju, mi ga **nećemo** koristiti (za ***RedHat/CentOS 6*** ili ***7***) jer želimo sve konfiguracije odrađivati ručno i biti svjesni koje datoteke su zaslužene za konfiguraciju raznih dijelova mrežnog podsustava, jer stalno govorimo o upotrebi Linuxa kao poslužitelja. ***U slučaju korištenja RedHat/CentOS 8.x ovaj servis mora biti aktivan (o detaljima malo kasnije)!.***

**Za RedHat/CentOS 6.x**
Isključimo ga trajno:

```
chkconfig NetworkManager off
```

Te ga zaustavimo:

```
service NetworkManager stop
```

**Za RedHat/CentOS 7.x (ne i za RedHat 8.x)**
Isključimo ga trajno:

```
systemctl disable NetworkManager
```

Te ga zaustavimo:

```
systemctl stop NetworkManager
```



Servis *NetworkManager* dobro je koristiti na osobnim računalima (stolnim, laptop ili tabletima) jer na vrlo jednostavan, a u slučaju kada koristimo grafičko sučelje **X Window** i na grafički lijep način možemo vrlo brzo konfigurirati našu mrežu i u konačnici naš izlaz prema internetu. Stoga na navedenim računalima u pravilu **ne isključujemo** *NetworkManager*.



**Za RedHat/CentOS 8.x** i novije: ***Network Manager*** je nužan za rad mreže pa ga ne smijemo isključivati!

Stoga ga trajno aktivirajmo i pokrenimo, ako smo ga slučajno zaustavili:

```
systemctl enable NetworkManager
systemctl start NetworkManager
```



Od ***RH/CentOS*** v.8+ ***Network Manager*** zamjenjuje servis ***network*** zadužen za rad s mrežnim sučeljima (konfiguracija i pokretanje, spuštanje te restart), stoga ga **nemojte** zaustavljati. Dakle ovo vrijedi samo za ***RedHat/CentOS*** **v.8.x** i novije.

## Red Hat 9.x.

Do *Red Hat* Linuxa 9.x *NetworkManager* servis pohranjivao je mrežne konfiguracije u takozvanom *ifcfg* formatu, unutar direktorija `/etc/sysconfig/network-scripts/`. Međutim od *Red Hat* 9+ datoteke pohranjene u `/etc/sysconfig/network-scripts/` također poznate kao *ifcfg* datoteke više neće biti primarna lokacija niti format pohrana za mrežne konfiguracijske datoteke.
Iako su *ifcfg* datoteke još uvijek dostupne, one više nisu zadani format niti koriste zadanu lokaciju na kojoj *NetworkManager* pohranjuje mrežne profile odnosno konfiguracije mrežnih sučelja i parametara rada mreže. Naime razne distribucije Linuxa koriste konfiguracije mrežnih sučelja na različitim lokacijama. Primjerice *Debian* distribucija Linuxa i njeni derivati tradicionalno koriste datoteku `/etc/network/interfaces`. Dok *Red Hat* i srodne distribucije Linuxa koriste direktorij `/etc/sysconfig/network-scripts/` s pripadajućim *ifcfg* datotekama, i tako dalje.

Pošto je sve više distribucija Linuxa prihvatilo *NetworkManager*, ključne datoteke u direktoriju `/etc/NetworkManager/system-connections/` postale su jedino mjesto za konfiguraciju mreže na Linuxu.

Konfiguracija se sastoji od sekcija; primjerice: `[connection]`,`[ipv4]`,`[ethernet]`, te opcija (tzv. ***ključeva***) unutar njih.

Pogledajmo kako na sustavu vidjeti koji je redoslijed primarnih i sekundarnih mrežnih konfiguracijskih datoteka odnosno gdje/odakle će se koristiti i snimati:

```
NetworkManager --print-config
[main]
plugins=keyfile,ifcfg-rh
```

Važan dio unutar sekcije `[main]` je `plugins`, koji je standardno postavljen kao kod nas;:

- Prvo `keyfile`→ što znači *NetworkManager* skripte (`/etc/NetworkManager/system-connections/`)
- A tek potom `ifcfg-rh` što označava *ifcfg* skripte (`/etc/sysconfig/network-scripts/` direktorij).

Naime, novi profili mrežnih veza sada se pohranjuju u takozvanom formatu datoteke ključa (engl. *key file*), koji ima mnoge prednosti. Na primjer, ovaj format se temelji na **INI** datoteci i može se lako analizirati i generirati. Svaki odjeljak (sekcija) u datotekama ključeva *NetworkManagera* odgovara *NetworkManager* nazivu postavke kako je opisano u *man* stranicama za: `nm-settings` i `nm-settings-keyfile`.

Svaki par ključ-vrijednost prema sekcijama konfiguracije naveden je u specifikaciji postavki navedenih *man* stranica. *NetworkManager* se temelji na konceptu profila veza. Ovi profili veze sadrže konfiguraciju mreže. Kada *NetworkManager* aktivira profil veze na mrežnom sučelju, konfiguracija će se primijeniti i uspostavit će se aktivna mrežna veza.
Korisnici mogu slobodno kreirati onoliko profila veza koliko god im odgovara.

***Network Manager konekcije (engl. profiles)*** odnose se na specifične, ugniježđene, neovisne grupe postavki koje opisuju sve konfiguracije potrebne za povezivanje s određenom mrežom. Poziva se jedinstvenim identifikatorom koji se naziva **UUID**. Mrežna veza je vezana za jednu određenu vrstu sučelja, ali ne nužno i za određeno hardversko sučelje (fizičku mrežnu karticu). Sastoji se od jednog ili više objekata postavki.

Postavke ***Network Managera*** odnose se na skupine povezanih parova ključ-vrijednost koji opisuju određeni dio veze (profil).

Samo manji dio ključeva iz nekoliko sekcija (**`[xy]`**) te njihovih dopuštenih vrijednosti, opisat ćemo u tablici:

| **`[sekcija]` - Ključ** | **Zadana vrijednost** | **Opis** |
|---|---|---|
| **`[connection]`** `auth-retries` | *1* | Broj ponovnih pokušava provjere autentičnosti. Nula znači unedogled. Ako nije definirano, pokušavat će se tri puta. |
| **`[connection]`** `autoconnect` | *TRUE* | Treba li ili ne *NetworkManager* automatski aktivirati vezu kada resursi za vezu budu dostupni. |
| **`[connection]`** `autoconnect-retries` | *-1* | Broj pokušaja povezivanja pri automatskoj aktivaciji prije odustajanja. Nula znači zauvijek, -1 znači globalnu zadanu vrijednost (4 puta ako se ne poništi). Postavljanje na 1 znači pokušaj aktivacije samo jednom prije blokiranja automatskog povezivanja. |
| **`[connection]`** `gateway-ping-timeout` | *0* | Ako je veći od nule, odgoditi IP adresiranje dok ne istekne vremensko ograničenje ili dok IP zadani usmjerivač (*default gateway*) ne odgovori na *ping*. |
| **`[connection]`** `id` | *Naziv* | Naziv/identifikator mrežnog profila (pr. **enp0s3**) |
| **`[connection]`** `interface-name` | *Naziv* | Ime mrežnog sučelja na koje se odnosi profil (pr. **LAN**). |
| **`[connection]`** `uuid` | *Broj* | Jedinstveni identifikator veze (engl. *universally unique identifier*). |

| **[sekcija]**<br>**Ključ** | **Zadana vrijednost** | **Vrijednost / Opis** |
|---|---|---|
| **[connection]**<br>type | *Naziv* | Vrsta mrežnog sučelja, primjerice:<br>• ethernet – *za Ethernet mreže (LAN mreže).*<br>• wireless – *za bežične mreže.*<br>• pppoe - *za Point to point over ethernet mreže.*<br>• bridge – *za Bridge mrežna sučelja.*<br>• bond – *za bond mrežna sučelja.*<br>• vlan – *za VLAN mrežna sučelja.*<br>• veth – *za VETH mrežna sučelja.* .... |
| **[ethernet]**<br>mac-address<br>speed<br>duplex<br>auto-negotiate<br>mtu | *MAC adresa*<br>*Brzina/duplex* | *MAC* adresa mrežnog sučelja. Primjerice:<br>mac-address=00:40:21:1f:ab:45<br>speed – *brzina mrežnog sučelja u* ***Mbps***.<br>duplex – (**full** \| **half**)<br>auto-negotiate – (**true** \| **false**)<br>mtu – **MTU** vrijednost mrežnog sučelja (**1-65536**) |
| **[ipv4]** ili **[ipv6]**<br>method | *auto* | Način konfiguracije IP adrese:<br>• auto – *za konfiguraciju pomoću DHCP poslužitelja.*<br>• disabled - *isključeno.*<br>• manual – za *ručnu konfiguraciju IP parametara sučelja*. |
| **[ipv4]** ili **[ipv6]**<br>address-data | *IP adresa* | IP adresa/*prefix*, *GW*. Primjerice: **10.0.1.45./24,10.0.1.1** |
| **[ipv4]** ili **[ipv6]**<br>dhcp-hostname | *Hostname* | DHCP *hostname* (ako se koristi "*dhcp-send-hostname*") prema DHCP poslužitelju. |
| **[ipv4]** ili **[ipv6]**<br>dhcp-send-hostname | *TRUE* | Ako je omogućeno (*TRUE*), šalje se *hostname* na DHCP poslužitelj. |
| **[ipv4]** ili **[ipv6]**<br>dns | *IP* | IP adrese DNS poslužitelja. |
| **[ipv4]** ili **[ipv6]**<br>gateway | *IP* | IP adresa podrazumijevanog usmjerivača (*Default gateway*). |
| **[ethernet]**<br>ethtool.**XYZ** | ***Ethtool opcije*** | Primjerice:<br>ethtool.coalesce-adaptive-rx=(**on** \| **off)**<br>ethtool.coalesce-adaptive-tx=(**on** \| **off**) |

Ako još uvijek koristite mrežne profile u *ifcfg* formatu, razmislite o migraciji u novi format, što možemo napraviti sa:

```
nmcli connection migrate
```



Za više detalja o ***NetworkManager*** servisu, pogledajte poglavlja:
**25.6.6. Rad s mrežnim servisom.**
**25.6.6.1. Konfiguracija mreže upotrebom naredbe nmcli i *NetworkManager* servisa.**

Pogledajmo primjer konfiguracije mrežnog sučelja `enp0s3` u datoteci:
`/etc/NetworkManager/system-connections/enp0s3.nmconnection`:

```
[connection]
id=enp0s3
uuid=82a62722-12f7-4d26-ab7c-0eb22c301229
type=ethernet
autoconnect=true

[ipv4]
method=auto

[ethernet]
mac-address=00:40:21:1f:ab:45
```

Jasno su vidljive sekcije: `[connection]`, `[ipv4]` i `[ethernet]` te definirane opcije (tzv, ***ključevi***) unutar njih.

**Izvori informacija: (1343),(1437),(1438),(1640),(1641), `man NetworkManager`,`man nm-settings`, `man nm-settings-keyfile`,`man NetworkManager.conf`.**

## 26.2. Konfiguracija i rad sa: SSH, SFTP i SCP

Funkcioniranje SSH protokola smo objasnili u prijašnjim poglavljima, a sada ćemo krenuti na detalje. SFTP protokol je praktično implementacija FTP protokola preko SSH kriptiranog (šifriranog) kanala. U paketu sa SSH protokolom dobivamo i nekoliko drugih protokola: ***SCP*** je implementacija **RCP** (*Remote Copy*) naredbe za kopiranje datoteka ili direktorija, ali također preko SSH kriptiranog kanala. Važno je znati da su SSH, SFTP i SCP funkcionalnosti implementirane unutar SSH servisa `sshd`.

### 26.2.1.1. Instalacija i pokretanje

SSH Servis (poslužiteljska strana) je uglavnom već instaliran. Ovako možemo provjeriti je li SSH servis već instaliran:

```
rpm -q openssh-server

openssh-server-5.3p1-112.el6_7.x86_64
```

Opis: pitali smo paketni menadžer *RedHat-a* (`rpm`) da nam provjeri je li paket imena: `openssh-server` već instaliran i ako je, da nam pokaže o kojoj se točno inačici radi. Mi smo u odgovoru dobili da se radi o inačici `5.3p1-112`.

U gornjem slučaju je sve u redu jer je SSH servis instaliran. U slučaju kada SSH servis nije instaliran, to je moguće napraviti na sljedeći način. U ovom primjeru ćemo instalirati i SSH poslužitelj i SSH klijent:

```
yum -y install openssh-server openssh-clients
```

**Za RedHat/CentOS 6.x**
Nakon ove točke, potrebno je sâmo pokrenuti servis:

```
service sshd start
```

Nakon toga omogućimo automatsko pokretanje SSH servisa tijekom svakog pokretanja sustava:

```
chkconfig sshd on
```

**Za RedHat/CentOS 7.x/8.x**
Nakon instalacije potrebno je samo pokrenuti ovaj servis na sljedeći način:

```
systemctl start sshd
```

Potom omogućimo automatsko pokretanje SSH servisa tijekom svakog pokretanja sustava:

```
systemctl enable sshd
```

Sada možemo provjeriti je li se SSH servis pokrenuo na TCP portu 22 pomoću naredbe `netstat` na sljedeći način:

```
netstat -tulpn | grep :22

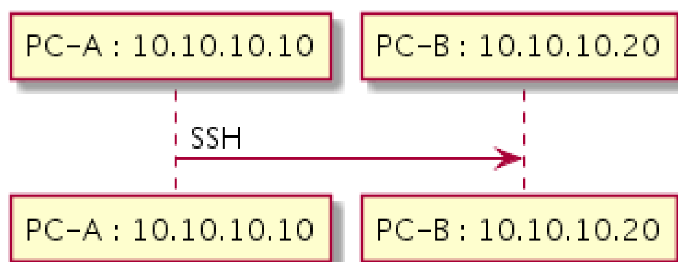
tcp        0      0 0.0.0.0:22          0.0.0.0:*          LISTENp      2030/sshd
tcp        0      0 :::22                    :::*          LISTEN       2030/sshd
```

Vidimo da je `TCP` port `22` u upotrebi na lokalnom računalu (`0.0.0.0`) te da ga koristi servis `sshd`. Ovo znači da je sve u redu. Sada se s drugog računala sa SSH klijentom možemo spojiti na ovo računalo, to jest na SSH poslužitelj.
Korisnički računi s kojima se možemo spajati su svi korisnički računi na ovom računalu (poslužiteljska strana) s kojima se moguće logirati i na sâmo računalo odnosno ovaj poslužitelj. Vrlo često je slučaj kada je direktno logiranje na SSH poslužitelj kao korisnik `root` onemogućeno zbog sigurnosnih razloga, što je moguće naknadno promijeniti.

### 26.2.1.2. Spajanje na udaljeni SSH servis

Zamislimo da se sa računala **`PC-A`** spajamo preko SSH klijenta, na SSH poslužitelj na računalu **`PC-B`** kao na slici **244.**

**Slika 244. Spajanje na udaljeno računalo upotrebom SSH protokola (i SSH klijenta).**




Dođimo do računala **`PC-A`**, IP adrese: 10.10.10.10 i pokrenimo *ssh* klijent, da se spoji na računalo **`PC-B`** s IP adresom:10.10.10.20 korištenjem korisničkog računa odnosno korisnika naziva `root`.

Dakle mi se nalazimo na Linux računalu **`PC-A`** i spajamo se na udaljeno Linux računalo **`PC-B`**, sa `ssh`, kao korisnik `root`:

```
[root@PC-A ~]# ssh 10.10.10.20 -l root
root@10.10.10.20's password:
[root@PC-B ~]#
```

Isto spajanje smo mogli ostvariti i s *Windows* računala primjerice upotrebom programa ***PuTTY***.
Nakon što budemo unijeli lozinku za računalo **`PC-B`** kako je vidljivo u koraku: `root@10.10.10.20's password:` spajamo se na njega i dobivamo njegovu ljusku (*Shell*) kao da smo na njega spojeni lokalno, odnosno kao da sjedimo ispred njega.

### 26.2.1.3. Konfiguracija SSH klijenta

Konfiguracija SSH klijenta na našem Linux računalu je naravno moguća, a nju možemo odraditi na nekoliko razina.
U primjerima koji slijede radit ćemo konfiguraciju SSH klijenta. Naime kada se mi, kao *SSH* klijenti spajamo na neko drugo računalo na mreži, prvo se učitava konfiguracija SSH klijenta, sa sljedećih datoteka, redom (ako postoje):

1. Globalna konfiguracijska datoteka koja vrijedi za sve korisnike sustava: `/etc/ssh/ssh_config`.
2. Specifična za svakog pojedinog korisnika: (unutar njegovog kućnog direktorija): `~/.ssh/config`.
3. I na kraju svega onoga što s naredbom `ssh` pokrećemo iz ljuske (*shella*).

Pri tome globalna konfiguracija (**1**) ima najveću težinu odnosno važnost te će pregaziti onu za korisnike (**2**) kao i ono što potencijalno upisujemo kada se ručno spajamo (**2**) na udaljeni *SSH* servis. Mi ćemo biti logirani kao korisnik `hrvoje` te se stoga nalazimo u njegovom početnom (*kućnom*) direktoriju odnosno mapi: `/home/hrvoje/`.
**A.)** Stoga prvo moramo kreirati poseban skriveni direktorij imena `.ssh` te mu definirati (smanjiti) ovlasti. Potom uđimo u njega, promijenimo mu ovlasti i kreirajmo nove *SSH* ključeve (ako ih nemamo), s naredbom `ssh-keygen`

```
mkdir /home/hrvoje/.ssh
chmod 0700 /home/hrvoje/.ssh
cd /.ssh
ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/hrvoje/.ssh/id_rsa):
Created directory '/home/hrvoje/.ssh'.
Enter passphrase (empty for no passphrase):
```

**Na pitanja tijekom kreiranja ključeva (`ssh-keygen`) sâmo stisnite *ENTER* i ključevi će biti snimljeni na svoje mjesto.**

**B.)** Sada možemo kreirati konfiguracijsku datoteku `config` za *SSH* klijenta `hrvoje`:

```
touch config
```

Unutar ove datoteke možemo definirati razne varijable i parametre rada *SSH* klijenta (pr. `Cipher, Ciphers, Compression, ConnectionAttempts, ConnectTimeout,TCPKeepAlive` …). Osim toga možemo i definirati ponašanje te način spajanja *SSH* klijenta za točno definirana računala na mreži; prema principu definiranja sekcije za računala (`Host`) i njegove pôd sekcije s parametrima za to računalo; poput:

```
Host [Alias]
        Opcija1 [vrijednost]
```

Za ovu konfiguracijsku datoteku `config` postoji stotinjak opcija i parametara. Stoga ćemo navesti samo nekoliko njih:

- `Host` **`XY`** - označava sekciju (**`XY`**) na koju se dalji blok konfiguracije odnosi.
- `Hostname` - označava ime udaljenog računala (ime računala ili IP adresa).
- `Port` - označava udaljeni port odnosno port udaljenog računala na koje se spajamo.
- `IdentityFile` - označava datoteku u kojoj se nalaze SSH ključevi koji će se koristiti za autentikaciju. Standardno je privatni (tajni) *SSH* ključ za svakog korisnika pohranjen u datoteku: `~/.ssh/id_rsa` dok je javni ključ pohranjen u datoteku: `~/.ssh/id_rsa.pub`.
- `SetEnv` i `SendEnv` se koriste za podešavanje varijabli na udaljenom računalu; sâmo u slučaju kada druga strana ima to omogućeno, odnosno mora imati postavljeno: `AcceptEnv=yes` u datoteci: `/etc/ssh/sshd_config`.
- `ServerAliveInterval` i `ServerAliveCountMax` - pomoću njih će *SSH* klijent, ako nije primio nikakve podatke u određenom vremenskom intervalu, prvo poslati prazan upit te zatražit odgovor od druge strane (poslužitelja). To može spriječiti ***Load Balancere*** kao i poslužitelje da prekinu vezu zbog neaktivnosti.

Otvorimo datoteku `/home/hrvoje/.ssh/config` (otvorimo ju s programom s kojim želite; primjerice s `vi`).
Zamislimo sljedeći scenarij: imamo dva poslužitelja: `server1` (`192.168.1.231`) te `server2` (`192.168.1.36`).
Poslužitelj `server1` ima SSH servis pokrenut na portu 22 (standardno), međutim `server2` ima *SSH* servis pokrenut na portu **22222**. Stoga, kada se sa poslužitelja `server1` želimo spojiti na poslužitelj `server2`, to moramo napraviti ovako:

```
ssh 192.168.1.36 -p 22222 -l hrvoje
```

Na oba poslužitelj imamo korisnika `hrvoje`, a želimo se na jednostavniji način moći spajati s jednog poslužitelja na drugi, preko *SSH* protokola, kao navedeni korisnik na definirani udaljeni port. Nadalje oba poslužitelja su dodana u `/etc/hosts` datoteku pa se njihovo ime može razlučiti u IP adresu. Stoga ćemo za početak na prvom poslužitelju `server1` odraditi korake **A.)** i **B.)**.
Potom ćemo u datoteku: `/home/hrvoje/.ssh/config` dodati sljedeće retke konfiguracije:

```
Host server2
    HostName 192.168.1.36
    User hrvoje
    Port 22222
```

Sada se s prvog poslužitelja, na drugi, možemo spojiti sâmo ovako:

```
ssh server2
```

Nakon toga će nas udaljeni sustav naravno zatražiti lozinku.
Dakle tijekom spajanja, naša konfiguracijska datoteka (`/home/hrvoje/.ssh/config`) će se pobrinuti da nadoda sve što smo u njoj definirali za spajanje na poslužitelj: `server2`. Moguće je definirati više unosa; za svaki poslužitelj zasebno.
Moguće je definirati i znatno kompleksnije stvari (pr. *tuneliranje* koje ćemo objasniti kasnije i sl.), ali to prepuštamo vama.

**Pogledajmo kako razmijeniti SSH ključeve između oba poslužitelja, kako više ne bi morali upisivati lozinku.**
*Prvo moramo biti logirani kao korisnik `hrvoje` jer za tog korisnika omogućujemo logiranje bez lozinke, samo sa ključevima.*
S poslužitelja `server1`* pokrenite sljedeću naredbu, te potvrdite spajanje sa (`yes`), a potom i sa lozinkom:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.1.36
```

U ovom koraku smo zapravo svoj(x) javni ključ (datoteku: `/home/hrvoje/.ssh/id_rsa.pub`) odnosno njen sadržaj (sâm ključ) prekopirali na udaljeni poslužitelj (`192.168.1.36`) u posebnu datoteku u koju se upisuju autorizirani korisnici.
Ime te datoteke je: `authorized_keys` odnosno za ovog korisnika je to datoteka: `/home/hrvoje/.ssh/authorized_keys`. Iz ove datoteke za svakog korisnika, provjerava se tko ima prava spojiti se na sustav bez lozinke, sâmo s javnim ključem, koji se i zapisuje u ovu datoteku prema principu; jedan redak (linija) – jedan unos (udaljeni poslužitelj).
Zatim s poslužitelja `server2`(y) pokrenite sljedeću naredbu, te potvrdite spajanje sa (`yes`), a potom i sa lozinkom:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.1.231
```

U ovom koraku smo svoj (y) javni ključ (datoteku: `/home/hrvoje/.ssh/id_rsa.pub`) odnosno njen sadržaj (sâm ključ) prekopirali na udaljeni poslužitelj (`192.168.1.231`) u posebnu datoteku u koju se upisuju njemu autorizirani korisnici.
Na ovaj način oba poslužitelja imaju javne ključeve susjednih poslužitelja pa je stoga omogućeno SSH spajanje bez lozinke.
Pogledajmo kako izgleda jedan javni ključ (datoteka: `/home/hrvoje/.ssh/id_rsa.pub`) s poslužitelja `server1`:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC+pY8uCoQo3FYtyVPe8unNyfi+dXNvt0+BkGXBttzV8jARvIFKIDykC5VKDNj/iHL
3eYsusmExd/DypTpymI6//UOPynCIboVBfoGQzszCclzWcLkbX/LoNGkdKJc6A8vg1lVWxv670UEoX41CQtMnfeo3msVc52
gLRFxD047f1mgHyi18OemM9Tdf0tjwkKmb+KwYBsbFZ0XZ2V4tiQ1r7JqHA8ElTJZtdT+78lQcXq5txhQ5It+VVswNEnQvE
Wpvn7heQzvLoD3FxuBsil45JzueKAEdOhhnmu0xqV0e1pSvJLwNHBTDA1RSmqyQnQr7eUKh4FIfADul9eHsTBP7
hrvoje@server1
```

**Izvori informacija: (921)**, **`man ssh_config`**, **`man ssh-keygen`**.

---

## 26.2.1.4. Napredna konfiguracija SSH servisa

Konfiguracijska datoteka sâmog SSH servisa se nalazi na lokaciji odnosno u datoteci: `/etc/ssh/sshd_config`.
U svim primjerima dolje pretpostavlja se da ćete urediti gore navedenu datoteku, snimiti ju te restartati *ssh* servis s naredbom:

```
service sshd restart
```

**Kako omogućiti ssh pristup `root` korisniku?**
Unutar konfiguracijske datoteke `/etc/ssh/sshd_config` potražimo parametar `PermitRootLogin` koji mora izgledati ovako: `PermitRootLogin yes`
Ovo je ujedno i potencijalni sigurnosni problem te se stoga preporučuje postaviti lozinku za `root` korisnika na vrlo kompleksnu dakle da sadrži velika i mala slova, brojeve i posebne znakove.

**Kako promijeniti standardni TCP port 22 u neki drugi; recimo na TCP port 11111.**
Unutar konfiguracijske datoteke `/etc/ssh/sshd_config` potražimo parametar `Port 22` koji moramo promijeniti u:

```
Port 11111
```

Ako koristite ***SELinux*** sustav, prilagodite ga za ovaj novi port. Pogledajte cjelinu: **28.2.SELinux sustav**.

**Povećajmo razinu sigurnosti**
Smanjimo vrijeme koje imamo za logiranje na 1 minutu. Također smanjimo broj pogrešnih pokušaja za logiranje na pet, nakon kojih više neće biti moguće ponovno logiranje s istim korisničkim računom na neko vrijeme. Ova metoda rješava i problem "*Brute Force*" napada na pojedini korisnički račun preko SSH protokola. Ograničimo i broj aktivnih istovremenih SSH veza (konekcija) na *ssh* poslužitelj na 10 komada, dodavanjem sljedećih opcija u konfiguracijsku datoteku:

```
LoginGraceTime 1m
MaxAuthTries 5
MaxSessions 10
```

Poboljšajmo i sigurnost odabirom sigurnijih algoritama za kriptiranje (ključna riječ `Ciphers`) te algoritama za sigurnu razmjenu ključeva (ključna riječ `KexAlgorithms`) kao i algoritama za autentikaciju poruka: *Message Authentication Code* (***MAC***) koji se definiraju s ključnom riječi `MACs`. Mi ćemo stoga dodati sljedeća tri reda s kojima definiramo kojim algoritmima se dozvoljava spajanje klijenata na naš SSH poslužitelj (***GCM*** se preporuča, ako je dostupan):

```
Ciphers aes256-gcm,aes128-cbc,aes192-cbc,aes256-cbc,aes256-ctr
KexAlgorithms ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-
hellman-group14-sha1,diffie-hellman-group-exchange-sha256
MACs hmac-sha2-256,hmac-sha2-512,hmac-sha1
```

I ne zaboravimo na restart *ssh* servisa nakon svih ovih promjena:

```
service sshd restart
```

---

## 26.2.1.5. SCP

***SCP*** (Engl. *Secure Copy*) naredba se koristi za kopiranje datoteka ili direktorija s računala na računalo na siguran (kriptiran) način. U istoj konfiguraciji kao na slici prije prekopirajmo datoteku `vazno.txt` s računala **`PC-A`** na **`PC-B`**. Gore navedena datoteka se nalazi unutar našeg direktorija `/root/podaci/` i na istu lokaciju ćemo je prekopirati na drugom računalu.
Ovo radimo kao korisnik `root`. Stoga pokrenimo naredbu `scp` na sljedeći način:

```
[root@PC-A ~]# scp /root/podaci/vazno.txt root@10.10.10.20:/root/podaci/
root@10.10.10.20 password:
vazno.txt                    100%   47KB  47.5KB/s   00:00
```

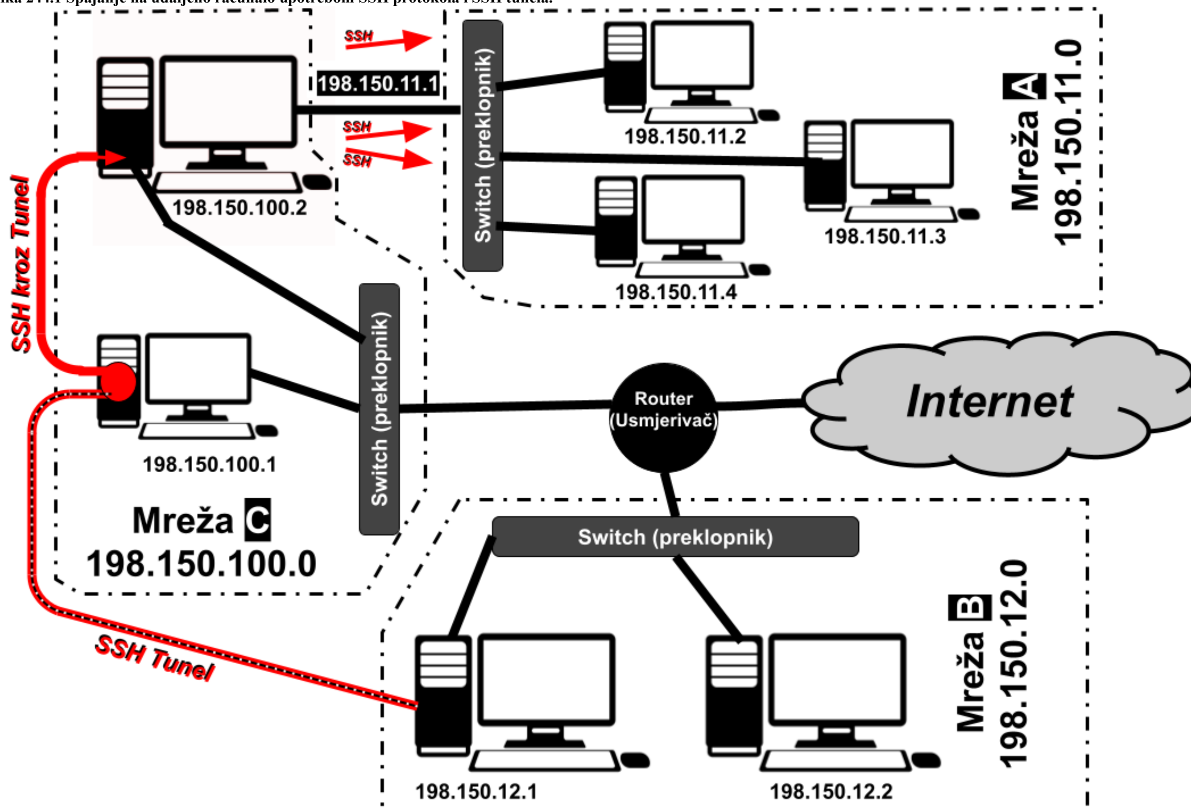
Nakon što upišemo lozinku za `root` korisnički račun na računalu na koje se spajamo tj. računalu **`PC-B`**, datoteka se počinje kopirati, a vidljivi su i napredak kopiranja te statistika o brzini kopiranja.

---

## 26.2.1.6. SSH tuneliranje

Pomoću SSH protokola moguće je raditi i takozvano tuneliranje. Naime moguće je pomoću SSH protokola spojiti se na udaljeno računalo ili poslužitelj te tu istu vezu iskoristiti na poseban način kao svojevrstan tunel za dalje spajanje na računala iza njega. Ova metoda se također može koristiti za izbjegavanje vatrozida, kada promet možete preusmjeriti kroz otvorenu SSH vezu na ona računala (poslužitelje i druge uređaje) kojima nemate pristup s vanjske mreže. Ovaj postupak stvara svojevrsni ***VPN*** (Engl. *Virtual Private Network*) tunel, ali bez posebne konfiguracije *VPN-a*. Dovoljan je sâmo odgovarajuće konfiguriran SSH klijent i SSH poslužitelj koji to dozvoljava. Logički način rada ovakvog SSH tunela pogledajte na slici **244.1**.

**Slika 244.1 Spajanje na udaljeno računalo upotrebom SSH protokola i SSH tunela.**




U primjeru na slici **244.1** vidimo da smo se s računala **198.150.12.1** sa SSH klijentom spojili na udaljeni SSH poslužitelj na IP adresi **198.150.100.1**, preko našeg usmjerivača. Prema tom udaljenom poslužitelju (**198.150.100.1**) smo otvorili poseban SSH tunel, kroz koji se pomoću SSH protokola dalje možemo spajati na sva druga računala, koja bi nam inače bila nedostupna jer se nalaze u izoliranoj mreži (mreža A), poput računala: **198.150.11.2**, **198.150.11.3** ili **198.150.11.4**. Osim ovakvog jednostrukog SSH tunela moguće su i napredne konfiguracije višestrukih tunela: s prvog poslužitelja na drugi, s drugog na treći i tako dalje, a s krajnjeg se onda možemo spajati na odredišna računala. U svakom slučaju SSH tuneliranje se naziva i SSH prosljeđivanje portova odnosno engleski *port forwarding*. To je stoga što SSH tuneliranjem stvaramo šifriranu SSH vezu između klijenta i poslužiteljskog stroja tako da taj komunikacijski kanal koristimo upotrebom željenog TCP porta na koji se potom spajamo.

Kod upotrebe SSH protokola, razlikujemo tri vrste odnosno načina prosljeđivanja porta (engl. *port forwarding*).

- **Lokalno prosljeđivanje porta** omogućuje nam prosljeđivanje porta na lokalnom (*ssh* klijentskom) stroju, na port na udaljenom (***ssh*** poslužiteljskom) stroju, koji se zatim prosljeđuje na port na odredišnom stroju. U ovoj vrsti prosljeđivanja, SSH klijent sluša na danom portu i tunelira bilo koju vezu s tim portom prema definiranom portu na udaljenom SSH poslužitelju, koji se zatim povezuje s portom na odredišnom stroju. Odredišni stroj može biti udaljeni SSH poslužitelj ili bilo koji drugi stroj (računalo/uređaj).
- **Udaljeno prosljeđivanje porta** je suprotno od lokalnog prosljeđivanja porta. Omogućuje nam prosljeđivanje porta s udaljenog (*ssh* poslužiteljskog) stroja na port na lokalnom (*ssh* klijentskom) stroju, koji se zatim prosljeđuje na port na odredišnom računalu. U ovoj vrsti prosljeđivanja, SSH poslužitelj sluša na danom portu i *tunelira* bilo kakvu vezu s tim portom do navedenog porta na lokalnom SSH klijentu, koji se zatim povezuje s portom na odredišnom stroju. Odredišni stroj može biti lokalni ili bilo koji drugi stroj (računalo/uređaj) u lokalnoj mreži. Ovakav način rada se naziva i ***reverzni ssh*** pristup.
- **Dinamičko prosljeđivanje porta** omogućuje nam stvaranje porta na lokalnom (*ssh* klijentskom) stroju, koji djeluje kao ***SOCKS proxy*** poslužitelj. Kad se klijent (pr. web preglednik) poveže na ovaj (*SOCKS 4/5 PROXY*) port, veza se prosljeđuje na udaljeni (*ssh* poslužitelj) stroj, koji se zatim prosljeđuje na dinamički port na odredišnom računalu. Na taj će se način sve aplikacije koje koriste ***SOCKS proxy*** povezati na SSH poslužitelj, a poslužitelj će sav promet proslijediti na svoje stvarno odredište.

## Primjeri konfiguracije i upotrebe SSH tuneliranja.

**Lokalno prosljeđivanje porta (*port forwarding*)**
U ovom primjeru naše Linux računalo (IP: **198.150.12.1**) otvara *SSH* tunel na udaljeni Linux poslužitelj (IP: **198.150.100.1**) s lokalnim prosljeđivanjem porta na TCP port broj **3333**. Dakle otvaramo lokalni TCP port **3333**. Ovo radimo s našeg računala:

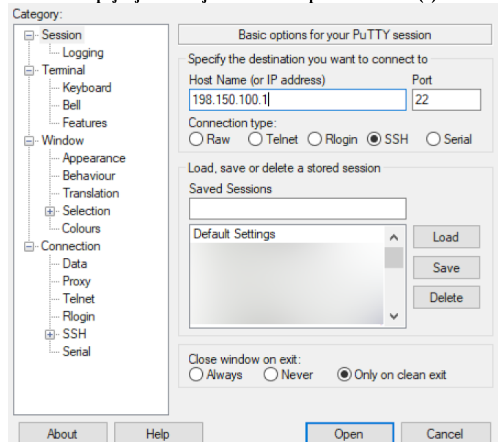
```
ssh -f -L 198.150.12.1:3333:198.150.100.1:22 root@198.150.100.1 -N
```

Nakon toga sustav će nas zatražiti lozinku za spajanje na udaljeni SSH poslužitelj (IP: **198.150.100.1**) za korisnika `root`.

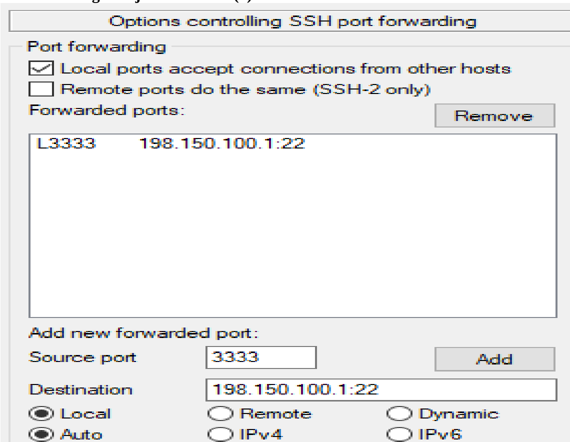
Kada se ovakva SSH veza ostvari, na našem računalu (IP: **198.150.12.1**) će se na našoj IP adresi otvoriti TCP port **3333**.
Sada se s bilo kojeg računala na našoj lokalnoj mreži (uključujući i naše) možemo spojiti na naše računalo na IP adresi **198.150.12.1** na TCP port **3333**, a ta veza (koja je *SSH* tunel) će nas provući kroz *SSH* tunel na odredišni poslužitelj na IP adresi **198.150.100.1**, na TCP port 22 (*SSH*) koji je port tog poslužitelja za *SSH* servis. Stoga ćemo se na njega preko *SSH* moći i logirati.

Moguće je koristiti i malo drugačije opcije da se postigne isto.
Pri navedenoj konfiguraciji `-L` označava da ćemo koristiti lokalno prosljeđivanje porta, te `-f` govori *SSH* klijentu da se pokrene u pozadini (kao servis), a `-N` kako ne želimo izvršavati nikakve *SSH* naredbe na udaljenom *SSH* poslužitelju (jer ovo ionako treba biti sâmo otvoreni *SSH* tunel).
Pogledajmo kako bi izgledala ista konfiguracija *SSH* klijenta u *Windowsima*, ako koristimo **PuTTY**/**KiTTY** *SSH* klijente.

**Slika 244.2 Spajanje na udaljeno računalo upotrebom SSH (1).**




**244.3. Konfiguracija SSH tunela (2).**




Ovdje inicijalno konfiguriramo spajanje na udaljeni *SSH* poslužitelj (slika 244.2) s IP adresom: **198.150.100.1**, na TCP port **22**. Zatim (slika 244.3) pod: ***Connection* → *SSH* → *Tunnels*** odabiremo „*Local port accept connections from other hsosts*“ što znači da dozvoljavamo konekcije s drugih računala. U donjem dijelu konfiguracije (slika 244.3) definiramo da će se na našem ***Windows*** računalu otvoriti port **3333** (definiran pod „***Source port***“). Potom moramo definirati na koji udaljeni *SSH* poslužitelj se spajamo s ovim tunelom [opcija ***Destination***]. Mi smo konkretno odabrali isti poslužitelj (**198.150.100.1**, na TCP port **22**) te kao odredište biramo „***Local***“ s čime definiramo da se radi o lokalnom prosljeđivanju porta. Na kraju odabiremo „***Add***“.


Međutim ovdje smo kao odredišni poslužitelj [opcija ***Destination***] odmah mogli navesti prvi sljedeći na koji želimo da se spoji ovaj *SSH* poslužitelj**(198.150.100.1)**; primjerice krajnji *SSH* poslužitelj s IP adresom: **198.150.11.2**.

**Pogledajte sliku 244.1.**

Zatim otvaramo ovu vezu [slika 244.2 ***Open***]. Nakon što smo se logirali na ovaj udaljeni *SSH* poslužitelj, na našem ***Windows*** računalu se otvara lokalni TCP port **3333**, na koji kada se spojimo s novim *SSH* klijentom, proći ćemo kroz *SSH* tunel sve do odredišnog *SSH* poslužitelja s IP adresom: **198.150.100.1**. Opcija koja *SSH* poslužitelju omogućava upotrebu lokalnog prosljeđivanja porta, definirana je u datoteci: `/etc/ssh/sshd_config`, kao opcija koja mora izgledati ovako: `AllowTcpForwarding yes`.

Ovakvo *tuneliranje* možemo koristiti i primjerice kod upotrebe ***SCTP*** protokola za pristup datotekama na udaljenom poslužitelju, recimo kod upotrebe programa ***WinSCP***. U slučaju kada želimo da nam ***WinSCP*** u pozadini kreira SSH tunel kroz IP adresu: **198.150.100.1** na odredišno računalo s IP adresom: **198.150.11.2** (s kojeg dohvaćamo datoteke), napravili bi sljedeće. Pod „***Host name***“ bi odabrali krajnje računalo (**198.150.11.2**, port **22**), a onda bi u opcijama ***Advanced* → *Connection* → *Tunnel***, uključili *tuneliranje* sa „***Connect through SSH tunnel***“. Tek ovdje bi pod „***Host name***“ unijeli podatke prvog SSH poslužitelja u nîzu (IP: **198.150.100.1**, *port* **22**) na koji se prvo spajamo, odnosno preko kojeg dolazimo do drugog poslužitelja s kojega dohvaćamo željene datoteke.

Dakle imamo *SSH* tunel: **naše računalo → 198.150.100.1 → 198.150.11.2.**

**Reverzni SSH**
Pogledajmo i primjer upotrebe udaljenog prosljeđivanja porta odnosno takozvanog ***reverznog ssh*** pristupa. To znači da radimo obrnuto spajanje u odnosu na prethodne primjere. Naime, ako se ne možemo spojiti na udaljeni *SSH* poslužitelj jer se nalazi iza vatrozida, ali se on može spojiti na nas, upravo to ćemo i napraviti na sljedeći način. Ovo radimo na udaljenom poslužitelju:

```
ssh root@198.150.12.1 -R 50000:localhost:22 -N -f
```

Nakon što unesemo lozinku za **root** korisnika na našem računalu (`198.150.12.1`) otvorit će se *SSH* tunel s udaljenog poslužitelja na naše računalu, tako da će se na našem računalu otvoriti lokalni port **50000.** Pri tome `localhost` označava *loopback* IP adresu 127.0.0.1, umjesto koje smo mogli koristiti i IP adresu naše mrežne kartice. Sada se s našeg računala možemo spojiti u obrnutom smjeru. Odnosno možemo se spojiti na SSH, na lokalni port **50000**, s čime se zapravo kroz *SSH* tunel spajamo na udaljeni *SSH* poslužitelj, na njegov port **22** [SSH protokol]. Ovo radimo na našem računalu:

```
ssh root@localhost -p 50000
```

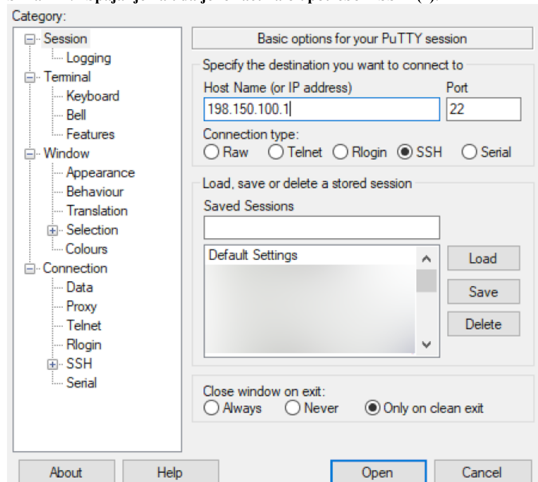
Opcija koja na *SSH* strani na koju se inicijalno spaja, omogućava korištenje lokalnog porta za vanjska računala (na mreži), definirana je u datoteci: `/etc/ssh/sshd_config`, kao opcija koja treba izgledati ovako: `GatewayPorts yes`.

**Dinamičko prosljeđivanje porta i SOCKS proxy**
U određenim slučajevima možemo imati potrebu spajati se na web sučelje udaljenih uređaja, do kojih je moguće doći jedino preko nekog drugog poslužitelja. Ako se vratimo na primjer sa slike 244.1 mi ćemo se se s računala **198.150.12.1** sa SSH klijentom spojiti na udaljeni SSH poslužitelj na IP adresi **198.150.100.1**. Prema tom udaljenom poslužitelju (**198.150.100.1**) ćemo otvorili poseban SSH tunel, kroz koji se pomoću **SOCKS* 5** protokola dalje možemo spajati na web sučelja (ili web stranice) svih drugih računala ili uređaja koji su dostupni samo tom poslužitelju. To znači da naše računalo tada postaje svojevrsni ***SOCKS 5 Proxy*** poslužitelj. Mi ćemo u ovom primjeru to ostvariti u dva koraka:

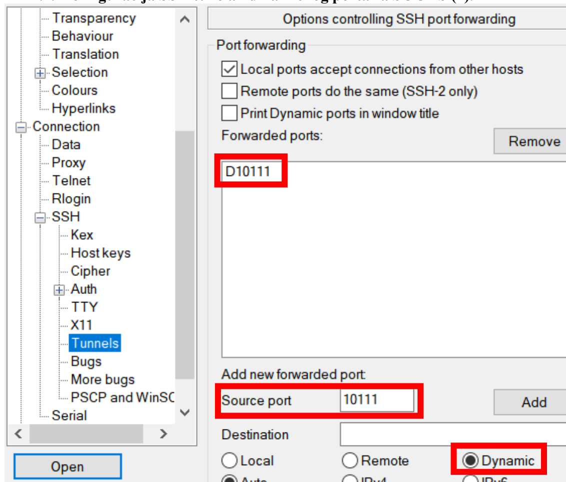
1. Otvaranjem SSH veze prema poslužitelju na IP adresi: **198.150.100.1**.
2. Povezivanjem prethodno otvorene SSH veze, kao SSH tunela, na naš lokalni port, primjerice port **10111**, na naše sučelje povratne veze (*loopback interface*): **127.0.0.1**, odabirom dinamičkog *port forwardinga*.

Odnosno cijela konfiguracija u programu **PuTTY** će izgledati ovako:

**Slika 244.2 Spajanje na udaljeno računalo upotrebom SSH (1).**




**244.4. Konfiguracija SSH tunela i dinamičkog porta za SOCKS (2).**




Odnosno, ako to isto želimo postići sa SSH klijentom u Linuxu, to možemo postići i ovako (`-D` je za dinamički port):

```
ssh -D 10111 198.150.100.1
```

← Dinamički *port forwarding* (`-D`) podržava **SOCKS 4** i **SOCKS 5** (**RFC 1928**) mrežne aplikacijske protokole.

I sada u našem web pregledniku, iz kojeg želimo pomoću **SOCKS 5** protokola koristiti SSH tunel, kako bismo došli do odredišnih web poslužitelja, trebamo učiniti sljedeće. Ako je Web preglednik ***Mozilla Firefox*** tada iz menija odaberimo **Settings** i u pretragu pojma upišimo: **Proxy**. Pojavit će se **Network Settings** pa ponovno odaberimo **Settings** s desne strane.
Sada će biti vidljivo: ***Configure Proxy Access to the Internet***.
Ovdje odabiremo: **Manual proxy configuration**.
Zatim pod **SOCKS host** i port upišite IP: **127.0.0.1** te **10111** kao port.

SOCKS Host 127.0.0.1 Port 10111

SOCKS v4 SOCKS v5

Na kraju sve i potvrdite. Nakon potvrde naš web preglednik će koristiti **SOCKS 5** protokol da bi kroz naše računalo (**127.0.0.1**) na portu **10111** mogao doći do web stranica kroz SSH tunel koji smo prethodno otvorili.



Za više detalja o **Proxy** poslužiteljima (i **SOCKS**) pogledajte poglavlje:
**26.4.5. Proxy poslužitelji.**

**Izvori informacija: (963),(964),(965),(966),(1588), RFC 1928, `man ssh`.**

### 26.2.1.7. SSH debugging

Kada se SSH klijent poveže s poslužiteljem, svaka strana nudi popise parametara veze drugoj strani. Popisi ovih parametara, s odgovarajućom ključnom riječi su zapravo varijable za *ssh* klijenta (*konf.* `ssh_config`) ili poslužitelja (`sshd_config`):

- `KexAlgorithms` - metode razmjene ključeva koje se koriste za generiranje ključeva prilikom povezivanja.
- `HostkeyAlgorithms` - algoritmi javnog ključa prihvaćeni za SSH poslužitelj kako bi se autentificirao SSH klijentu unutar komunikacijske sesije/kanala.
- `Ciphers` - algoritmi korišteni za kriptiranje (šifriranje) veze.
- `MACs` - kodovi za provjeru autentičnosti poruka koji se koriste za otkrivanje eventualnih izmjena prometa.

Za uspješno ostvarivanje veze, mora postojati barem jedan međusobno podržan izbor unutar skupa parametara veze za svaki navedeni parametar odnosno korak tijekom uspostave veze. Ako se klijent i poslužitelj ne mogu dogovoriti oko zajedničkog skupa parametara, veza se neće ostvariti. Primjerice *OpenSSH* (v.7.0 i noviji) kreirat će poruku o pogrešci poput ove:

```
Unable to negotiate with legacyhost: no matching key exchange method found.
Their offer: diffie-hellman-group1-sha1
```

U ovom slučaju klijent i poslužitelj nisu se mogli dogovoriti oko algoritma za razmjenu ključeva. Ovdje je poslužitelj ponudio samo jednu metodu `diffie-hellman-group1-sha1`. *SSH* klijent podržava ovu metodu, ali je ne uključuje prema zadanim postavkama jer je slaba (zastarjela i nesigurna) i unutar teorijske domene takozvanog napada ***Logjam***.
Stoga SSH klijent nije htio prihvatiti ovu metodu odnosno algoritam za razmjenu ključeva i odbacio je vezu.
Dakle ovdje je rješenje sa strane poslužitelja ukloniti ovu metodu (korištenjem varijable `KexAlgorithm`) unutar datoteke: `sshd_config`. Primjerice dodavanjem sljedećeg retka (a potom restart SSHD servisa [*sshd*]):

```
KexAlgorithms diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-
hellman-group14-sha256,diffie-hellman-group14-sha1
```

Sa strane klijenta provjeru svih odobrenih (aktivnih) algoritama za razmjenu ključeva možemo vidjeti s naredbom:

```
ssh -Q kex
```

Nekoliko povezanih opcija ulazi u igru kasnije, u koraku tijekom autentifikacije korisnika:

- `PubkeyAcceptedKeyTypes` (*ssh/sshd*): algoritmi javnog ključa koje će klijent ponuditi i pokušati koristiti, a poslužitelj prihvatiti za provjeru autentičnosti javnim ključem (npr. putem datoteke: `.ssh/authorized_keys`).
- `HostbasedKeyTypes` (*ssh*) i `HostbasedAcceptedKeyTypes` (*sshd*): tipovi ključeva koje će klijent pokušati primijeniti, a poslužitelj prihvatiti za autentifikaciju temeljenu na računalu (npr. putem datoteka: `.rhosts` ili `.shosts`)



***Neusklađenost između klijenta i poslužitelja tijekom provjere autentičnosti uzrokovat će neuspjeh provjere autentičnosti, unatoč tome što se čini da je ona uredno konfigurirana!***

Na primjer, definicija kriptografskog algoritma `ssh-dss` (odnosi se na zastarjeli **DSA**) može biti navedena u datoteci: `.ssh/authorized_keys`, ali možda neće proći provjeru autentičnosti jer, prema zadanim postavkama, *sshd* (servis/poslužitelj) ne prihvaća ovu vrstu ključa.



Algoritam digitalnog potpisa (**DSA**) objavljen je u Standardu digitalnog potpisa **DSS** (*Digital Signature Standard*) od strane NIST-a u **FIPS 186** standardu. Stoga je on u SSH označen kao `ssh-dss`. *OpenSSH* v.7.0 i novije inačice onemogućuju algoritam javnog ključa `ssh-dss` (**DSA**), pošto je on nesiguran te se ne preporučuje se njegova upotreba.

Za provjeru *ssh* klijenta i algoritama koje on podržava, imamo nekoliko mogućnosti (uz naš komentar **#** u istom retku):

```
ssh -Q cipher       # Lista podržanih algoritama za šifriranje (kriptiranje) veze.
ssh -Q mac          # Lista podržanih MAC algoritama.
ssh -Q key          # Lista podržanih vrsta javnog ključa.
ssh -Q kex          # Lista podržanih algoritama za razmjenu ključeva.
```

Također je moguće provjeriti koju konfiguraciju *ssh* zapravo koristi kada se pokušava spojiti na određeni udaljeni *ssh* poslužitelj, koristeći opciju `-G`:

```
ssh -G user@server
```

Ova naredba će ispisati sve opcije konfiguracije, uključujući odabrane vrijednosti za parametre: `Ciphers`, `MACs`, `HostKeyAlgorithms` i `KexAlgorithms`.

Za dodatne detalje prilikom otvaranja veze na udaljeni *ssh* poslužitelj možemo koristiti prekidač `-vvv`, primjerice ovako:

```
ssh -vvv user@server
```

Nakon toga biti će nam ispisani svi koraci i detalji unutar svakog od koraka tijekom (pokušaja) uspostave veze.

**Izvori informacija: (921),(1527),(1528), `man ssh`,`man ssh_config`,`man sshd_config`, FIPS 186-4, FIPS 140-3**

# 26.3. TFTP

*Trivial File Transfer Protocol* (***TFTP***) je vrlo jednostavan protokol za razmjenu datoteka preko mreže. On za transport koristi ***UDP*** protokol na portu **69** te ga i nadograđuje na poseban način. *TFTP* je definiran u standardu RFC783 te proširen s opcijama u RFC2347. Sâmo *TFTP* zaglavlje je veličine 4 bajta.

## 26.3.1. Instalacija

***TFTP*** je samostalan servis odnosno *daemon*, ali se za pokretanje oslanja na tzv. "*Superserver Daemon*" koji je zadužen za sâmo pokretanje i sigurnost *TFTP*-a, kao i nekih drugih manjih Linux servisa. Ovaj *Superserver Daemon* se zove `xinetd`.
Naime `xinetd` sluša dolazne mrežne zahtjeve za pristup mrežnim servisima i na osnovu protokola koji se zahtjeva pokreće konkretan servis odnosno *daemon*. Dakle osnovni identifikator koji koristi `xinetd` je broj porta, i to zapravo odredišni (*destination*) port koji označava konkretan protokol na osnovu kojega zaključuje što pokrenuti. *Xinetd* ima ugrađene mehanizme za kontrolu pristupa (***ACL***), napredne mogućnosti logiranja te mogućnost aktiviranja željenog servisa u određeno vrijeme. Nadalje on ima mogućnost ograničavanja broja pokrenutih poslužiteljskih servisa te ugrađene mehanizme protiv skeniranja otvorenih portova. Sve gore navedeno su osnovni razlozi zbog kojih se ***TFTP*** ne pokreće samostalno već koristi `xinetd` servis iznad sebe. Naime ***TFTP*** ne posjeduje niti jedan od navedenih zaštitnih mehanizama, te je simbioza s `xinetd` najbolja moguća.
*TFTP* servis opcionalno možemo dobiti i instalacijom servisa: `dnsmasq`, ali to sada nećemo objašnjavati.

**Provjerimo imamo li instaliran `tftp-server` odnosno TFTP poslužiteljsku komponentu.**
S već poznatim paketnim menadžerom *RedHat-a* ćemo vidjeti je li softverski paket imena `tftp-server` instaliran:

```
rpm -q tftp-server

package tftp-server is not installed
```

O ovom slučaju on nije instaliran te ćemo ga instalirati. Kao što smo rekli, on ovisi o `xinetd` pa, ako niti on nije instaliran instalirati ćemo ih automatski. Sjetimo se kako `yum` zna provjeriti ovisi li jedan paket o nekom drugom te će ih sve instalirati pravim redoslijedom (ispis smo skratili):

```
yum -y install tftp-server
```

Sada je sve potrebno i instalirano. Ipak kako bi sve radilo potrebno je konfigurirati i `xinetd` i `tftp-server` da se podižu zajedno s cijelim operativnim sustavom što znači *Runleveli* **3**, **4** i **5**. Dakle za ***RedHat/CentOS*** **6.x.** napravimo sljedeće:

```
chkconfig --level 345 xinetd on
```

Pošto zapravo `xinetd` pokreće `tftp-server` odnosno servis imena `tftp` on nije direktno vidljiv preko naredbe `chkconfig` odnosno nalazi se na kraju ispod `xinetd` konfiguracije (za ***RedHat/CentOS*** **6.x.**).
Za ***RedHat/CentOS*** **6.x.** pogledajmo kako to sada izgleda (izrezan je samo kraj ispisa naredbe od dolje):

```
chkconfig

xinetd          0:off   1:off   2:off   3:on    4:on    5:on    6:off

xinetd based services:
        tftp:           off
```

Dakle vidimo kako je `tftp` isključen. Uključimo ga sa sljedećom naredbom:

```
chkconfig tftp on
```

Pokrenimo `xinetd` koji će sada pokrenuti i `tftp` servis:

```
service xinetd start
```

Za ***RedHat/CentOS 7.x./8.x*** procedura je slična, stoga prvo trajno aktivirajmo servis, te ga pokrenimo:

```
systemctl enable tftp
systemctl start tftp
```

Provjerimo je li se `xinetd` podigao na **UDP** portu **69**, jer *TFTP* protokol radi na UDP portu *69*.

```
netstat -tunap | grep xinetd

udp        0      0 0.0.0.0:69          0.0.0.0:*             22259/xinetd
```

Dakle sve je u redu jer vidimo da je na svim IP adresama (`0.0.0.0`) našeg računala otvoren port `69`.
Odnosno za ***RedHat/CentOS 7.x./8.x*** procedura provjere rada servisa može biti i sljedeća (ispis je skraćen):

```
systemctl status tftp
● tftp.service - Tftp Server
   Loaded: loaded (/usr/lib/systemd/system/tftp.service; indirect; vendor prese>
   Active: active (running) since Mon 2022-02-07 04:00:47 EST; 16min ago
```



Napredniji servis ***dnsmasq*** u sebi ima integriran ***TFTP*** protokol, pa ga možete koristiti umjesto ***xinetd+tftpd***.
Za instalaciju i konfiguracija ***dnsmasq*** servisa, pogledajte poglavlje:
**25.8.4.4. Kickstart metoda automatizirane instalacije operativnog sustava.**

**Izvori informacija: (717), `man xinetd`, `man tftpd`.**

## 26.3.2. Konfiguracija

Konfiguracijska datoteka *TFTP* servisa se nalazi u datoteci `/etc/xinetd.d/tftp`. Ova datoteka skraćeno izgleda ovako:

```
service tftp
{
        disable = no
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /var/lib/tftpboot
}
```

Važni parametri koje ovdje vidimo su:

- `user` - definira koji korisnik pokreće servis; po mogućnosti ne koristiti `root` korisnika zbog sigurnosti.
- `server_args` - definira direktorij u kojemu se nalaze datoteke koje se dijele preko *TFTP*-a.

Dakle u našem slučaju unutar direktorija `/var/lib/tftpboot/` trebaju se nalaziti sve datoteke i direktoriji koje dijelimo preko *TFTP* protokola.



Ako radimo bilo kakve promjene na konfiguracijskoj datoteci potrebno je restartati `xinetd` servis.



Napredniji servis ***dnsmasq*** u sebi ima integriran ***TFTP*** protokol, pa ga možete koristiti umjesto ***xinetd+tftpd***.
Za instalaciju i konfiguracija ***dnsmasq*** servisa, pogledajte poglavlje:
**25.8.4.4. Kickstart metoda automatizirane instalacije operativnog sustava.**

## 26.3.3. Upotreba i primjeri

Na prvom računalu **PC-A** instalirajmo ***TFTP*** klijent te se probajmo spojiti na drugo računalo koje je ***TFTP*** poslužitelj: **PC-B**.

```
yum -y install tftp
```

Sada kreirajmo datoteku imena `proba.txt` na ***TFTP*** poslužitelju **PC-B** unutar direktorija `/var/lib/tftpboot`.

```
echo 12345 > /var/lib/tftpboot/proba.txt
```

Pošto smo korisnik `root`, uđimo u direktorij `/root/` gdje će se i kopirati sve datoteke preko ***TFTP*** protokola.
S prvog računala se spojimo na drugo računalo (**PC-B :** 10.10.10.20) upotrebom `tftp` klijenta koji smo upravo instalirali.

```
[root@PC-A ~]# cd /root/
[root@PC-A ~]# tftp 10.10.10.20
```

Sada prekopirajmo datoteku imena `proba.txt` te izađimo iz *TFTP* klijenta:

```
tftp> get proba.txt
tftp> quit
```

*TFTP* protokol podržava dvije osnovne operacije (i naredbe) nad datotekama:

- `get` *ime_datoteke* – je za dohvat/kopiranje (*download)* datoteke točno definiranog imena sa *TFTP* poslužitelja.
- `put` *ime_datoteke* – je snimanje (*upload*) na *TFTP* poslužitelj; datoteke točno definiranog imena.

Na našem računalu (**PC-A**) pogledajmo datoteku koju smo kopirali s *TFTP* poslužitelja:

```
[root@PC-A ~]# cat proba.txt

12345
```

Važno je razumjeti kako nije moguće izlistati datoteke koje se nalaze na *TFTP* poslužitelju, kao kod *FTP* protokola, već je potrebno znati točno ime datoteke koju želimo dohvatiti (kopirati). ***TFTP*** se oslanja na **UDP** kao transportni protokol, ali ga i nadopunjuje na poseban način. Naime od **UDP**-a se očekuje da brzo (i nepouzdano) prenese pakete odnosno podatke na odredište, bez ikakve potvrde od primatelja: je li ih zaprimio; kao što to radi ***TCP***. Kako bi ***TFTP*** prijenos bio pouzdan, u ***TFTP*** protokol na aplikacijskoj razini je implementirana provjera, tako da se *TFTP* aplikacija praktično ponaša kao **TCP** protokol: nakon svakog zaprimljenog ***UDP*** paketa, ***TFTP*** klijent šalje *TFTP* poslužitelju potvrdu da ga je uredno zaprimio (***TFTP*** **[UDP]** paket). Tako da u konačnici na svaki poslani ***TFTP*** paket, primatelj šalje pošiljatelju jedan ***TFTP*** (**UDP**) potvrdni paket (***TFTP Opcode: Acknowledgement (4)***). Dakle to je ***TFTP*** poruka koja se transportira preko ***UDP*** protokola.
Veličina (***UDP***) paketa koji se šalju na mrežu je **512** bajta, što je moguće povećati, ali je to potrebno postaviti i na klijentu (***TFTP*** opcija kod dohvaćanja datoteke: ***53 RRQ***: `blksize` ***XY***) i poslužitelju (pr. -***blocksize*** ***XY***, a ovisi o implementaciji).

$$Maksimalna\ veličina\ ovog\ bloka = \boldsymbol{MTU} - \boldsymbol{zaglavlja}\ (TFTP\ [4\ bajta]\ +\ UDP\ [8\ bajta] + IP\ [20\ bajta]\ )$$

**Izvori informacija: `man tftp`, `man xinetd`, RFC783, RFC2347, RFC2348, RFC906.**

# 26.4. HTTP

*HTTP* (engl. *Hypertext Transfer Protocol*) je aplikacijski protokol koji je i osnova današnjeg Web-a (*World Wide Web*).
*HTTP* protokol za transport koristi ***TCP*** protokol, na portu 80, odnosno za *HTTPS* (kriptiranu/sigurnu inačicu *HTTP* protokola) koristi **TCP** na portu 443. Na Linux sustavima postoji nekoliko različitih poslužitelja za *HTTP* protokol.
Mi ćemo koristiti jedan od najpoznatijih koji se zove ***Apache***.
Osim njega često se koriste: ***Nginx***, ***Lighttpd***, ***Apache Tomcat***, ali i neki drugi.

## 26.4.1. Instalacija

Provjerimo je li *Apache* već instaliran. U ***CentOS/RedHat/Fedora*** distribucijama Linuxa servis koji je zadužen za ***Apache*** poslužitelj se zove `httpd`. Isto tako se zove i softverski paket. Provjerimo je li on već instaliran:

```
rpm -qa httpd
```

Po svemu sudeći nije instaliran pošto ga naš paketni menadžer *RedHat-a* nije pronašao na našem računalu.

Instalirajmo ga (ispis je skraćen za potrebe prikaza):

```
yum -y install httpd
```

I nakon nekoliko trenutaka, on će biti instaliran.
Dakle sada je sve potrebno instalirano, uključujući pakete o kojima je ovisila instalacija našeg željenog `httpd` paketa.

## 26.4.2. Konfiguracija i pokretanje

Konfiguracijska datoteka *Apache* odnosno `httpd` servisa se nalazi u datoteci: `/etc/httpd/conf/httpd.conf`.
Unutar ove datoteke postoji cijeli nîz konfiguracijskih parametara od kojih ćemo proći sâmo nekoliko osnovnih potrebnih za rad. Dijelovi konfiguracije koji nisu u upotrebi su "*zakomentirani*" odnosno na početku reda se nalazi znak `#`. Tako je i sa sâmim imenom *HTTP* poslužitelja koje bi trebalo biti definirano uz ime domene kojoj pripada.
Prvo otvorimo (uredimo) datoteku: `/etc/httpd/conf/httpd.conf` te pronađimo red u kojem se nalazi:

```
#ServerName
```

Ispod gornjeg retka dodajmo naš novi redak s imenom našeg ***HTTP*** poslužitelja (`test.testlab.hr`):

```
ServerName test.testlab.hr
```

Ovdje smo imenovali naš ***HTTP*** poslužitelj i njemu pripadajuću domenu. Pogledajmo još neke od osnovnih parametara:

- `StartServers 8` - označava koliko *servisa* će se pokrenuti odjednom kod podizanja *HTTP* poslužitelja.
- `MinSpareServers 5` - označava minimalni broj poslužitelja koji će dodatno biti pokrenuti.
- `MaxClients 256` - označava ograničenje broja spojenih klijenata na svaki pojedini *HTTP* poslužiteljski servis.

Pokrenimo sada *Apache* servis:

```
service httpd start
```

Provjerimo je li stvarno sve u redu odnosno radi li HTTP protokol na TCP portu 80:

```
netstat -tunap | grep httpd

tcp        0      0 :::80              :::*                        LISTEN      1198/httpd
```

Vidimo da je TCP port `80` otvoren, a za njega je zadužen servis `httpd` aktivan. Dakle sve je u redu sa strane mreže.

Sada pogledajmo što se događa sa strane Linux procesa:

```
ps -ef | grep httpd

root      1198     1  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1200  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1201  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1202  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1203  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1204  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1205  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1206  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
apache    1207  1198  0 19:23 ?        00:00:00 /usr/sbin/httpd
```

U ispisu vidimo kako je osam procesa pokrenuto od strane korisnika `apache` kako bi i trebalo biti jer su oni su zaduženi za kompletnu funkcionalnost *HTTP* poslužiteljskog protokola. Dakle ukupno osam *apache* (`httpd`) procesa čeka na korisnike.
Iz prijašnjeg ispisa (`netstat -tunap | grep httpd`) vidimo i jedan proces pokrenut od strane korisnika `root`, što je isto normalno jer ovaj korisnik (*root*=administrator na Unix/Linux sustavima) pokreće inicijalni proces koji otvara **TCP** port broj **80**, zadužen za *HTTP* protokol. Ovaj inicijalni proces je uvijek u stanju slušanja (`LISTENING`) što se mreže tiče.

Navedeno je važno zbog tri stvari:

- Samo `root` korisnik ima pravo na sustavu otvarati portove koji su ispod broja 1023, zbog sigurnosnih razloga.
- Ovaj proces koji je pokrenuo `root` korisnik, pokreće sve pôd procese koji su zapravo odgovorni za sâm ***http*/web** poslužitelj. Osim toga ovaj proces se brine i o pokretanju novih *HTTP* pôd procesa (`httpd`) prema potrebi.
- Prvi `httpd` proces (***PID***: `1198`) zapravo sâmo prosljeđuje promet na pôd procese koji ga u konačnici i obrađuju.

Pogledajmo stablo procesa ispod našeg (vršnog) web servisa `httpd`:

```
pstree -p | grep httpd

        |-httpd(1198)-+-httpd(1200)
        |              |-httpd(1201)
        |              |-httpd(1202)
        |              |-httpd(1203)
        |              |-httpd(1204)
        |              |-httpd(1205)
        |              |-httpd(1206)
        |              `-httpd(1207)
```

Sada je sve puno jasnije: u zagradama su "*Process ID*" odnosno ***PID*** brojevi svakog procesa. Vidljivo je da je jedan (vršni) proces (***PID***: `1198`) pokrenuo osam pôd procesa s ***PID*** brojevima: `1200-12007`, kako smo i očekivali.
Podsjetimo se da su ***PID*** brojevi jedinstveni identifikacijski brojevi za svaki pokrenuti program (proces) unutar Linuxa.
Pošto nam je ovih osam procesa previše i nepotrebno, a konfigurirano je da jedan *httpd* proces može obrađivati do 256 korisnika, što ukupno znači 8×256=2048 istovremenih korisnika. Pošto mi ne očekujemo toliku navalu na naš HTTP poslužitelj smanjit ćemo ovu brojku. Ponovno otvorimo datoteku: `/etc/httpd/conf/httpd.conf` te pronađimo retke u kojima se nalaze sljedeći unosi:

```
StartServers       8
MinSpareServers    5
```

Zatim promijenio ovaj broj u `2` za oba parametra jer želimo da se pokrenu sâmo 2 *httpd* servisa (procesa) i maksimalno također dva rezervna (`MinSpareServers`). Odnosno u konačnici će ukupno biti pokrenuta sâmo dva inicijalna procesa.

```
StartServers       2
MinSpareServers    2
```

Snimimo datoteku te restartajmo (za **RedHat/CentOS 6.x**) naš *httpd* servis:

```
service httpd restart
```

Odnosno za **RedHat/CentOS 7.x** ili noviji, restartajmo naš *httpd* servis na sljedeći način:

```
systemctl restart httpd
```

Provjerimo koliko sada imamo `httpd` procesa:

```
ps -ef | grep httpd

root      1671     1  0 19:45 ?        00:00:00 /usr/sbin/httpd
apache    1673  1671  0 19:45 ?        00:00:00 /usr/sbin/httpd
apache    1674  1671  0 19:45 ?        00:00:00 /usr/sbin/httpd
```

Uspjeli smo pa su se pokrenuta sâmo dva, tako da se na naš novi *Web* poslužitelj sada istovremeno može spojiti 512 korisnika.

## Automatsko pokretanje Apache http (web) poslužitelja

Ako želimo da se naš Apache http poslužitelj pokreće tijekom svakog pokretanja ili restarta Linux sustava, to je jednostavno:
**Za RedHat/CentOS 6.x.**

```
chkconfig httpd on
```

**Za RedHat/CentOS 7.x.** ili noviji trebamo napraviti sljedeće:

```
systemctl enable httpd
systemctl start httpd
```

I to je to.



*HTTP* protokol je prvo (v.1.1.) definiran u RFC2068, da bi potom bio nadograđen sa RFC2616 te RFC7230. Dok je *HTTP/2* definiran u RFC7540 i u konačnici proširen sa RFC8164. Najveća novost *HTTP/2* je u efikasnijem iskorištavanju konekcije: uvođenjem *multipleksiranja* (višestrukog/paralelnog prenošenje) upita i odgovora, komprimiranja zaglavlja te prioritizaciji upita, kao i efikasnijim mehanizmima za streaming podataka. U novije vrijeme se pojavio i HTTP/3 koji je trenutno (početak 2021.g.) u izradi, a njegova važna novost je upotreba novog QUIC protokola za prijenos podataka. Naime zbog načina *multipleksiranja* poruka u *HTTP/2* i sâme **TCP** veze kao i njene propusnosti te vremena odziva odnosno kašnjenja, osmišljen je novi protokol ***QUIC***. **QUIC nudi brzinu i odziv približno poput UDP-a, ali i pouzdanost poput TCP-a.** ***QUIC*** se oslanja na **UDP** kao transportni protokol, a koristi se sâmo za *HTTP/HTTPS* aplikacijski protokol.

**Izvor informacija:** `man httpd`.

### 26.4.3. Izgled HTTP paketa na mreži

U primjeru koji slijedi gledat ćemo kako izgledaju ***HTTP*** paketi kada završe na mreži. U našem Web pregledniku ćemo otvoriti sljedeću web stranicu/adresu: http://www.mock-server.com/ što će uzrokovati slanje ***HTTP GET*** paketa na mrežu.
Prije nego se poslao ovaj paket, sustav je morao poslati ***DNS*** upit, kako bi saznao da je za web poslužitelj: `www.mock-server.com` pripadajuća IP dresa: `99.86.243.121`. Pogledajmo ***HTTP*** paket prema **OSI** slojevima (kraj smo skratili):

```
(1)Frame 285: 559 bytes on wire(4472 bits),559 bytes captured(4472 bits) on interface 0
(2)Ethernet II, Src: (2c:d4:44:b3:5f:b6), Dst: (d4:ca:6d:71:8a:23)
(3)Internet Protocol Version 4, Src: 192.168.1.20, Dst: 99.86.243.121
(4)Transmission Control Protocol, Src Port:50656,Dst Port:80, Seq: 1,Ack: 1,Len:505
 Hypertext Transfer Protocol
    GET / HTTP/1.1\r\n
    Host: www.mock-server.com\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0)
                Gecko/20100101 Firefox/71.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Referer: https://www.google.com/\r\n
    Connection: keep-alive\r\n
    Cookie: _ga=GA1.2.75156916.1576148739; _gid=GA1.2.269235266.1576148739;
```

Na OSI sloju dva `(2)` vidimo da paket šaljemo s naše MAC adrese `2c:d4:44:b3:5f:b6` na odredišnu IP adresu našeg usmjerivača (Engl. *Default Gateway*) `d4:ca:6d:71:8a:23`. Na OSI sloju tri `(3)` vidimo našu IP adresu `192.168.1.20` i konačnu odredišnu IP adresu `99.86.243.121` na koju se paket mora poslati za web poslužitelj imena: `www.mock-server.com`, koju je naš sustav dobio pomoću DNS upita koji je morao poslati prije ovog paketa. I sada u OSI sloju četiri `(4)` odnosno transportnom sloju, vidimo da je odredišni (***TCP***) port (`Dst Port`) broj `80` koji definira da se radi o *HTTP* protokolu.
Potom u sljedećem OSI sloju kreće ***HTTP*** komunikacija koju ćemo ukratko objasniti:

- `GET / HTTP/1.1\r\n` - označava da se radi o HTTP poruci vrste: `GET`.
- `Host:` - označava da slijedi HTTP adresa poslužitelja na koji se spajamo: `www.mock-server.com`
- `User-Agent:` -ovdje slijede podaci koji se šalju na Web poslužitelj o Web pregledniku koji se koristi.
- `Accept:` - označava da slijede podaci o vrsti sadržaja (kao ***MIME*** *tip*) koje podržava klijent (Web preglednik)
- Slijede jezik koji se očekuje (`Accept-Language`) kao i vrsta enkodiranja (`Accept-Encoding`) te ostali podaci.

Slijedi ***HTTP*** odgovor odnosno novi mrežni paket koji dolazi od Web/HTTP poslužitelja prema našem web pregledniku:

```
(1) Frame 394: 809 bytes on wire (6472 bits), 809 bytes captured (6472 bits)
(2) Ethernet II, Src: (d4:ca:6d:74:8f:23), Dst: (2c:d4:44:b3:5f:b6)
(3) Internet Protocol Version 4, Src: 99.86.243.121, Dst: 192.168.1.20
(4) Transmission Control Protocol,SrcPort:80,DstPort:50656,Seq:5841,Ack:506,
Len:755
   [5 Reassembled TCP Segments (6595 bytes): #389(1460), #390(1460), #391(1460),
#393(1460), #394(755)]
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Content-Type: text/html\r\n
    Transfer-Encoding: chunked\r\n
    Content-Encoding: gzip\r\n
    [HTTP response 1/5]
    [Time since request: 0.014234000 seconds]
    [Request in frame: 285]
    [Next request in frame: 407]
    [Next response in frame: 416]
    [Request URI: http://www.mock-server.com/]
    HTTP chunked response
    Content-encoded entity body (gzip): 6078 bytes -> 25149 bytes
    File Data: 25149 bytes
Line-based text data: text/html (700 lines)
    <!DOCTYPE html>\n
    <html lang="en" xmlns:og="http://ogp.me/ns#" prefix="og: http://ogp.me/ns#">\n
    \n
    \t<head>\n
    \t<!-- meta tags -->\n
    \t<meta charset="utf-8" />\n
```

Ovo je odgovor s Web/HTTP poslužitelja koji nam dolazi preko našeg usmjerivača (MAC: `d4:ca:6d:74:8f:23`) `(2)` s IP adrese sâmog Web/HTTP poslužitelja `99.86.243.121` - OSI sloj tri `(3)`. Ovaj odgovor dolazi s izvorišnog TCP porta broj `80` jer je ovo odgovor s HTTP poslužitelja pa se šalje s njegovog TCP porta 80.
Zatim uz ostale podatke slijedi sâm HTML dokument i to nakon polja: `Line-based text data: text/html`.
Važno je razumjeti da je konkretan sadržaj (*payload*) razlomljen u pet TCP segmenata (paketa), kako je vidljivo na OSI sloju četiri `(4)` kao poruka: `5 Reassembled TCP Segments`.

**Izvor informacija: (802), RFC 2616, RFC 7230, RFC 7231.**

## 26.4.4. Secure Socket Layer, Transport Layer Security i kriptografija

***HTTPS*** protokol (*HyperText Transfer Protocol **Secure***) je internetski protokol koji je nastao kao proširenje klasičnog protokola *Hypertext Transfer Protocol* (***HTTP***). Koristi se za sigurnu komunikaciju preko računalne mreže, a naširoko se koristi na Internetu. Upotrebom *HTTPS-a,* komunikacijski kanal je kriptiran (šifriran) korištenjem nekog od dva protokola: *Transport Layer Security* (**TLS**) ili, ranije korištenog *Secure Sockets Layer* (SSL) protokola. Stoga se često za HTTPS kaže da koristi HTTP protokol preko TLS (ili SSL-a) kanala. Naime svaki puta kada bi u svom web pregledniku otvorili adresu poput: **http**://www.google.com/ to bi značilo da se želite spojiti na udaljeni web poslužitelj na navedenoj adresi, ali preko nesigurnog **HTTP** protokola. Nesiguran znači da gotovo bilo tko može vidjeti vašu komunikaciju; primjerice unošenje korisničkog računa i lozinke za spajanje na web poslužitelj, ali i sve druge (osjetljive) podatke. Srećom u novije vrijeme sve moderne web stranice odnosno web poslužitelji koriste sigurniju **HTTPS** inačicu, pa bi stoga ista adresa upotrebom **HTTPS** protokola bila: **https**://www.google.com (uočite ***https***).

**http://www.google.com**

Nesigurna (nezaštićena ***HTTP***) komunikacija

**https://www.google.com**

Sigurna (zaštićena ***HTTPS***) komunikacija.

**Transport Layer Security** (***TLS***) protokol, nasljednik je sada zastarjelog protokola *Secure Socket Layer* (***SSL***). Oni su kriptografski protokoli dizajnirani za pružanje sigurne komunikacije preko (nesigurne) računalne mreže poput interneta. *TLS* protokol se naširoko koristi u mnogim aplikacijama kao što su elektronička pošta, *Voice over IP, Instant Messaging* i drugima. Međutim njegova upotreba u osiguravanju HTTP protokola odnosno prometa, javno je najvidljivija. TLS možemo promatrati kao kriptirani (šifrirani) komunikacijski kanal, a kao takvog ga i koriste mnoge druge aplikacije i sustavi, primjerice ***OpenVPN*** koristi ***TLS*** za VPN tunele.
*TLS* protokol prvenstveno ima za cilj osigurati kriptografsku zaštitu, uključujući privatnost (povjerljivost), integritet i autentičnost korištenjem certifikata, između dvije ili više računalnih aplikacija koje komuniciraju.
On radi u aplikacijskom sloju i sâm se sastoji od dva sloja: TLS zapisa i TLS protokola dogovaranja parametara uspostavljanja veze i parametara rada (engl. *handshake*). Aplikacije koje koriste TLS protokol rade prema principu klijent-poslužitelj, a one su za komunikaciju preko mreže dizajnirane na način da se spriječi prisluškivanje i neovlašteno korištenje odnosno pristup podacima u prijenosu. Budući da aplikacije mogu komunicirati sa ili bez TLS-a (ili SSL-a), potrebno je da klijent zatraži od poslužitelja uspostavu TLS veze. Jedan od standardnih načina da se to postigne je korištenje drugog broja porta za TLS veze.

Na primjer, port 80 obično se koristi za nekriptirani HTTP promet, dok je port 443 uobičajeni port koji se koristi za kriptirani (šifrirani) HTTPS promet. Drugi mehanizam je da klijent uputi zahtjev specifičan za TLS protokol, poslužitelju za prebacivanje veze na TLS. Na primjer, postavljanjem ***STARTTLS*** zahtjeva pri uspostavi veze kada se koristi protokol za elektroničku poštu.

Za implementaciju TLS i SSL-a pod Linuxom koriste se OpenSSL biblioteke. *OpenSSL* biblioteke sadrže implementaciju otvorenog kôda za SSL i TLS protokole. Osnovna *OpenSSL* biblioteka napisana je u programskom jeziku C, a implementira osnovne kriptografske funkcije, ali pruža i razne uslužne funkcije. Dostupni su i dodaci (*wrapperi*) koji omogućuju korištenje *OpenSSL* biblioteka u mnogim programskim jezicima. Ove biblioteke su obično već instalirane, a ako nisu to možete učiniti sa:

```
yum -y install openssl
yum -y openssl-libs
```

***GnuTLS*** je druga implementacija za TLS, SSL i DTLS protokole, također otvorenog kôda, ali s drugačijom licencom. Ona nudi sučelje za programiranje aplikacija (***API***) za aplikacije koje omogućavaju sigurnu komunikaciju preko mrežnog transportnog sloja, kao i sučelja za pristup za: ***X.509***, ***PKCS #12***, ***OpenPGP*** i drugim strukturama.
Za instalaciju *GnuTLS* i drugih potrebnih programa i biblioteka napravite sljedeće:

```
yum -y install gnutls
yum -y install gnutls-utils
yum -y install nss-tools
```

***Vratimo se na uspostavljanje TLS/SSL komunikacijskog zaštićenog kanala***
Nakon što klijent i poslužitelj pristanu na korištenje TLS-a, oni pregovaraju o uspostavljanju sigurne veze pomoću procedure takozvanog rukovanja (engl. *handshake*). TLS i SSL protokoli koriste mehanizam uspostave veze mehanizmom ***asimetrične kriptografske metode*** za uspostavljanje ne samo postavki lozinke, već i zajedničkog ključa specifičnog za sesiju s kojim se potom daljnja komunikacija kriptira odnosno šifrira pomoću ***simetrične*** lozinke.

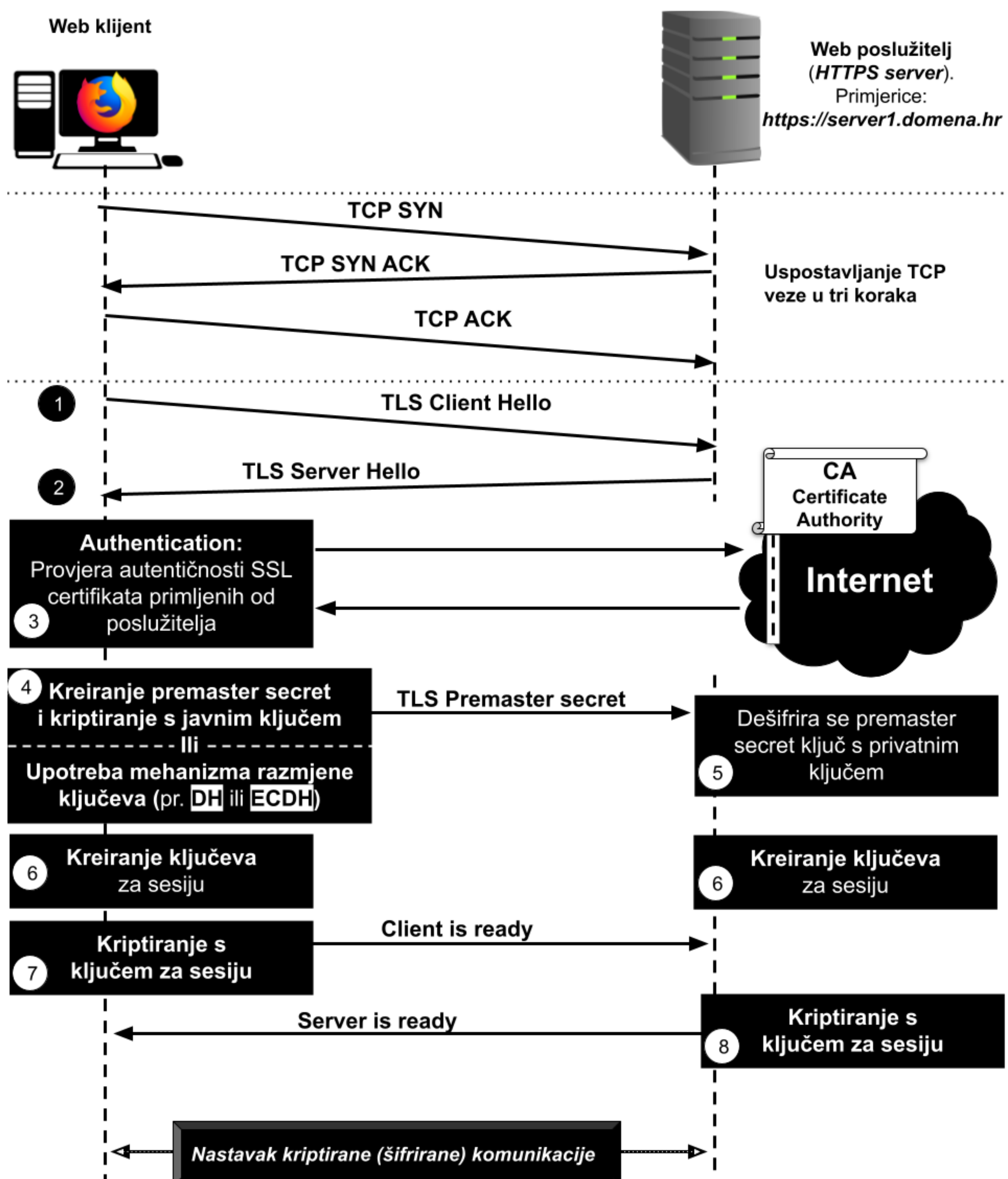
***O simetričnim i asimetričnim kriptografskim mehanizmima, govorit ćemo nešto kasnije!***

Za listu svih *kripto paketa* koji su podržani na sustavu, pokrenite sljedeću naredbu:

```
openssl ciphers
TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:TLS_AES_12
8_CCM_SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES256-CCM:ECDHE-ECDSA-
AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-CCM:ECDHE-ECDSA-
AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-
SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:AES256-GCM-SHA384:AES256-CCM:AES128-
GCM-SHA256:AES128-CCM:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:DHE-RSA-
AES256-GCM-SHA384:DHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES256-CCM:DHE-RSA-AES128-GCM-
SHA256:DHE-RSA-AES128-CCM:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA:PSK-AES256-GCM-SHA384: . . .
```

Za detalje o *kripto paketima*, pogledajte sljedeću sliku (244.A.) i korake (**1.b.** i **2.c.**)

Slika 244.A. Pogled na TLS/SSL komunikaciju:




Tijekom uspostavljanja ***TLS*** veze (tzv. rukovanja), klijent i poslužitelj dogovaraju se o različitim parametrima koji se koriste za uspostavljanje sigurnosti veze. Pri tome postoje dvije metode rada:

## Asimetrična RSA metoda

TLS uspostava veze počinje kada se klijent poveže s poslužiteljem s omogućenim TLS-om i zahtijeva sigurnu vezu.

1. U prvom koraku **klijent** šalje (***TLS Client Hello***) popis podržanih kriptografskih paketa: algoritama kriptiranja, izrade provjernog zbroja [*hash*] i drugo. Dakle klijent i poslužitelj se prvo odgovaraju:
   a. O inačici TLS-a (ili starijeg SSL-a) koju će koristiti. Primjerice TLS v.1.1 ili 1.2. i slično.
   b. Dogovaraju se koji kriptografski paket će koristiti. Na slici 244.A. pogledajte jedan primjer kriptografskog paketa, na način da klijent šalje poslužitelju (***TLS Client Hello***) što sve podržava. Izbor izgleda poput: ***TLS_AES256-GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256, TLS_AES128-CBC_DHE-RSA_SHA256, …***
   c. Šalje niz nasumičnih bajtova poznatih kao: ***Client random***.
2. **Poslužitelj** odgovara klijentu (***TLS Server Hello***):
   a. Da očekuje određenu minimalnu inačicu TLS-a (pr. TLS v.1.2) ispod koje neće raditi.
   b. Šalje mu svoj: ***SSL certifikat, javni ključ*** *(public key)* i druge detalje vezane za certifikat/domenu.
   c. Šalje mu svoj (poslužiteljski) odabrani popis *kriptografskih paketa*.

d. Šalje i drugi nasumični niz koje generira poslužitelj, a koji se naziva ***server random***, za čiji izračun se koriste: *vrijeme i datum, slučajno generirani brojevi, unikatni identifikator poslužitelja, … .*

3. **Autentikacija**: **klijent** provjerava SSL certifikat poslužitelja kod certifikacijskog tijela koje ga je izdalo (***Certificate Authority***). To potvrđuje da je poslužitelj onaj koji tvrdi da jest i da klijent komunicira sa stvarnim vlasnikom domene (i certifikata) za koju se izdaje.
4. ***Kreiranje ključeva za sesiju (Premaster secret)* – klijent** šalje još jedan nasumični nîz bajtova koji se naziva "***premaster secret***". Ovaj nîz šifriran je ***javnim ključem*** poslužitelja upotrebom ***modulusa****, a poslužitelj ga može dešifrirati samo svojim privatnim ključem, koji je dio pâra: *javni+privatni* ključ. Pri tome je klijent izvukao javni ključ iz SSL certifikata koji je dobio od poslužitelja.
5. **Poslužitelj** dešifrira ***premaster secret*** pomoću svog privatnog ključa jer su podaci šifrirani javnim ključem koji je dio pâra: *javni+privatni* ključ od poslužitelja. *Pri tome se privatni ključ (nikada) ne dijeli nikome*.
6. ***Kreiranje ključeva sesije (Master secret):*** i **klijent** i **poslužitelj** generiraju ključeve sesije (***Master secret***) na osnovi: ***client random***, ***server random*** i ***premaster secret***. Pri tome bi i poslužitelj i klijent bi trebali doći do istih rezultata.
7. Klijent je spreman za rad te šalje poruku "***finished***" koja je kriptirana ključem sesije (***Master secret***).
8. Poslužitelj je spreman za rad te šalje poruku "***finished***" koja je također kriptirana ključem sesije.
9. Sigurna (zaštićena) komunikacija se nastavlja te je kriptirana ključem sesije.

**Diffie-Hellman (DH) RSA metoda razmjene ključeva**

1. Uspostava TLS veze (**TLS Client Hello**) – isto kao kod *asimetrične **RSA*** metode odnosno klijent šalje svoj popis podržanih kriptografskih paketa prema poslužitelju.
2. **Server Hello –** isto kao i kod klasične ***RSA*** metode: poslužitelj šalje odgovor sa serverskom listom kriptografskih paketa koje on dozvoljava, ***SSL certifikatom, javnim ključem*** i drugim detaljima, ali se dodaje ***digitalni potpis poslužitelja***. Poslužitelj koristi svoj privatni ključ za kriptiranje ***client random***, ***server random*** i svog ***DH parametra****. Ovi kriptirani podaci funkcioniraju kao digitalni potpis poslužitelja, utvrđujući da poslužitelj ima privatni ključ koji se podudara s javnim ključem iz TLS (SSL) certifikata.
3. ***Potvrda digitalnog potpisa:*** klijent prima certifikat web poslužitelja i provjerava da nije istekao. Klijent će također zatražiti od ***CA***-a da zatraži kopiju "popisa opoziva certifikata" (engl. *Certificate Revocation List*) (**CRL**). Ova provjera osigurava da certifikat nije opozvan iz bilo kojeg razloga: kao što je eventualna kompromitiranost ključa, zadržavanje certifikata, datum isteka i bilo što drugo što bi moglo učiniti certifikat nevažećim. Pretpostavimo da je certifikat valjan. Klijent također provjerava digitalni potpis dešifrirajući primjerice sa SHA-256 *hash* javnim ključem, a zatim ponovno izračunava *provjerni* zbroj (hash) kako bi vidio podudaraju li se obije vrijednosti. Klijent šalje svoj ***DH parametar**** poslužitelju.
4. Klijent i poslužitelj izračunavaju ***premaster secret***: umjesto da klijent generira ***premaster secret*** i šalje ga poslužitelju, kao u ***RSA*** uspostavi veze, klijent i poslužitelj koriste ***DH parametre**** koje su razmijenili kako bi zasebno izračunali odgovarajući ***premaster secret***.
5. ***Kreiranje ključeva sesije (Master secret)***: sada, klijent i poslužitelj izračunavaju ključeve sesije iz: ***premaster secret-a***, ***client random*** i ***server random***, baš kao i u ***RSA*** uspostavi veze.
6. Klijent je spreman za rad te šalje poruku "***finished***" koja je kriptirana ključem sesije.
7. Poslužitelj je spreman za rad te šalje poruku "***finished***" koja je kriptirana ključem sesije.
8. Sigurna (zaštićena) komunikacija se nastavlja te je također kriptirana ključem sesije.

Pogledajmo tablicu s dostupnim algoritmima za razmjenu ili dogovaranje razmjene kripto ključeva:

| Naziv/oznaka algoritma | Opis | Komentar |
|---|---|---|
| **DH_anon** | Anonymous Diffie–Hellman | Prva inačica DH algoritma koja je u današnje vrijeme nepouzdana. |
| **DH** | Diffie-Hellman | Pouzdani algoritam za razmjenu kriptografskih ključeva, koji ne koristi dijeljeni ključ. |
| **DHE** | *Ephemeral Diffie-Hellman* | Modifikacija izvornog ***DH*** algoritma u kojem se koristi privremeni (efemerni) statični ključ. |
| **ECDH** | Elliptic curve Diffie-Hellman | Druga modifikacija ***DH*** algoritma koja koristi mehanizam tzv. eliptične krivulje. |
| **ECDHE** | *Ephemeral elliptic-curve Diffie–Hellman* | Još jedna modifikacije izvornog ***DH*** algoritma koji je kombinacija prethodna dva. |
| **PSK** | Pre-shared key | Radi na principu dijeljenja unaprijed definiranog simetričnog ključa. Zbog čega je sam po sebi potencijalno nesiguran, osim ako se ne kombinira s ***DH*** metodama. |
| **SRP** | Secure Remote Password | Mehanizam koji može koristiti ***SRP*** metodu preko password-authenticated key exchange (PAKE) protokola ili uz dodatnu upotrebu ***PKI*** certifikat. |

****Modulus** u kriptografiji je veliki broj koji se dobiva od dva velika **prosta** broja, a koristi se u kriptografskim algoritmima!*

| ISAKMP | Internet Security Association and Key Management Protocol | Protokol za uspostavu atributa veze (SA) te razmjenu kriptografskih ključeva primjerice upotrebom IKE ili KINK mehanizama.<br>Koristi se primjerice u IPSec protokolu. |
|---|---|---|

Pogledajmo i tablicu s dostupnim algoritmima za digitalno potpisivanje:

| Oznaka algoritma | Naziv | Komentar |
|---|---|---|
| **DSA** | Digital Signature Algorithm | Više se ne preporuča njegova primjena |
| **ECDSA** | Elliptic Curve Digital Signature Algorithm | Moderan i siguran za upotrebu |
| **RSA** | Rivest–Shamir–Adleman | Najsporiji i najstariji. Iako je on i algoritam za kriptiranje, obično se koristi samo za kriptiranje i prijenos simetričnih ključeva, ali ne i drugih podataka. |
| **EdDSA** | Edwards-curve Digital Signature Algorithm | Moderan i siguran za upotrebu.<br>Dostupna je i varijanta: **Ed25519** (EdDSA+SHA512+Curve25519) |

Pogledajmo tablicu s dostupnim algoritmima za izradu provjernog zbroja odnosno takozvane *hash* funkcije:

| Oznaka algoritma | Naziv | Komentar |
|---|---|---|
| **MD5** | Digital Signature Algorithm | Više se ne preporuča primjena u kriptografiji**.** |
| **SHA-0**<br>**SHA-1** | *Secure Hash Algorithm* | Više se ne preporuča primjena u kriptografiji. |
| **SHA-2** | Oznake/izlazni broj bitova:<br>**SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224** i **SHA-512/256**. | Siguran za primjenu. |

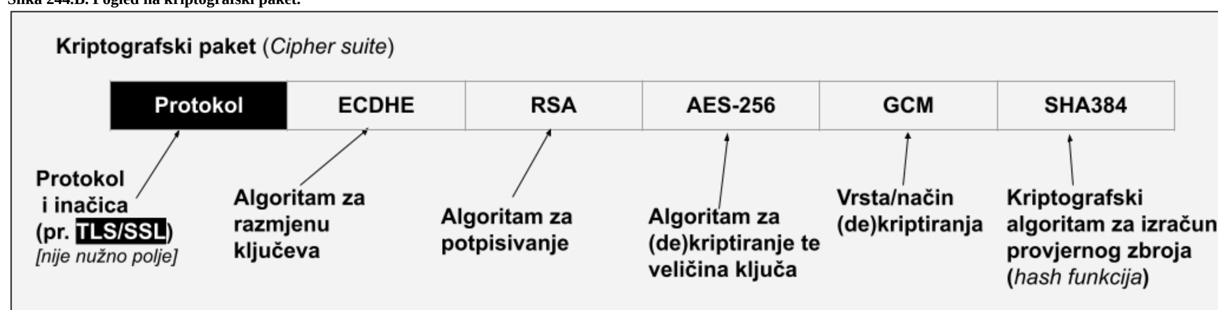
Pogledajmo tablicu s često korištenim algoritmima za kriptiranje:

| Naziv/oznaka algoritma | Opis |
|---|---|
| **AES** | *Advanced Encryption Standard* - moderan i siguran algoritam za kriptiranje (šifriranje) podataka. Definiran u: **FIPS PUB 197** i **ISO/IEC 18033-3**. |
| **ARIA** | Baziran na ***AES***, smatra se modernim i sigurnim algoritmom. Definiran u **RFC 5794**. |
| **DES**<br>**3DES** | Nesigurni i treba ih izbjegavati. |
| **Camellia** | Smatra se modernim i sigurnim algoritmom. Definiran u **RFC 3713**. |
| **ChaCha20** | **ChaCha20** - moderan i siguran algoritam, postiže veliku brzinu u radu, čak 3-5 puta veću od primjerice ***AES-GCM***, u slučaju kada za AES ne postoji hardverska podrška u procesoru (***AES-NI***). Koristi se u kombinaciji s **Poly1305**. Definiran u **RFC 7905**. |
| **RSA** | Rivest–Shamir–Adleman - najstariji i najsporiji. U praksi se koristi samo za kriptiranje i prijenos simetričnih ključeva, čak i u kombinaciji s gore navedenima koji se u pravilu koriste za kriptiranje svih ostalih podataka. |
| | **Message authentication code (MAC) i drugi mehanizmi** |
| **CBC** | *Cipher block chaining* - za uvezivanje blokova podataka i dodatne kripto operacije. Radi u kombinaciji s algoritmima za kriptiranje (pr. **AES**). Njegova upotreba se ne preporuča te radi najsporije. RFC 3268. |
| **CCM** | *Counter with cipher block chaining message authentication code* **(MAC)-** kombinira CBC i *counter* načina rada te radi dva AES izračuna po bloku podataka. Sigurniji i brži od **CBC** ali nešto sporiji od **GCM**. Definiran u **RFC 6655.** |

| Naziv algoritma | Message authentication code (MAC) i drugi mehanizmi |
|---|---|
| **GCM** | *Galois Counter Mode* **(MAC)**- Preporučen od TLS v.1.2. Najbrži od gornja dva, dodatno jer podržava i paralelno procesiranje (obradu). Definiran u **RFC 5288**. |
| **Poly1305** | *Poly1305* - za provjeru autentičnosti (*Message authentication Code*) (**MAC**), obično se koristi u kombinaciji s **ChaCha20** ili **AES**. Definiran u **RFC 7905** |

Sada na slici **244.B.** pogledajmo kako izgleda jedan kriptografski paket odnosno od kojih dijelova se sastoji.

**Slika 244.B. Pogled na kriptografski paket.**




---

# Certifikat javnog ključa (PKI)

U kriptografiji, certifikat javnog ključa, također poznat kao digitalni certifikat ili certifikat identiteta, je elektronički dokument koji se koristi za dokazivanje valjanosti javnog ključa. Certifikat uključuje podatke o ključu, podatke o identitetu njegovog vlasnika (koji se naziva subjekt) i digitalni potpis entiteta koji je ovjerio sadržaj certifikata (koji se naziva *izdavatelj*).
Ako je potpis valjan, a softver koji ispituje certifikat vjeruje izdavatelju, tada može koristiti taj ključ za sigurnu komunikaciju sa subjektom certifikata. Primjerice u sustavima enkripcije e-pošte, potpisivanja kôda i e-potpisa subjekt certifikata je obično osoba ili organizacija. Međutim, kod upotrebe TLS-a predmet certifikata je obično računalo ili neki drugi uređaj, iako TLS certifikati mogu identificirati organizacije ili pojedince uz njihovu osnovnu ulogu u identificiranju uređaja. U tipičnoj shemi infrastrukture javnog ključa engl. *public-key infrastructure* (**PKI**), izdavatelj certifikata je certifikacijsko tijelo (***CA***), a to je obično tvrtka koja naplaćuje klijentima izdavanje certifikata za njih. Nasuprot tome, u shemi takozvane mreže povjerenja (engl. *web of trust*), pojedinci izravno potpisuju ključeve jedni drugima, u formatu koji obavlja sličnu funkciju kao i certifikat javnog ključa. Najčešći format za certifikate javnog ključa definiran je u **X.509**.

## *Kriptografija upotrebom javnog (i privatnog) ključa*

Kriptografija s javnim (i pripadajućim privatnim) ključem odnosno ***asimetrična kriptografija*** je kriptografski sustav koji koristi parove ključeva. Svaki par se sastoji od javnog ključa (koji može biti poznat drugima) i privatnog ključa (koji je skriven to jest ne smije biti poznat nikome osim vlasniku). Generiranje takvih parova ključeva ovisi o kriptografskim algoritmima koji se temelje na matematičkim funkcijama koje se nazivaju jednosmjerne funkcije. Učinkovita sigurnost zahtijeva čuvanje privatnog ključa privatnim; dok se javni ključ može otvoreno distribuirati bez ugrožavanja sigurnosti.
U takvom sustavu, svaka osoba može kriptirati poruku koristeći javni ključ namijenjenog primatelja, ali ta se kriptirana poruka može dekriptirati (dešifrirati) samo privatnim ključem primatelja. To omogućuje, na primjeru TLS-a, poslužiteljskom programu da generira kriptografski ključ namijenjen odgovarajućoj kriptografiji sa simetričnim ključem, a zatim da koristi klijentov javno dijeljeni ključ za kriptiranje tog novo generiranog simetričnog ključa. Poslužitelj tada može poslati ovaj kriptirani (šifrirani) simetrični ključ preko nesigurnog kanala klijentu; a samo ga klijent može dešifrirati koristeći klijentov privatni ključ, koji se uparuje s javnim ključem koji poslužitelj koristi za kriptiranje poruke. Budući da i klijent i poslužitelj imaju isti simetrični ključ, oni mogu sigurno koristiti kriptiranje simetričnog ključa (koje je u pravilu mnogo brže) za komunikaciju preko inače nesigurnih komunikacijskih kanala. Ovakav način rada ima prednost u tome što se ne mora ručno, unaprijed, dijeliti simetrične ključeve (što predstavlja novi sigurnosni problem) dok daje veću propusnost podataka pri kriptografiji sa simetričnim ključem.
U usporedbi sa ***simetričnom*** enkripcijom, ***asimetrična*** enkripcija je sporija od svake dobre simetrične enkripcije, prespora za mnoge svrhe. Današnji kripto sustavi (kao što su primjerice ***TLS*** i ***Secure Shell***) koriste i *simetričnu* i *asimetričnu* enkripciju, često korištenjem *asimetrične* enkripcije za sigurnu razmjenu tajnog ključa koji se zatim koristi za *simetrično* kriptiranje.
Primjeri *asimetričnih* algoritama za kriptiranje su: ***Difie Hellman (DH)***, ***DSA***, ***ECDSA***, ***ECDH***, ***EdDSA***, ***RSA*** i drugi.

# Simetrično kriptiranje

Algoritmi za *simetrično* kriptiranje su algoritmi za kriptografiju koji koriste iste kriptografske ključeve i za enkripciju otvorenog teksta i za dešifriranje kriptiranog (šifriranog) teksta. Pri tome ključevi mogu biti identični, ili eventualno može postojati jednostavna transformacija između dva ključa. Ključevi, u praksi, predstavljaju zajedničku tajnu između dvije ili više strana koja se može koristiti za održavanje privatne (zaštićene) informacijske veze. Zahtjev da obje strane imaju pristup tajnom ključu jedan je od glavnih nedostataka kriptiranja sa *simetričnim* ključem, u usporedbi sa kriptiranjem s javnim ključem (također poznatom kao enkripcija *asimetričnim* ključem). Međutim, algoritmi kriptiranja sa *simetričnim* ključem obično su bolji za masovno kriptiranje. Imaju manju veličinu ključa, što znači manje prostora za pohranu i brži prijenos. Zbog toga se enkripcija *asimetričnim* ključem često koristi za razmjenu tajnog ključa za enkripciju sa *simetričnim* ključem.
Primjeri simetričnih algoritama za kriptiranje su: ***Twofish***, ***Serpent***, ***AES***, ***Camellia***, ***Salsa20***, ***ChaCha20***, ***Blowfish***, ***RC4***, ***DES***, ***3DES*** i drugi. Međutim kriptiranje poruke ne jamči da će ona ostati nepromijenjena dok je kriptirana (šifrirana).

Stoga se često kriptiranom tekstu dodaje kôd za provjeru autentičnosti (tzv. **MAC**) poruke kako bi se osiguralo da će primatelj zabilježiti promjene u kriptiranom tekstu. Kodovi za provjeru autentičnosti poruke mogu se konstruirati iz AEAD mehanizma (npr. ***AES-GCM***). Naime, u kriptografiji, kod za provjeru autentičnosti poruke [*message authentication code*] (**MAC**), čini kratka informacija (podatak) koja se koristi za provjeru autentičnosti poruke. Drugim riječima, za potvrdu da je poruka stigla od navedenog pošiljatelja (njena autentičnost) i da nije promijenjena. ***MAC*** vrijednost štiti integritet podataka poruke, kao i njezinu autentičnost, dopuštajući verifikatorima, koji također posjeduju tajni ključ, da otkriju bilo kakve promjene u sadržaju poruke. Za tu namjenu se koriste neki od posebnih algoritama, poput: ***HMAC***, ***OMAC***, ***CCM***, ***GCM, Poly1305*** i drugih.

# TLS (i drugi) digitalni certifikati

U kriptografiji, certifikat javnog ključa (engl. **public key certificate**), također poznat kao digitalni certifikat ili certifikat identiteta, je elektronički dokument koji se koristi za dokazivanje valjanosti javnog ključa.
Certifikat uključuje podatke o ključu, podatke o identitetu njegovog vlasnika (koji se naziva subjekt) i digitalni potpis entiteta koji je ovjerio sadržaj certifikata (koji se naziva izdavatelj). Ako je potpis valjan, a softver koji ispituje (provjerava) certifikat vjeruje izdavatelju, tada može koristiti taj ključ za sigurnu komunikaciju sa subjektom certifikata. U sustavima enkripcije e-pošte, potpisivanja to jest takozvanog e-potpisa, subjekt certifikata je obično osoba ili organizacija. Međutim, kod upotreba TLS-a, predmet certifikata je obično računalo ili drugi uređaj, iako TLS certifikati mogu identificirati organizacije ili pojedince uz njihovu osnovnu ulogu u identificiranju uređaja. TLS, koji se ponekad naziva i starijim imenom *Secure Sockets Layer* (SSL), poznat je po tome što je dio ***HTTPS*** protokola za sigurno pregledavanje weba.
U tipičnoj shemi infrastrukture javnog ključa (engl. *Public Key Infrastructure*) (**PKI**), izdavatelj certifikata je certifikacijsko tijelo (engl. *Certificate Authority*) (**CA**), obično tvrtka koja naplaćuje klijentima izdavanje certifikata za njih. Nasuprot tome, moguće je da pojedinci izravno potpisuju ključeve jedni drugima, u formatu koji obavlja sličnu funkciju kao i certifikat javnog ključa. Najčešći format za certifikate javnog ključa definiran je u standardu **X.509**. Budući da je ***X.509*** vrlo općenit, format je dodatno ograničen profilima definiranim za određene slučajeve upotrebe, kao što je Infrastruktura javnog ključa (X.509) kako je definirano u **RFC 5280**.

***Postoji nekoliko vrsta certifikata:***

1. **TLS/SSL poslužiteljski certifikat**, koji osigurava da je komunikacija između klijentskog računala i poslužitelja sigurna. Protokol zahtijeva od poslužitelja da predstavi digitalni certifikat, koji dokazuje da:
   a. Predmet certifikata odgovara imenu računala (*hosta*) na koje se klijent pokušava povezati.
   b. Pouzdano tijelo za izdavanje certifikata (**CA**) je potpisalo je certifikat odnosno da je certifikat pouzdan i valjan.



Polje **Subject** certifikata mora identificirati primarni naziv računala i njegove domene (**FQDN**) kao takozvani zajednički naziv (engl. *Common Name*) odnosno **CN**. Dodatno certifikat može biti valjan i za domene i njihove pod domene. Takvi se certifikati obično nazivaju certifikati alternativnog imena subjekta (engl. *Subject Alternative Name*) odnosno ***SAN*** ili certifikati objedinjene komunikacije (engl. *Unified Communications Certificates*) – ***UCC***. Ovakvi certifikati sadrže polje alternativnog naziva subjekta (**SAN**), iako ih mnogi **CA**-ovi također stavljaju u polje *Subject Common Name* (**CN**) radi kompatibilnosti unatrag. **Ipak, u novijim implementacijama preferira se uporaba SAN imena, radije nego CN imena.**

Nakon što je provjera valjanosti certifikacijskog puta uspješna, klijent može uspostaviti šifriranu vezu s poslužiteljem. Poslužitelji izloženi prema Internetu, kao što su javni web-poslužitelji, moraju dobiti svoje certifikate od provjerenog, javnog tijela za izdavanje certifikata (***CA***).

# Priča o tijelu za izdavanje certifikata (Certificate authority - CA) i certifikatima

U kriptografiji, certifikacijsko tijelo odnosno autoritet (**CA**) je entitet koji pohranjuje, potpisuje i izdaje digitalne certifikate. Digitalni certifikat potvrđuje vlasništvo nad javnim ključem od strane imenovanog subjekta certifikata. To omogućuje drugima (pouzdanim stranama) da se oslone na potpise ili na tvrdnje o privatnom ključu koji odgovara certificiranom javnom ključu. **CA** djeluje kao treća strana od povjerenja – kojoj vjeruje i subjekt (vlasnik) certifikata i strana koja se oslanja na certifikat. Format ovih certifikata određen je standardom **X.509** (ili **EMV**).
Jedna osobito uobičajena upotreba za tijela za izdavanje certifikata (CA) je potpisivanje certifikata koji se koriste u HTTPS-u, protokolu koji nadograđuje HTTP protokol dodajući mu sigurnosni aspekt rada odnosno osiguravajući pregledavanje *World Wide Weba*. Druga uobičajena upotreba je izdavanje osobnih iskaznica od strane nacionalnih vlada za korištenje u dokumentima s elektroničkim potpisivanjem.

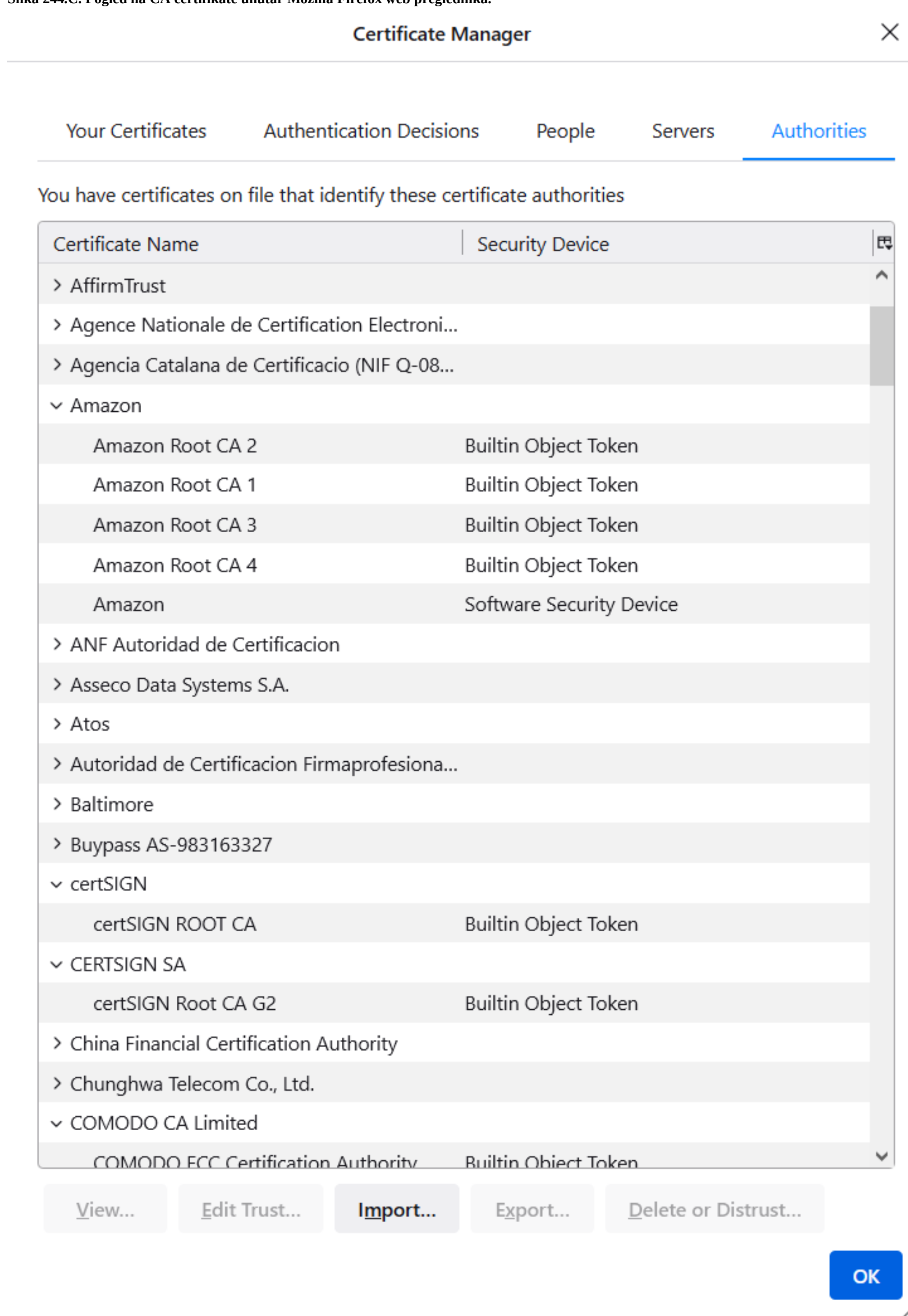
Klijent koristi ***CA*** certifikat za provjeru autentičnosti ***CA*** potpisa na certifikatu poslužitelja, kao dio autorizacije prije pokretanja sigurne veze. Obično klijentski softver, na primjer, web preglednici, uključuju skup pouzdanih tijela (organizacija ili tvrtki) za izdavanje certifikata (**CA**). To ima smisla jer korisnici moraju vjerovati svom klijentskom softveru. To znači da tijekom instalacije vašeg web preglednika, s njim dolazi i cijeli niz korijenskih (*root*) CA certifikata. Navedenih korijenskih (*root*) CA certifikacijskih tijela u načelu postoje dvije kategorije:

- Komercijalni poput: **IdenTrust**, **DigiCert**, **Sectigo**, **GoDaddy, Entrust** i drugih.
- Javni (pr. Let's Encrypt), ali i oni unutar velikih tvrtki (pr. Amazon Web Services [*Amazon Trust Services*], Cloudflare, Google Cloud Platform [*Google Trust Services LLC*] i drugi).

U svakom slučaju ***root CA*** certifikata koje imate već unesene u primjerice vaš web preglednik postoji sigurno stotinjak. Primjerice od 24. kolovoza 2020. u web pregledniku *Mozilla Firefox* pouzdano je *147 root certifikata*, koji predstavljaju 52 organizacije. Na sljedećoj poveznici pogledajte sve certifikate koji su već uključeni i vaš ***Firefox*** web preglednik: https://ccadb-public.secure.force.com/mozilla/IncludedCACertificateReport

**Kako radi certifikacijsko tijelo za izdavanje certifikata (CA)?**
CA izdaje digitalne certifikate koji sadrže javni ključ i identitet vlasnika. Odgovarajući privatni ključ nije javno dostupan, ali ga krajnji korisnik (obično web poslužitelj) koji je generirao pâr ključeva (*javni+privatni*), čuva u tajnosti. Certifikat je također potvrda ili provjera valjanosti od strane CA da javni ključ sadržan u certifikatu pripada osobi, organizaciji, poslužitelju ili drugom entitetu navedenom u certifikatu. CA-ova obveza u takvim shemama je provjera vjerodajnica podnositelja zahtjeva, tako da korisnici i pouzdane strane mogu vjerovati informacijama u izdanom certifikatu.
CA za to koriste razne standarde i testove. U biti, certifikacijsko tijelo odgovorno je reći "da, ova osoba je ona za koju se izjašnjava, a mi, **CA**, to potvrđujemo". Nadalje, ako korisnik vjeruje CA i može provjeriti potpis CA, tada također može pretpostaviti da određeni javni ključ doista pripada onome tko je identificiran u certifikatu.
Naime, nakon što mi, kao vlasnici domene i web poslužitelja kreiramo svoj certifikat koji sadrži podatke o nama i naš javni ključ za kriptiranje sadržaja, možemo od nekog od certifikacijskih tijela (CA) zatražiti digitalno potpisivanje našeg certifikata. Kada CA zaprimi naš zahtjev, oni provjeravaju naš identitet i u slučaju da je sve u redu, oni u naš certifikat dodaju i svoj potpis te ga kriptiraju sa svojim privatnim ključem. Pošto je njihov potpis kriptiran njihovim privatnim ključem, sadržaj potpisa se može dekriptirati i provjeriti s njihovim javnim ključem koji je svima javno dostupan unutar njihovog ***root CA certifikata***.
Njihov ***root CA certifikat*** se već treba nalaziti unutar vašeg web preglednika (pr. *Mozilla* ***Firefox***).
Za ***root CA certifikate*** u web pregledniku ***Firefox***, odite na postavke preglednika:
(**Settings**) —> *Privacy & Security* → *Certificates* → ***View Certificates*** ⇒

**Slika 244.C. Pogled na CA certifikate unutar Mozilla Firefox web preglednika.**

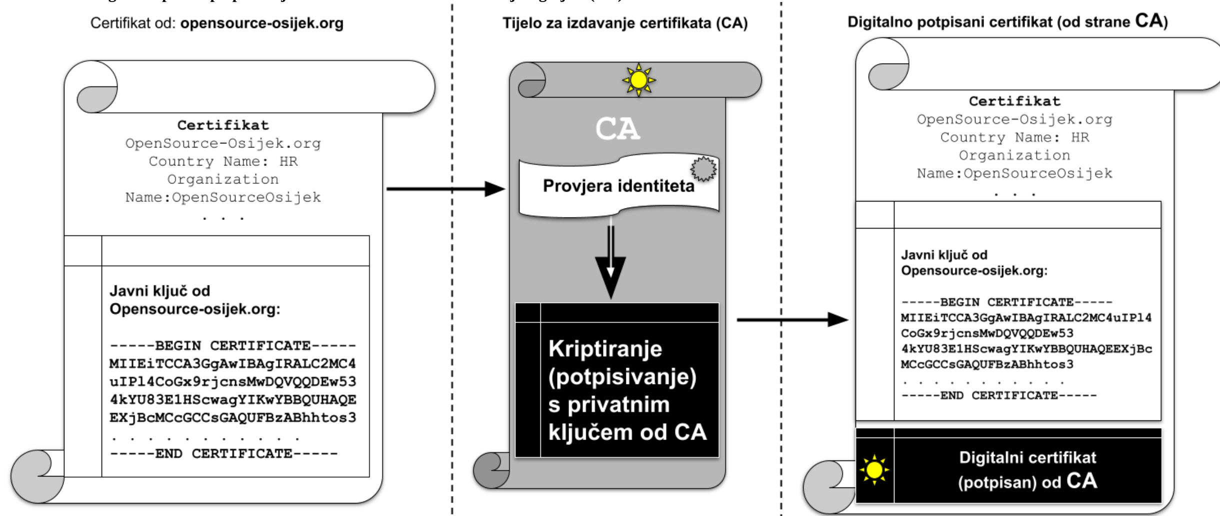


Unutar svake **CA** certifikacijske tvrtke su navedeni njihovi ***root CA*** certifikati.
Za probu, unutra *Firefox* web preglednika odaberite jedan od njih te odaberite "***View***".

Na kraju priče s potpisivanjem, ovakva poruka se sada može smatrati potpisanom, jer s provjerenim javnim ključem svatko može biti siguran da je poruka odnosno njihov digitalni potpis autentičan to jest da dolazi s njihove strane. To na kraju znači i da je naš certifikat vjerodostojan.

Pogledajte ovaj proces na slici **244.D.**

**Slika 244.D. Pogled na proces potpisivanja certifikata od strane certifikacijskog tijela (CA).**




## Vlastito potpisani (engl. *self signed*) certifikati

Prvo ćemo stvoriti privatni ključ. Privatni ključ važan je za enkripciju i najvažnija je komponenta našeg certifikata. Kreirat ćemo 2048-bitni **RSA** privatni ključ zaštićen lozinkom (`domain.key`) s naredbom **openssl**:

```
openssl genrsa -des3 -out domain.key 2048
```

Nakon toga potrebno je upisati lozinku za kreiranje ključa, a potom ćemo dobiti **RSA** ključ u datoteci: `domain.key`.
Ako želimo da naš certifikat bude potpisan, potreban nam je zahtjev za potpisivanje certifikata odnosno takozvani *certificate signing request* (***CSR***). ***CSR*** uključuje javni ključ i neke dodatne informacije kao što su organizacija/tvrtka, država i drugo.
Sada kreirajmo ***CSR*** (`domain.csr`) iz našeg postojećeg privatnog ključa:

```
openssl req -key domain.key -new -out domain.csr
```

Zatim ćemo morati potvrditi lozinku za **RSA** ključ (`domain.key`):

```
Enter pass phrase for domain.key:
```

I sada je potrebno popuniti podatke o certifikatu, poput države, organizacije, domene, adrese elektroničke pošte i slično:

```
Country Name (2 letter code) [XX]:HR
State or Province Name (full name) []:Osijek
Locality Name (eg, city) [Default City]:Osijek
Organization Name (eg, company) [Default Company Ltd]:Open Source Osijek
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:opensource-osijek.org
Email Address []:e-mail.adresa@gmail.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Osobito pripazite na to da kod popunjavanja polja `Common Name` upišete svoju domenu (***FQDN***).
Polja "*A challenge password*" i "*An optional company name*" mogu ostati prazna.

**I konačno slijedi (samo)potpisivanje certifikata**

Vlastito (samo) potpisani certifikat je certifikat koji je potpisan vlastitim privatnim ključem. Može se koristiti za šifriranje podataka jednako kao i certifikati potpisani od strane ***CA***, ali će se našim korisnicima prikazati upozorenje koje kaže da certifikat nije pouzdan. Kreirajmo samo potpisani certifikat (`domain.crt`) s našim postojećim privatnim ključem (`domain.key`) i ***CSR***-om (`domain.csr`):

```
openssl x509 -signkey domain.key -in domain.csr -req -days 365 -out domain.crt
Signature ok
subject=C = HR, ST = Osijek, L = Osijek, O = Open Source Osijek, CN = opensource-
osijek.org, emailAddress = e-mail.adresa@gmail.com
Getting Private key
```

I sada još trebamo upisati lozinku koju smo koristili za privatni ključ, pa ćemo dobiti poruku poput:

```
Enter pass phrase for domain.key:
```

Nakon što upišemo lozinku, kreirat će nam se certifikat (**domain.crt**).

Opcije koju smo koristili (**-days 365**) definira da će ovaj cerfifikat biti validan (ispravan) 365 dana.

Novo kreirani certifikat (koji je u ***PEM*** formatu) možemo pregledati sa sljedećom naredbom:

```
openssl x509 -in domain.crt -text -noout
```

## Vlastito potpisani (engl. *self signed*) certifikati i vlastiti CA

U određenim scenarijima primjene, moguće je da mi budemo i vlastito tijelo za izdavanje certifikata (***CA***) stvaranjem samopotpisanog korijenskog **CA** certifikata (tzv. ***root CA***). Sve što je potom potrebno je instalirati ga kao pouzdani certifikat u web preglednik(e) s kojim ćemo koristiti naš domenski certifikat.

**Za to je potrebno kreirati vlastito potpisani takozvani root CA certifikat**

Kako bi skratili postupak (od prethodnog primjera) u jednom koraku ćemo kreirati privatni **RSA** ključ (`rootCA.key`) i *root **CA** certifikat* (`rootCA.crt`):

```
openssl req -x509 -sha256 -days 1825 -newkey rsa:2048 -keyout rootCA.key -out
rootCA.crt
```

U sljedećem trenutku ćemo morati upisati i potvrditi lozinku za **RSA** ključ:

```
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

A zatim trebamo unijeti podatke o certifikatu; poput države, organizacije, domene, adrese elektroničke pošte i slično::

```
Country Name (2 letter code) [XX]:HR
State or Province Name (full name) []:Osijek
Locality Name (eg, city) [Default City]:Osijek
Organization Name (eg, company) [Default Company Ltd]:Open Source Osijek
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:opensource-osijek.org
Email Address []:e-mail.adresa@gmail.com
```

Pripazite da tijekom popunjavanja polja `Common Name` upišete svoju domenu (**FQDN**) – *Fully Qualified Domain Name*.



Polje ***Common Name*** (**CN**) specificira entitet odnosno uobičajeno domenski naziv entiteta, iako se u novijim implementacijama može koristiti polje ***Subject Alternative Name*** (**SAN**) umjesto njega. Međutim zbog kompatibilnosti unatrag preporuča se primarna upotreba **CN** te **SAN** kao dodatka. Primjerice za **mail.google.com** imamo definirana i **CN** i **SAN** polja:

```
Subject Name → Common Name (CN): mail.google.com
Subject Alt Names (SAN)→ DNS Name: mail.google.com, DNS Name: inbox.google.com
```

I tek sada dolazimo do potpisivanja našeg **CSR** certifikata (`domain.csr`) koji smo kreirali prije, s našim ***root CA*** certifikatom, jer smo mi u ovom slučaju samo proglašeni **CA**. U međukoraku, pošto smo mi sada praktično ***CA***, moramo za našu domenu kreirati posebnu datoteku (`domain.ext`), sa sljedećim sadržajem:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
subjectAltName = @alt_names
[alt_names]
DNS.1 = opensource-osijek.org
```

Pod `DNS.1` se upisuje FQDN ime naše domene!

Sada možemo potpisati naš **CSR** (`domain.csr`) s korijenskim **CA** certifikatom i njegovim privatnim ključem:

```
openssl x509 -req -CA rootCA.crt -CAkey rootCA.key -in domain.csr -out domain.crt -
days 365 -CAcreateserial -extfile domain.ext
```

Nakon toga moramo potvrditi lozinku za privatni ključ (`rootCA.key`):

```
Enter pass phrase for rootCA.key:
```

Kao rezultat toga, CA-potpisan certifikat bit će u datoteci `domain.crt`.

Naš ***CA*** potpisani certifikat, sada možemo pregledati:

```
openssl x509 -text -noout -in domain.crt
```

Naš certifikat (domain.crt) je ***X.509*** certifikat koji je ***ASCII PEM*** kodiran. Prema potrebi možemo koristiti OpenSSL da ga pretvorimo u druge formate za višenamjensku upotrebu.

**Primjeri konverzije formata certifikata:**

Konverzija iz ***PEM*** formata u ***DER*** format:

```
openssl x509 -in domain.crt -outform der -out domain.der
```

Konverzija iz ***PEM*** formata u ***PKCS12*** format:

```
openssl pkcs12 -inkey domain.key -in domain.crt -export -out domain.pfx
```

***PKCS12*** format koristi primjerice *Microsoft IIS* poslužitelj.

**PKCS**
U kriptografiji, ***PKCS*** je skraćenica za "standarde kriptografije javnog ključa" (engl. *Public Key Cryptography Standards*). Riječ je o skupini kriptografskih standarda s javnim ključem koje je osmislila i objavila tvrtka *RSA Security LLC*, početkom ranih 1990-ih. Premda on nije industrijski standard (jer je navedena tvrtka zadržala kontrolu nad njima), neki od standarda posljednjih godina prešli su u proces "praćenja standarda" relevantnih organizacija za standardizaciju kao što su ***IETF*** i ***PKIX*** koji sada rade na njima.

**PKCS#12**
*PKCS #12* definira format arhivske datoteke za pohranjivanje mnogih kriptografskih objekata unutar jedne datoteke.
On se obično koristi za spajanje privatnog ključa s njegovim *X.509* certifikatom ili za spajanje svih članova lanca povjerenja. Datoteka PKCS #12 može biti kriptirana i potpisana. Spremnici za unutarnju pohranu, nazvani "*SafeBags*", također mogu biti kriptirani i potpisani. Nekoliko *SafeBagova* unaprijed je definirano za pohranu certifikata, privatnih ključeva i ***CRL***-ova. Dostupan je još jedan ***SafeBag*** za pohranu svih drugih podataka po izboru pojedinca. **PKCS #12** jedan je od standarda iz obitelji standarda pod već navedenim standardima za kriptografiju javnog ključa (PKCS).
Ekstenzija naziva datoteke za ***PKCS #12*** datoteke je `.p12` ili `.pfx`.

Ove datoteke se mogu kreirati i čitati pomoću *OpenSSL* naredbe:

```
openssl pkcs12
```

**2. TLS/SSL klijentski certifikati** provjeravaju autentičnost klijenta koji se povezuje na TLS uslugu, na primjer za pružanje kontrole pristupa. Budući da većina usluga pruža pristup pojedincima, a ne uređajima, većina klijentskih certifikata sadrži adresu e-pošte ili osobno ime klijenta, a ne ime računala (hosta). Osim toga, certifikacijsko tijelo koje izdaje certifikat klijenta obično je davatelj usluga na kojeg se klijent povezuje jer je davatelj taj koji treba izvršiti provjeru autentičnosti.
Iako većina web-preglednika podržava klijentske certifikate, najčešći oblik provjere autentičnosti na Internetu je par koji čine korisničko ime i pripadajuća lozinka. Klijentski certifikati češći su u virtualnim privatnim mrežama (VPN) i uslugama udaljene radne površine (engl. *Remote Desktop*), gdje se provjerava autentičnost određenog uređaja.

**3. Certifikat e-pošte**. U skladu sa ***S/MIME*** protokolom, certifikati e-pošte mogu i provjeravati integritet poruke, ali i šifrirati (kriptirati) ih. Kako bi uspostavili šifriranu komunikaciju putem e-pošte, stranke koje komuniciraju moraju unaprijed imati svoje digitalne certifikate. Svaki mora drugome poslati digitalno potpisanu e-poštu i odlučiti se za uvoz certifikata pošiljatelja. Neka javno pouzdana tijela za izdavanje certifikata (CAovi) daju certifikate e-pošte, ali češće se koristi S/MIME kada se komunicira unutar određene organizacije, a ta organizacija pokreće vlastiti CA, kojemu sudionici u tom sustavu e-pošte vjeruju.



Za **S/MIME** pogledajte poglavlje:
**4.7. Format datoteka (*MIME type*).**

Kako smo do sada mogli zaključiti, svaki certifikat sadrži više podataka, pa pogledajmo što obično sadrži jedan **X.509** certifikat:

- **Serial number** - koristi se za jedinstvenu identifikaciju certifikata unutar sustava **CA**. To se posebno koristi za praćenje informacija o opozivu certifikata.
- **Subject** - označava entitet kojem certifikat pripada: računalo/poslužitelj (*host*), pojedinac ili organizacija.
  - **CN** – *Common Name*, identificira glavni subjekt ili entitet certifikata, a uobičajeno se koristi kako bi se identificirao domenski naziv entiteta odnosno **FQDN** naziv. Proširenje može biti upotreba *Subject Alternative Name* (**SAN**) kao dodatnih entiteta: uobičajeno dodatnih domena, poddomena, tzv. *wildcard** certifikata (pr. ***.domena.com*) i drugog.
- **Issuer** - označava subjekt koji je provjerio podatke i potpisao potvrdu.
- **Not Before** - označava najranije vrijeme i datum za koji certifikat vrijedi. Obično se postavlja na nekoliko sati ili dana prije trenutka izdavanja potvrde kako bi se izbjegli problemi s vremenom.
- **Not After** - označava vrijeme i datum nakon kojih potvrda više ne vrijedi.
- **Key Usage** - važeća kriptografska upotreba javnog ključa certifikata. Uobičajene vrijednosti uključuju provjeru valjanosti digitalnog potpisa, šifriranje ključa i potpisivanje certifikata.
- **Extended Key Usage** - označavaju aplikacije u kojima se certifikat može koristiti. Uobičajene vrijednosti uključuju provjeru autentičnosti TLS poslužitelja, zaštitu e-pošte i potpisivanje koda.
- **Public Key** - označava javni ključ koji pripada subjektu certifikata.
- **Signature Algorithm** - on sadrži algoritam za izračun provjernog zbroja (*hash*) i algoritam šifriranja. Na primjer "sha256RSA" gdje je **sha256** algoritam za izračun provjernog zbroja, a **RSA** algoritam za šifriranje.
- **Signature** - na tijelu certifikata se izrađuje izračun provjernog zbroja (koristi se algoritam definiran u polju "**Signature Algorithm**"), a zatim se provjerni zbroj šifrira (koristi se algoritam šifriranja u polju "**Signature Algorithm**") privatnim ključem izdavatelja. …

**Pogledajmo kako dohvatiti certifikat s Google poslužitelja u takozvani PEM kodirani format datoteke:**

```
echo -n | openssl s_client -connect www.google.com:443 | sed -ne '/-BEGIN
CERTIFICATE-/,/-END CERTIFICATE-/p' > cert.pem
```

Nakon gornjeg niza naredbi dobivamo certifikat u datoteci: `cert.pem`, u **PEM** formatu.

## Privacy-Enhanced Mail (PEM) format

*Privacy-Enhanced Mail* (**PEM**) je format datoteke za pohranu i slanje kriptografskih ključeva, certifikata i drugih podataka, na temelju skupa IETF standarda iz 1993. godine koji definiraju "poštu s poboljšanom privatnošću".
Iako izvorni standardi nikada nisu bili široko prihvaćeni i zamijenjeni su s ***PGP***-om i S/MIME, tekstualno kodiranje koje su definirali postalo je vrlo popularno. IETF je konačno formalizirao **PEM** format u [RFC 7468](). Mnogi kriptografski standardi koriste **ASN.1** notaciju za definiranje svojih struktura podataka i *Distinguished Encoding Rules* (**DER**) za serijalizaciju tih struktura. Budući da **DER** proizvodi binarni izlaz, problematično je prenijeti takav sadržaj i datoteke preko mreže, primjerice upotrebom sustava poput elektroničke pošte (ili drugih sustava), koji podržavaju samo ASCII kodiranje.
***PEM format rješava ovaj problem kodiranjem binarnih podataka pomoću base64 načina kodiranja***.
***PEM*** također definira i jednoredna zaglavlja s oznakom za prvi (`-----BEGIN`) i zadnji redak (`-----END`), te dijela koji slijedi u prvom ili zadnjem retku, a koji definira namjenu. Tako imamo namjenu certifikata (`CERTIFICATE-----`), zahtjeva za certifikat (`CERTIFICATE REQUEST-----`), privatnog ključa (`RSA PRIVATE KEY-----` ili `PRIVATE KEY-----`) ili (`X509 CRL-----`).
Stoga primjerice kod certifikata imamo prvi redak:

```
-----BEGIN CERTIFICATE-----
... te slijedi sadržaj certifikata ...
... na kraju dolazi zadnji redak koji izgleda ovako:
-----END CERTIFICATE-----
```

***PEM*** podaci se obično pohranjuju u datoteke sa sufiksom (ekstenzijom) `.pem`, sufiksom `.cer` ili `.crt` za certifikate ili sufiksom `.key` za javne ili privatne ključeve.
Sada pogledajmo kako iz ***PEM*** kodiranog certifikata izvući nama čitljive podatke odnosno sadržaj certifikata:

```
openssl x509 -in cert.pem -text -noout
```

## Primjer dohvaćanja certifikata s poslužitelja na mreži (internetu)

Pogledajmo i drastično skraćeni ispis certifikata za **google.com**:

```
openssl s_client -connect google.com:443 | openssl x509 -noout -text

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b0:b6:30:2e:2e:20:f9:78:0a:81:b1:f6:b8:dc:9e:c3
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
        Validity
                        Not Before: May 25 10:22:17 2022 GMT
                        Not After: Aug 17 10:22:16 2022 GMT
        Subject: CN = www.google.com
        Subject Public Key Info:
                        Public Key Algorithm: id-ecPublicKey
                           Public-Key: (256 bit)
                            pub:
                    04:a6:22:74:3f:19:18:6d:bb:a5:df:2b:cb:c9:1f:
                    0e:37:cc:8c:ae:a7:3f:db:cf:ff:52:4f:4d:8d:af:
                     . . .
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
                 X509v3 Key Usage: critical
                    Digital Signature
                 X509v3 Extended Key Usage:
                    TLS Web Server Authentication
                  X509v3 Subject Alternative Name:
                    DNS:*.google.com,*.cloud.google.com, *.datacompute.google.com,…
 Signed Certificate Timestamp:
                    Version    : v1 (0x0)
                    Log ID     : 41:C8:CA:B1:DF:22:46:4A:10:C6:A1:3A:09:42:87:5E:
                                 4E:31:8B:1B:03:EB:EB:4B:C7:68:F0:90:62:96:06:F6
                    Timestamp : May 25 11:22:18.467 2022 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                 30:46:02:21:00:92:E3:3F:21:DF:CD:8B:DD:73:38:97:
                                 42:04:5F:BE:9E:9F:73:5B:42:F4:83:81:D3:34:63:BF:
. . .
```

Unutar ovog certifikata vidimo mnoge elemente, poput:

- Podatka koji od **CA (***Certificate Authority*) je potpisao certifikat (polje ***Issuer:***)
- Za koga se izdaje certifikat (polje ***Subject:*** ) odnosno ***CN*** definira entitet kojem je certifikat dodijeljen.
- Unutar ***X509v3*** ekstenzija se između ostalog definira: *Subject Alternative Name* (**SAN**) koji ima dodatne mogućnosti identifikacije subjekta certifikata izvan osnovnog *Common Name* (**CN**). Dakle **SAN** omogućava uključivanje dodatnih identifikatora, poput: alternativnih domenskih imena (**FQDN**), kao i takozvane *wildcard certifikate* (pr. `*.google.com`), IP adresa, e-mail adresa ili URI-ja.
- *Signature Algorithm* – definira koji algoritam se koristi za generiranje digitalnog potpisa u certifikatu.
- Vremena valjanosti certifikata (polje ***Validity:*** )
- Javni ključ i svi ostali podaci koje smo prethodno spominjali.
- …i mnoge druge informacije i podaci.



Konfiguracijska datoteka ***OpenSSL*** sustava je: `/etc/pki/tls/openssl.cnf`

## Detaljan pogled na TLS certifikat i pripadajuće ključeve

**CERTIFIKAT**

Ako smo vlasnici domene, certifikata i primarnog ključa, pogledajmo detaljnije naš certifikat. Pokrenimo sljedeću naredbu, s kojom ćemo iz datoteke s certifikatom: `opensource-osijek.org.crt` vidjeti dekodirane detalje o našem certifikatu.
← *sigurnosno važne informacije smo zamijenili i skratili, a ostale podatke koje smo već opisali, izostavili!*

```
openssl x509 -noout -text -in opensource-osijek.org.crt
Certificate:
     Signature Algorithm: sha256WithRSAEncryption
     Issuer: C=US, O=Let's Encrypt, CN=R3
     Subject: CN=*.opensource-osijek.org
     Subject Public Key Info:
          Public Key Algorithm: id-ecPublicKey
               Public-Key: (2048 bit)
                 Modulus:
                01:d3:dc:72:81:1c:1d:09:0c:fc:1a:3b:cd:d0:15:
                A4:9d:40:1c:d0:c7:05:e0:e5:5b:b5:51:b2:ed:bd: . . .
            publicExponent: 65537 (0x10001)
     X509v3 extensions:
          X509v3 Extended Key Usage:
               TLS Web Server Authentication, TLS Web Client Authentication
          X509v3 Subject Key Identifier:
               53:26:77:89:8E:8F:A0:BE:0F:D3:39:30:52:BB:BE:86:46:31:DF:51
          X509v3 Authority Key Identifier:
               14:2E:B3:17:B7:58:56:CB:AE:50:09:40:E6:1F:AF:9D:8B:14:C2:C6
          Authority Information Access:
               OCSP - URI:http://r3.o.lencr.org
               CA Issuers - URI:http://r3.i.lencr.org/
          X509v3 Certificate Policies:
               Policy: 2.23.140.1.2.1
```

Veći dio elemenata certifikata smo već opisali, pa ćemo nastaviti s onim detaljima o kojima nismo govorili:

- `Modulus` - *modulus* vrijednost unutar certifikata, koja mora bit identična s *modulus* vrijednosti unutar privatnog ključa pripadajućeg certifikata. *Modulus* se koristi u procesima enkripcije i dekripcije, kao i u generiranju ključeva.

Slijede neke od ***X509v3*** ekstenzija, poput:

- `X509v3 Extended Key Usage` `(EKU)` - označava namjenu TLS certifikata, a ona može biti:
  - `Server Authentication` – za identifikaciju poslužitelja (servera).
  - `Client Authentication` – za identifikaciju klijenta.
- `X509v3 Subject Key Identifier` `(SKI)` – ovo polje sadrži jedinstveni identifikator javnog ključa subjekta (entiteta kojem pripada certifikat). Koristi se za povezivanje certifikata s njegovim javnim ključem.
- `X509v3 Authority Key Identifie` `(AKI)` – ovo polje sadrži jedinstveni identifikator javnog ključa izdavatelja (***Certificate Authority*** - **CA**) koji je potpisao certifikat. Koristi se za provjeru lanca certifikata i validaciju potpisa *CA-a*.
- `X509v3 Authority Information Access` `(AIA)` – ovo polje može sadržavati informacije o lokaciji CA-a, kao što su URL adrese za preuzimanje CA certifikata i liste povlačenja certifikata (CRL).
- `X509v3 Certificate Policies` `(CP)` – ovo polje može sadržavati informacije o politikama koje su primijenjene pri izdavanju certifikata. Obično se koristi za definiranje zahtjeva za validacijom identiteta i sigurnosnim procedurama koje mora pratiti entitet kojem je izdan certifikat.

**PRIVATNI KLJUČ**

Pogledajmo privatni ključ za našu domenu, to jest pogledajmo sadržaj datoteke `opensource-osijek.org.key`:

```
-----BEGIN PRIVATE KEY-----
MIIssAIBADANBrkqhk4d9w0BAQEFAASCBKYwggSiAgEAAoIBAQDS3HGBHB0JsPss
tyueRT+DWGMxGZdyB/3Z9sI+GCSyOL/ffBfpvuXsuRmV74C6FvG+Zx1+Ya+64svD
. . .
-----END PRIVATE KEY-----
```

Važno je razumjeti da se kod TLS-a privatni ključ koristi za nekoliko stvari:

- Kada se klijent poveže na poslužitelj, poslužitelj mu šalje svoj javni TLS ključ. Klijent koristi javni ključ iz certifikata poslužitelja za kriptiranje simetričnog ključa, koji se zatim šalje poslužitelju. Zatim se simetrični ključ koristi za kriptiranje podataka koji se šalju između klijenta i poslužitelja. Ovo osigurava sigurnu komunikaciju bez potrebe za slanjem javnog ključa klijenta poslužitelju.
- Privatni TLS ključ se koristi i za izračunavanje **MAC**-a (*Message Authentication Code*) za sve podatke koji se prenose između klijenta i servera. MAC se koristi za provjeru jesu li podaci u tranzitu bili modificirani. Dakle MAC se koristi za zaštitu integriteta podataka.

Iako privatni ključ možda izgleda kao jednostavni ključ za dekodiranje (dešifriranje), on se zapravo sastoji od više dijelova. Dekodirajmo sadržaj ovog privatnog ključa (navedene datoteke), pomoću sljedeće naredbe:

```
openssl rsa -noout -text -in opensource-osijek.org.key

Private-Key: (2048 bit, 2 primes)
Modulus:
     01:d3:dc:72:81:1c:1d:09:0c:fc:1a:3b:cd:d0:15:
     a4:9d:40:1c:d0:c7:05:e0:e5:5b:b5:51:b2:ed:bd: . . .
publicExponent: 65537 (0x10001)
privateExponent:
     32:50:2f:47:64:e2:81:4d:c6:19:8c:dd:ea:e2:80: . . .
prime1:
     12:fd:43:1a:21:8b:9b:13:0d:0b:6d:a2:b2:de:ec:
     2f:62:97:d6:83:2b:80:e7:b8:c6:1b:fc:4d:7f:3d: . . .
prime2:
     12:d3:b5:fa:c6:35:57:9a:30:a5:94:38:8a:ed:e3:
     E6:40:73:4c:6d:34:10:de:86:08:bb:bc:66:40:f1: . . .
exponent1:
     01:2f:31:10:55:e0:33:5b:3c:65:37:eb:00:4d:97:
     c1:b4:5d:eb:a9:f7:e9:7f:85:11:c0:a6:21:55:e2: . . .
exponent2:
     34:87:54:dc:e2:af:c5:05:19:a0:97:3e:38:01:77:
     A5:d0:ea:a3:c2:cd:8f:01:35:44:08:38:0a:d3:4a: . . .
coefficient:
     1a:03:94:33:ed:69:79:22:b9:a6:61:37:57:98:47:
     75:59:14:52:c0:4a:25:7f:10:4c:89:b1:10:c0:c5: . . .
```

Sada, kada smo dekodirali privatni ključ i dobili njegove sastavnice, pogledajmo što one znače:

`Modulus` - *modulus* je jedan od glavnih sastavnih dijelova RSA kriptografskog ključa. To je veliki broj koji se dobiva množenjem dva velika prosta (*prime*) broja: ***prim1*** i ***prim 2***. *Modulus* se koristi u procesima enkripcije i dekripcije, kao i u generiranju ključeva. Važno je razumjeti da ista vrijednost *modulusa* mora biti unutar certifikata i unutar privatnog ključa, što moramo moći provjeriti usporedbom našeg certifikata s privatnim ključem. Prvo po pitanju *modulusa* provjerimo certifikat:

```
openssl x509 -noout -modulus -in opensource-osijek.org.crt
01:d3:dc:72:81:1c:1d:09:0c:fc:1a:3b:cd:d0:15:
A4:9d:40:1c:d0:c7:05:e0:e5:5b:b5:51:b2:ed:bd:
. . .
```

A potom po pitanju *modulusa* provjerimo i privatni ključ:

```
openssl rsa -noout -modulus -in opensource-osijek.org.key
01:d3:dc:72:81:1c:1d:09:0c:fc:1a:3b:cd:d0:15:
A4:9d:40:1c:d0:c7:05:e0:e5:5b:b5:51:b2:ed:bd:
. . .
```

← Vidimo da se *modulus* vrijednosti certifikata i njegovog pripadajućeg primarnog ključa u potpunosti podudaraju, što ukazuje na to da su oni međusobno povezani.

Nastavimo s pregledom drugih elemenata privatnog ključa:

- `PublicExponent` – javni eksponent je drugi dio RSA javnog ključa. To je broj koji se koristi zajedno s *modulusom* za enkripciju podataka. Javni eksponent je obično mali broj, često je 65537 (0x10001), zbog svojih matematičkih svojstava.
- `PrivateExponent` – privatni eksponent je ključni dio RSA privatnog ključa. On se koristi za dekripciju podataka koji su kriptirani (šifrirani) s javnim ključem. Ovaj eksponent dobiva se kao rezultat procesa generiranja ključeva, a najčešće pomoću **CRT** algoritma (zbog brzine rada) i ima važnu ulogu u RSA algoritmu.
- `Prime1` – prvi od dva *prosta** (*prime*) broja koji su korišteni u procesu generiranja RSA ključeva. Ovi *prosti* brojevi se odabiru pažljivo kako bi se osigurala sigurnost kriptografskog sustava.
- `Prime2` – drugi od dva *prosta** (*prime*) broja koji su korišteni u procesu generiranja RSA ključeva.

- `Exponent1` – prvi faktor u kineskom teoremu o ostacima **CRT** (engl. *Chinese Remainder Theorem*) koji se koristi u RSA algoritmu za optimizaciju procesa dekripcije. Kineski teorem o ostacima (**CRT**) je matematički teorem koji omogućuje dekriptiranje poruka brže nego korištenjem samo osnovnog **RSA** algoritma. Naime **CRT** omogućuje izvođenje paralelne dekripcije, što rezultira bržim procesom dekripcije. To omogućuje učinkovitije rukovanje velikim porukama i smanjenjem vremena potrebnog za dekripciju.
- `Exponent2` – drugi faktor u kineskom teoremu o ostacima (**CRT**) koji se koristi u RSA algoritmu za optimizaciju procesa dekripcije. **Exponent1** i **Exponent2**, koriste se zajedno s ostalim informacijama poput *prostih* brojeva (*prime1* i *prime2*), *modulusa* i *privatnog eksponenta*, kako bi se ubrzao proces dekripcije poruka u RSA kriptografiji pomoću **CRT** algoritma.
- `Coefficient` - koeficijent koji se koristi za izračun teorema kineskog ostatka (CRT) za ubrzanje dekripcije.

***Prosti brojevi** (engl. *prime numbers*) su svi prirodni brojevi veći od 1 koji su djeljivi samo s 1 i sa samim sobom. Oni se intenzivno koriste u kriptografiji.

**Izvori informacija:**
**(1248),(1249),(1250),(1251),(1252),(1253),(1254),(1255),(1256),(1257),(1258),(1259),(1260),(1261),(1262),(1263),(1264), (1265),(1276), RFC 5246, RFC 5280, RFC 7468, RFC 8446, `man openssl`,`man openssl-pkcs12`.**

---

### 26.4.4.1. Konfiguracija HTTPS servisa (Apache)

Nakon što ste instalirali i konfigurirali ***Apache*** web poslužitelj, kako je opisano u prethodnom poglavlju, potrebno je instalirati i programske module koji su mu potrebni za TLS/SSL funkcionalnosti:

```
yum -y install mod_ssl
```

Sada krećemo s kreiranjem certifikata i ključeva, s time da prvo kreiramo privatni ključ:

```
mkdir /root/tmp
cd /root/tmp/
openssl genrsa -out ca.key 2048
```

Potom kreirajmo ***CSR***:

```
openssl req -new -key ca.key -out ca.csr
```

Zatim moramo upisati potrebne podatke:

```
Country Name (2 letter code) [XX]:HR
State or Province Name (full name) []:Osijek
Locality Name (eg, city) [Default City]:Osijek
Organization Name (eg, company) [Default Company Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:OpenSource-osijek.org
Email Address []:nas.mail@gmail.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Završavamo s kreiranjem vlastito potpisanog ključa (uz sve napomene koje smo spomenuli u prethodnoj cjelini)

```
openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
```

Sada kopirajmo sve kreirano u definirana odredišta unutar kojih se očekuju datoteke (certifikati i ključevi) koje smo kreirali:

```
cp ca.crt /etc/pki/tls/certs
cp ca.key /etc/pki/tls/private/ca.key
cp ca.csr /etc/pki/tls/private/ca.csr
```

Zatim moramo otvoriti i urediti konfiguracijsku datoteku: `/etc/httpd/conf.d/ssl.conf` te pronaći sljedeće retke:

```
SSLCertificateFile
SSLCertificateKeyFile
```

Njih trebamo promijeniti ovako (dakle promijeniti ih da pokazuju na datoteke koje smo kreirali):

```
SSLCertificateFile /etc/pki/tls/certs/ca.crt
SSLCertificateKeyFile /etc/pki/tls/private/ca.key
```

Nakon što smo snimili ove promjene, potrebno je restartati Apache (***httpd***) servis:

```
systemctl restart httpd
```

Nakon što se Apache servis restarta, on će slušati na starom HTTP portu 80, ali i na novom HTTPS portu 443. Provjerite i ***vatrozid (iptables*** ili ***firewalld)*** i ***SELinux*** servis, kako oni ne bi blokirali HTTPS promet.

Međutim, certifikati koje smo kreirali nisu autorizirani kod nekog od ***CA***, pa će se kod svakog spajanja web preglednika na našu web stranicu pojaviti upozorenje kako stranica nije sigurna. Da bi to riješili sljedeći korak bi bio korištenje nekog od ***CAova*** odnosno potpisivanje našeg certifikata.
Izbor je upotreba komercijalnih ***CA****ova* ili primjerice upotreba ***Letsencrypta*** koji je besplatan.
Napomena: za ovaj proces morate imati javno registriranu domenu, a domena mora biti u vašem vlasništvu, imati **DNS A** zapis i imati IP adresu koja se javno može usmjeravati; u suprotnom, proces neće uspjeti.

Ovdje nećemo objašnjavati cijelu proceduru, ali ona se svodi na nekoliko inicijalnih koraka:

```
yum -y install epel-release
yum -y install certbot python3-certbot-apache
```

Tada je potrebno pokrenuti proceduru potpisivanja našeg certifikata od strane ***Letsencrypta*** koji je **CA** (*Certificate Authority*).

To se svodi na sljedeću naredbu (pr. za domenu ***opensource-osijek.org***)

```
certbot --apache -d opensource-osijek.org
```

Nakon što ova procedura završi (što može potrajanti nekoliko minuta jer CA radi provjere autentičnosti), dobivamo potrebne certifikate od strane ***Letsencrypta***. Za detalje oko nastavka procedure te procedure i mehanizama osvježavanja ovih certifikata (jer traju samo 90 dana), proučite upute koje su dostupne na internetu

**Povećanje razine sigurnosti Apache web poslužitelja**
Kod otvaranja nove veze prema Web poslužitelju, odgovor Web poslužitelja (poruka ***TLS Server Hello***) sadrži koje minimalne inačice TLS ili SSL protokola podržava odnosno dozvoljava zbog sigurnosnih razloga. Upravo na to možemo utjecati, jer se lista protokola koje želimo ili ne želimo dozvoliti definira u konfiguraciji Apache web poslužitelja u varijabli: **SSLProtocol**.

Otvorimo datoteku: `/etc/httpd/conf.d/ssl.conf` i pronađimo redak u kojem se nalazi ova varijabla (ne smije počinjati sa znakom **#** koji možete maknuti, ako redak počinje s njim.

Mi ćemo zabraniti upotrebu sljedećih protokola: **SSL v.3.** i **TLS v.1.0**, kao i **TLS v.1.1.** jer je poznato da su ranjivi. Dok ostale protokole dozvoljavamo (***all***). Stoga naš konfiguracijski redak sada treba izgledati ovako:

```
SSLProtocol          all -SSLv3 -TLSv1 -TLSv1.1
```

Osim protokola koje možemo zabraniti, možemo dozvoliti upotrebu samo određenih kriptografskih paketa, što znači da su svi ostali (potencijalno ranjivi) zabranjeni. Pronađimo redak koji sadrži varijablu: **SSLCipherSuite**.
Za početak ćemo odobriti samo one kriptografske pakete koji su najsigurniji; što može značiti da se neki klijenti (stariji web preglednici) neće uopće moći spojiti na naš web poslužitelj.

Pogledajmo ovu konfiguraciju (u jednom retku):

```
SSLCipherSuite       ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
```

Nakon ovih promjena potrebno je restartati *Apache* (***httpd***) servis:

```
systemctl restart httpd
```

Sada, kada se spojimo s našim klijentom (web preglednikom), odabirom na lokot uz vezu, kao što je vidljivo na slici, u slučaju otvaranja stranice: **https://www.google.com** :




Te odabirom na više informacija ("*More information*"), vidimo da se koristi ***TLS v.1.3.*** te odabrani kriptografski paket:***TLS_AES_256_GCM_SHA384***:

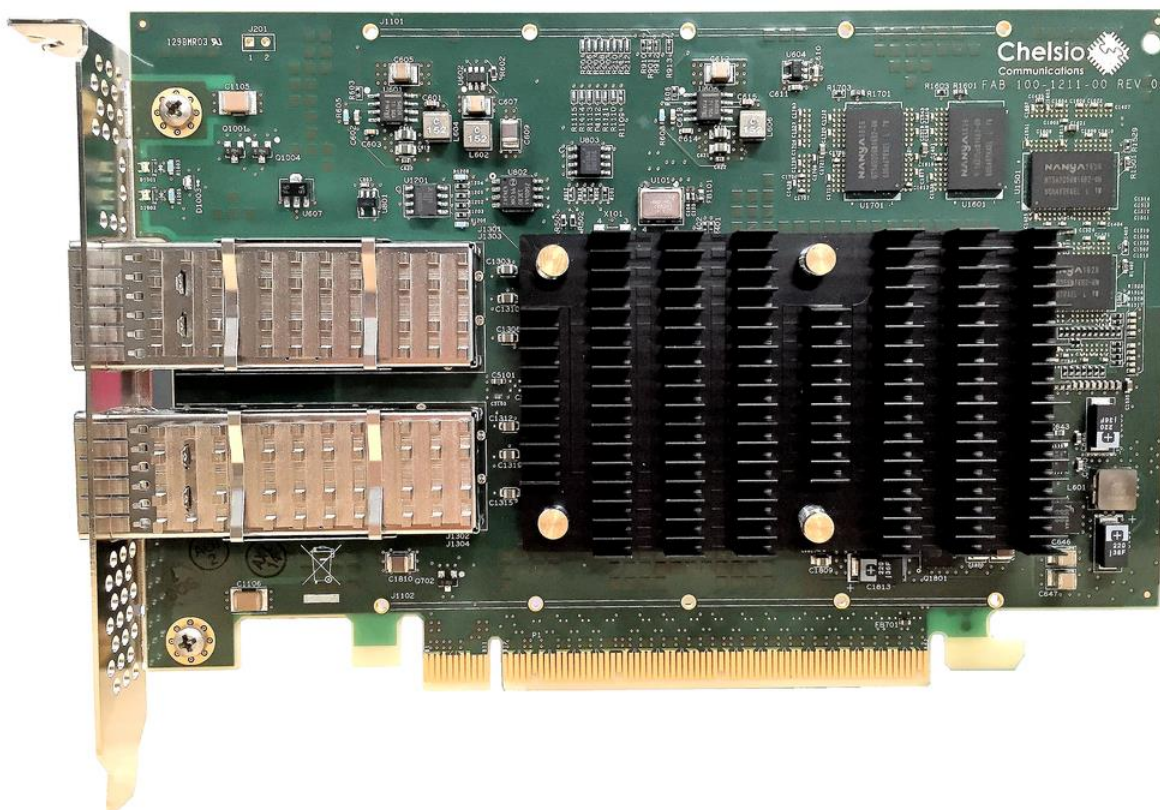
**Technical Details**
Connection Encrypted (TLS_AES_256_GCM_SHA384, 256 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.

**Izvori informacija: (1266),(1267).**

### 26.4.4.2. Upotreba hardverski ubrzanih kripto funkcija

Moderni procesori i *čipseti* imaju ugrađene komponente koje hardverski ubrzavaju određene operacije.
Primjerice jedna od takvih komponenti je zadužena za kriptografski algoritam **AES**, dok je druga zadužena za generiranje slučajnih [nasumičnih] brojeva (*rdrand*). Postoji i primjerice tehnologija koja se naziva *Intel Quick Assist* (**QAT**) koja ubrzava mnoge operacije koje su potrebne u kriptografiji: od algoritama za kriptiranje (*AES*, *ARC4*, ..) preko algoritama za izračun provjernog zbroja (*SHA-1*, *MD5*, *SHA-2*, ….) do algoritama za autentikaciju (*HMAC*, *AES-CCM*), generiranje slučajnih brojeva, algoritama za digitalno potpisivanje i razmjenu ključeva, komprimiranje i drugo. U svakom slučaju ovdje govorimo o posebno dizajniranom elektroničkom sklopu koji je integriran u procesor, *chipset* ili u ekstremnim slučajevima izveden kao zasebna samostalna kartica (obično u PCI express izvedbi). Takve specijalizirane kripto kartice obično ubrzavaju sve *SSL/TLS*, *IPSEC/VPN* operacije, pa tako sve operacije koje bi inače morao odrađivati centralni procesor, one preuzimaju na sebe. Pošto one obično imaju neku vrstu ***ASIC*** sklopova, sve navedene operacije odrađuju se ekstremno brzo. Primjerice kartica tvrtke ***Cavium*** *NITROX* (sada ***Marvell***) *Nitrox V* može osigurati propusnost od čak 100 Gbps za *SSL/TLS* ili *IPSec* mrežni promet. Drugi primjer je kartica tvrtke ***Chelsio***, *Terminator* (***T6***), kao što je primjerice kartica *T62100-CR*, na slici **244.E**.

**Slika 244.E. Pogled na kripto karticu Chelsio Terminator T6**




Ova konkretna specijalizirana kripto kartica, koja se često naziva ***SSL offloader***, nudi i drastična ubrzanja kod upotrebe ***OpenvSwitch****a* (za virtualizaciju), kao i upotrebu ***VxLAN***, ***NAT***, ***NVGRE*** i drugih protokola i tehnologija **(1287)**.



Postoje i druge kartice za posebne namjene, poput **NFP** ili **DPU**. Za detalje o njima pogledajte poglavlje: **26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP + eBPF*).**

## Hardverski ubrzan AES protokol

*Intel Advanced Encryption Standard New Instructions* (**AES-NI**) poseban je skup instrukcija za x86 procesore koji je dizajniran za ubrzanje izvršavanja *AES* algoritama. Simetrično kriptiranje temeljeno na AES algoritmu široko se koristi u raznim sigurnosnim aplikacijama i implementacijama protokola poput: ***IPSec***, ***SSL***/***TLS***, ***HTTPS***, ***SSH*** i drugih. *OpenSSL* kripto biblioteke i pripadajući programi također podržavaju kriptiranje temeljeno na *AES*-u.
Prvo provjerimo podržava li naš procesor *AES-NI* instrukcije:

```
grep -m1 -o aes /proc/cpuinfo
aes
```

Pošto ovdje imamo `aes`, to znači da je skup *AES* instrukcija unutar našeg procesora dostupan.
Međutim da bismo provjerili može li *OpenSSL* iskoristiti *AES* skupove instrukcija, možemo koristiti *OpenSSL*-ov **EVP** API. Kada se pozove **EVP** API, on može automatski otkriti prisutnost AES-NI i ubrzati izračune AES enkripcije pomoću *AES* skupova instrukcija. Tako možemo usporediti *AES* performanse sa ili bez **EVP** funkcija. Ako je AES-NI dostupan za *OpenSSL*, vidjet ćemo značajno povećanje performansi kada se koriste **EVP** funkcije.

Stoga prvo pokrenimo test bez *AES* (**EVP**) funkcija u procesoru:

```
openssl speed -elapsed aes-128-cbc
. . .
type        16 bytes    64 bytes   256 bytes   1024 bytes   8192 bytes  16384 bytes
aes-128 cbc 223706.26k  227914.56k 226772.48k  231195.31k   241615.01k   241128.79k
```

A sada ponovimo isti test pomoću **EVP** funkcija to jest upotrebom ***AES-NI*** instrukcija u procesoru**:**

```
openssl speed -elapsed -evp aes-128-cbc

type        16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes  16384 bytes
aes-128-cbc 807177.06k  1266887.72k 1349963.69k 1251893.59k  1320219.99k 1390176.94k
```

Ako usporedimo prvi i drugi test, razlika je vidljiva (u tablici):

| ***Bez AES-NI*** | ***Sa AES-NI (EVP)*** | ***Razlika (ubrzanje)*** |
|---|---|---|
| **16 bajta:** 223.706,26k | **16 bajta:** 807.177,06k | 3,61 x  (361 %) |
| **64 bajta:** 227.914,56k | **64 bajta:** 1.266.887,72k | 5,56 x (556 %) |
| **256 bajta:** 226.772,48k | **256 bajta**: 1.349.963,69k | 5,95 x (595 %) |
| **1024 bajta:** 231.195,31k | **1024 bajta:** 1.251.893,59k | 5,41 x (541 %) |
| **8192 bajta:** 241.615,01k | **8192 bajta:** 1.320.219,99k | 5,46 x (546 %) |
| **16384 bajta:** 241.128,79k | **16384 bajta:** 1.390.176,94k | 5,77 x (577 %) |

Dakle prosječna razlika, ovisna u veličini bloka, je ubrzanje od pet puta (cca. **500+ %**).

**Kernel moduli i AES-NI**

Da bi nam *AES-NI* uopće radio potrebna je podrška za njega ili već ugrađena u Linux kernel ili dostupna kao kernel modul. Za procesore tvrtke *Intel*, ako je u pitanju podrška dostupna kao kernel modul, to je kernel modul imena `aesni_intel`, koji (ako nije učitan) možemo učitati i ovako:

```
modprobe aesni_intel
```

U novijim kernelima (5.x.) moguće je da je ovaj kernel modul već integriran u kernel. To možemo vidjeti sa:

```
grep _AES /boot/config-$(uname -r)

CONFIG_CRYPTO_AES=y
CONFIG_CRYPTO_AES_NI_INTEL=y
```

Vidimo da podrška za ***AES*** postoji u kernelu (`CONFIG_CRYPTO_AES=y`) te da je u sâm kernel ugrađen i kernel modul za *Intelove* procesore (`CONFIG_CRYPTO_AES_NI_INTEL=y`), pa tada nije potrebno učitavati ovaj kernel modul jer je on tada već učitan sa sâmim kernelom. *OpenSSL* nam osim ubrzavanja *AES* operacija može nuditi i druge mogućnosti. Sada pogledajmo što je sve *OpenSSL* prepoznao:

```
openssl engine
(rdrand) Intel RDRAND engine
(aesni) Intel AES-NI engine
(dynamic) Dynamic engine loading support
```

Slijedi kratki opis:

- `rdrand` - označava da *OpenSSL* podržava upotrebu ***RDRAND*** hardverskog generatora slučajnih brojeva..
- `aesni` - označava da *OpenSSL* podržava upotrebu ***AES-NI*** hardverske kripto značajke.
- `dynamic` - označava da *OpenSSL* podržava upotrebu dinamičkih modula za kripto značajke.

Navedeni mehanizmi (*engine*) koji su dostupni u pravilu se automatski koriste unutar (*OpenSSL*) kompatibilnih aplikacija, a dodatno se mogu i zasebno navoditi.

**Pouzdani(ji) i brži hardverski generator slučajnih brojeva**

Već od treće generacije procesora tvrtke *Intel* (i procesora novijih od 2015.g. tvrtke *AMD*)  postoji podrška za brzo i sigurno generiranje slučajnih brojeva, koji se intenzivno koriste u kriptografiji. Pogledajmo kako provjeriti imamo li ovu podršku u hardveru procesora:

```
grep -m1 -o rdrand /proc/cpuinfo
rdrand
```

Vidimo da imamo hardversku funkciju generatora slučajnih brojeva (`rdrand`) unutar našeg procesora. Upravljački program za *Intel* procesor je (`intel-rng`) dok je za *AMD* procesore to (`amd-rng`). Pogledajmo kako ga učitati (za *Intel* CPU):

```
modprobe intel-rng
```

Na nekim (ili novijim kernelima, moguće je da se on već nalazi unutar kernela, kao što je ovdje slučaj:

```
grep -i CRYPTO_RNG /boot/config-$(uname -r)

CONFIG_CRYPTO_RNG=y
CONFIG_CRYPTO_RNG2=y
CONFIG_CRYPTO_RNG_DEFAULT=y
```

Sve kripto značajke s pripadajućim upravljačkim programima, mogu se vidjeti na sljedeći način:

```
cat /proc/crypto
```

Naime iako na sustavu imamo `/dev/urandom` koji također radi slično kao `/dev/random`, ali kada se dosegne donji prag entropije, on ne čeka već počinje koristiti takozvani pseudo *random number generating* formulu kako bi simulirao što "slučajniji" i što nepredvidljiviji nîz podataka. Ovaj uređaj radi višestruko brže od `/dev/random`, ali je donekle predvidiv pa se ne preporuča njegova upotreba za ekstremno sigurnosno zahtjevne primjene. Ipak postoji mogućnost uporabe posebnog hardverskog generatora slučajnih brojeva, koji je vidljiv kao posebni uređaj: `/dev/hwrng` (ako ga imate) te servisa `rngd` (dolazi u softverskom paketu: `rng-tools`). `Rng-tools` je skup uslužnih programa povezanih s generiranjem slučajnih brojeva u kernelu. Glavni program je ***rngd***, razvijen za provjeru i prijenos nasumičnih podataka s hardverskog uređaja u entropijski bazen kernela. Ovo je korisno za povećanje količine entropije u kernelu kako bi primjerice posebna datoteka `/dev/random` bila brža. Prema zadanim postavkama `/dev/random` je vrlo spor jer prikuplja entropiju samo od upravljačkih programa uređaja i drugih (sporih) izvora. **Rngd** omogućuje korištenje bržih izvora entropije, uglavnom hardverskih generatora slučajnih brojeva (pr. *RDRAND*), prisutnih u modernom hardveru odnosno već i u modernim procesorima mnogih tvrtki (*Intel, AMD, ARM*, …).
Trenutnu entropiju možete vidjeti s naredbom:

```
cat /proc/sys/kernel/random/entropy_avail
```

Sve vrijednosti veće od nekoliko tisuća (2000+) ukazuju na upotrebu hardverskog izvora entropije.
Instalirajmo softverski paket `rng-tools` u kojem ćemo dobiti potreban servis i određene naredbe:

```
yum -y install rng-tools
```

Sada ga možemo i pokrenuti te vidjeti kako se inicijalizirao:

```
systemctl start rngd
systemctl status rngd
● rngd.service - Hardware RNG Entropy Gatherer Daemon
 Loaded: loaded (/usr/lib/systemd/system/rngd.service;enabled;vendor preset: enabled)
 Active: active (running) since Tue 2022-06-25 18:15:16 CEST; 14min ago
. . .
18:15:16 rocky-HH.localdomain systemd[1]: Started Hardware RNG Entropy Gatherer Daemon.
18:15:16 rocky-HH.localdomain rngd[17284]: Disabling 7: PKCS11 Entropy generator (pkcs11)
18:15:16 rocky-HH.localdomain rngd[17284]: Disabling 5: NIST Network Entropy Beacon (nist)
18:15:16 rocky-HH.localdomain rngd[17284]: Initializing available sources
18:15:16 rocky-HH.localdomain rngd[17284]: [hwrng ]: Initialization Failed
18:15:16 rocky-HH.localdomain rngd[17284]: [rdrand]: Enabling RDSEED rng support
```

Provjerite postoji li ***hwrng*** (`/dev/hwrng`) uređaj na računalu. Ako uređaj ne postoji, očekuje se greška poput:

```
rngd: [hwrng ]: Initialization Failed
```

Za testiranje performansi njegovog rada, možemo pokrenuti sljedeće naredbe:

```
cat /dev/random | rngtest -c 1000
```

Dok ovu naredbu možemo koristiti za testiranje kreiranja niza slučajnih brojeva:

```
dd if=/dev/random of=/dev/null bs=1024 count=1 iflag=fullblock
```

Navedenu naredbu pokrenite prije pokretanja ***rngd*** servisa i zatim poslije njega, a očekujemo razliku u brzini.

## *Virtualizacija i RNG*

Unutar virtualnog računala u pravilu nemamo pristup izvoru hardverske entropije iz najmanje dva razloga:

- Prema definiciji, kao virtualni stroj, koristite slojeve apstrakcije i nemate hardverske uređaje.
- Korištenje hardverskih uređaja stvorilo bi problematične ovisnosti koje bi onemogućile bilo kakvu migraciju virtualnog računala.

Kako bi riješili ovaj problem, za QEMU/KVM je osmišljen takozvani ***virtio-rng*** uređaj. Entropija dostupna na razini *hipervizora* (fizičkog računala) tj. domaćina sada se može poslati na razinu virtualnog računala putem ovog uređaja.
Ovisno o vrsti i konfiguraciji vašeg *Hipervizora* (pr. Linux ***KVM+QEMU***) svakako je potrebno dodati ***virtio-rng*** uređaj na vaše virtualno računalo. Zatim ćete unutar tog virtualnog računala (nakon restarta) vidjet novi uređaj pod nazivom `/dev/hwrng`.
Ovaj uređaj sada može transparentno puniti `/dev/random` uređaj s visokokvalitetnim (pseudo) slučajnim brojevima.
Ostatak konfiguracije je isti kao i za fizičko računalo.

## *Virtio_crypto*

***Virtio-crypto*** je virtualni uređaj koji je podržan od strane **VIRTIO** sustava za para virtualizaciju i sastoji se od vršnog upravljačkog programa (*front-end*) i upravljačkog programa niže razine (*back-end*). Vršna komponenta pokreće *virtio-crypto* uređaji unutar virtualnog računala ili Linux kontejnera. Upravljački program niže razine mora biti podržan unutar QEMU-a, a izvršava se na fizičkom poslužitelju (*Hipervizoru*). Ako VM ili kontejnerska aplikacija izvodi operacije kriptiranja ili provjere autentičnosti pristupanjem *virtio-crypto* uređaju, *virtio-crypto* upravljački program unutar virtualnog računala će prenijeti zadatak na upravljački program niže razine na fizičkom poslužitelju (*Hipervizoru*) i poslati ga kripto resursu fizičkog poslužitelja na obradu.

*Virtio-crypto* upravljački program više razine integriran je u Linux kernel, dok je *virtio-crypto* uređaj niže razine implementiran u QEMU, a on se standardno oslanja na softverske kripto izračune. Međutim postoji i njegova implementacije upotrebom DPDK (i neke druge) koja može koristiti herdverski ubrzane funkcije (pr. ***AES-NI***, ***RDRAND*** ili ***QAT*** unutar procesora ili *chipseta*) ili upotrebu sa specijalizirane TLS/SSL kripto kartice.

## *Crypto API*

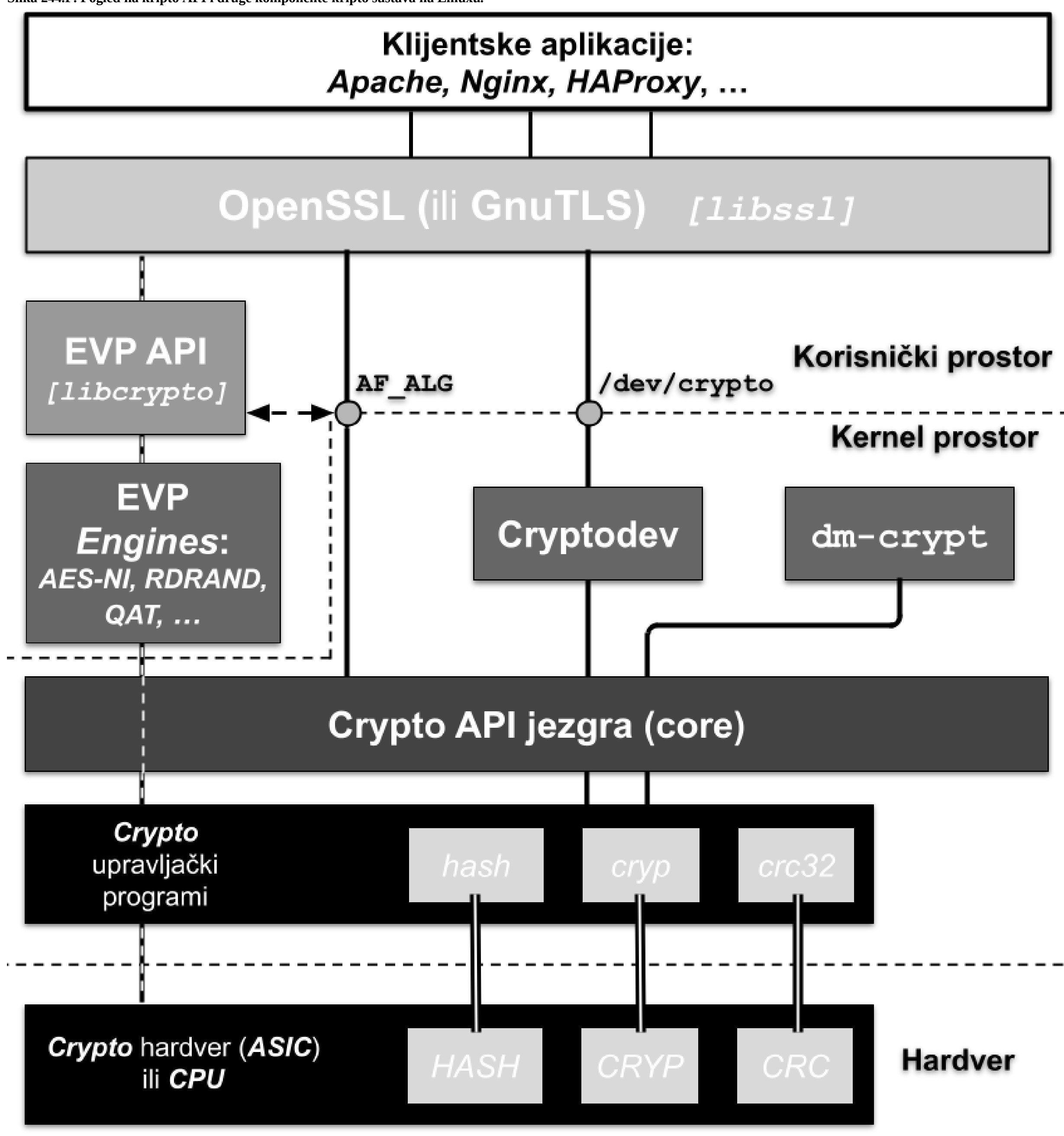
*Crypto API* je kriptografsko programsko sučelje u Linux kernelu, za različite dijelove kernela koji se bave kriptografijom, kao što su primjerice ***IPsec*** i ***dm-crypt***. Uveden je u inačici kernela 2.5.45 i od tada se proširio kako bi uključio sve popularne algoritme i funkcije koje se koriste u kriptografiji. Naime svrha ovog API-ja je pružiti brze kriptografske operacije opće namjene ostatku kernela odnosno prema aplikacijama više razine.

Iako je prvotna potreba razvoja ovakvog sustava bila vezana za *IPSec*, razvoj je proširen i za druge namjene, kao što su primjerice kriptografski datotečni sustavi odnosno datotečni sustavi koji imaju mogućnost kriptiranja sadržaja (datoteka), a koji također mogu koristiti ovu mogućnost. Kripto API pruža implementacije jedno blokovnih kriptografskih algoritama te algoritme za izračun provjernog zbroja (*hash*). Osim toga, kripto API pruža brojne "predloške" koji se mogu koristiti zajedno s navedenim algoritmima i mehanizmima. Predlošci uključuju sve vrste načina ulančavanja blokova, poput ***HMAC*** mehanizma itd. Korisnik *crypto* API sustava može izravno koristiti navedene algoritme ili ih može pozvati zajedno s predloškom za formiranje više blokovnih mehanizama za kriptiranje. Lista svih crypto upravljačkih programa (kernel modula) vidljiva je sa:

```
ls -al /lib/modules/$(uname -r)/kernel/arch/x86/crypto/
```

Dok je lista svih aktivnih kripto algoritama i mehanizama dostupnih kernelu vidljiva u posebnoj datoteci: `/proc/crypto`.

**Slika 244.F. Pogled na kripto API i druge komponente kripto sustava na Linuxu.**



## Intel Quick Assist (QAT)

***Intel QuickAssist Technology*** obično poznata kao ***QuickAssist*** ili ***QAT*** je hardverska je tehnologija koja ubrzava kriptografske i kompresijske algoritme. Tvrtka *Intel* tehnologiju *QuickAssist* prvi puta je objavila 2013. godine s pojavom Intel *Atom C2xx8 "Rangeley" procesora (QAT GEN 1.)*, ali i određenih *Xeon* procesora. *QAT* tehnologija osim što je implementirana u posebne kategorije procesora izvedena je i u nekim varijantama *chipseta* ili u varijanti zasebnih kartica koja naravno postižu još bolje performanse. Pogledajmo neke od mehanizama, tehnologija i algoritama koje ona ubrzava:

- *Simetrične* kriptografske algoritme (***AES***, ***DES***, ***3DES***, ***ARC4***).
- Wireless algoritme (***Kasumi***, ***Snow***, ***3G***).
- Algoritme za autentikaciju i izračun provjernog zbroja [*hash*] (***SHA-1***, ***MD5***, ***SHA-2*** [*SHA-224, SHA-256, SHA384, SHA-512*], ***HMAC***, ***AES-XCBC***, ***AES-CCM, AES-GCM, AES-XTS, …***)
- *Asimetrične* kriptografske i druge algoritme (***RSA***, ***DSA***, ***DH***, ***ECDSA, ECDH, Curve2519***).
- …

*QAT* tehnologija omogućuje upotrebu navedenih tehnologija, bez opterećivanja CPU-a, i k tome drastično brže jer su sve navedene značajke implementirane u hardveru (u integriranim ***ASIC*** sklopovima).
Ovu tehnologiju možemo pronaći u primjerice:

- *Atom C2xxx* procesorima (koji imaju prvu generaciju to jest takozvanu *GEN 1*).
- *Atom C3xxx i C5xxx* (koji imaju drugu generaciju to jest takozvanu *GEN 2*).
- *Atom P5000,* kao i u *Xeon D1700 ili D2700* (koji imaju treću generaciju to jest takozvanu *GEN 3*).
- *Nekim Intel® Xeon® Scalable* ili *Xeon® D* Procesorima
- *Intel chipsetima*: *C627*, *C627A, C629* i *C629A* te nekim drugima.
- *Specijaliziranim karticama, poput*: *Intel® QuickAssist Adapter*: *8960*, *8970* i drugima.

Za ***QAT*** tehnologiju pod linuxom, prvo je potrebna podrška za *QAT*, koju dobivamo s kernel modulom **`intel_qat`**:

```
modprobe intel_qat
```

Zatim je potreban upravljački program, ovisan o hardveru (procesoru, *chipsetu*, …) koji se nalazi negdje u strukturi direktorija:

```
ls -al /lib/modules/$(uname -r)/kernel/drivers/crypto/qat/
```

Zatim ovisi za koju namjenu ga želimo koristiti; primjerice trebamo li podršku za ***OpenSSL*** i druge aplikacije koje će koristiti ***OpenSSL***, tada je u načelu procedura sljedeća. Naime, ako pogledamo sliku (**244.F.**) vidljivo je kako ***OpenSSL*** podržava mogućnost upotrebe takozvanih ***EVP*** mehanizama (*EVP Engines*). Zapravo se radi o modulima koji nam daju podršku za upotrebu različitog specijaliziranog hardvera poput ***QAT*** ili nekih drugih kripto kartica. Podrška za **QAT** unutar *OpenSSLa* je konkretno dostupna preko `QAT_Engine` mehanizma.
Za detaljne upute kako instalirati *QAT* mehanizam u *OpenSSL*, posjetite stranicu: https://github.com/intel/QAT_Engine
Nakon instalacije, možete testirati je li on sada instaliran unutar ***OpenSSL*** sustava:

```
openssl engine
(rdrand) Intel RDRAND engine
(qatengine) Intel Quick Assist
(dynamic) Dynamic engine loading support
```

Potom provjerimo detalje:

```
openssl engine -t -c -v qatengine
```

Ako je sve u redu dobit ćemo nešto poput sljedećeg ispisa:

```
(qatengine) Reference implementation of QAT crypto engine(qat_hw) <qatengine version>
 [RSA, DSA, DH, AES-128-CBC-HMAC-SHA1, AES-128-CBC-HMAC-SHA256,
 AES-256-CBC-HMAC-SHA1, AES-256-CBC-HMAC-SHA256, TLS1-PRF, HKDF, X25519, X448]
    [ available ]
    ENABLE_EXTERNAL_POLLING, POLL, SET_INSTANCE_FOR_THREAD,
    GET_NUM_OP_RETRIES, SET_MAX_RETRY_COUNT, SET_INTERNAL_POLL_INTERVAL,
    GET_EXTERNAL_POLLING_FD, ENABLE_EVENT_DRIVEN_POLLING_MODE,
    GET_NUM_CRYPTO_INSTANCES, DISABLE_EVENT_DRIVEN_POLLING_MODE,
    SET_EPOLL_TIMEOUT, SET_CRYPTO_SMALL_PACKET_OFFLOAD_THRESHOLD,
    ENABLE_INLINE_POLLING, ENABLE_HEURISTIC_POLLING,
    GET_NUM_REQUESTS_IN_FLIGHT, INIT_ENGINE, SET_CONFIGURATION_SECTION_NAME,
    ENABLE_SW_FALLBACK, HEARTBEAT_POLL, DISABLE_QAT_OFFLOAD
```

Za probu, možemo kreirat ***RSA*** ključeve bez ***QAT***:

```
openssl speed rsa2048
```

A potom s podrškom za ***QAT***:

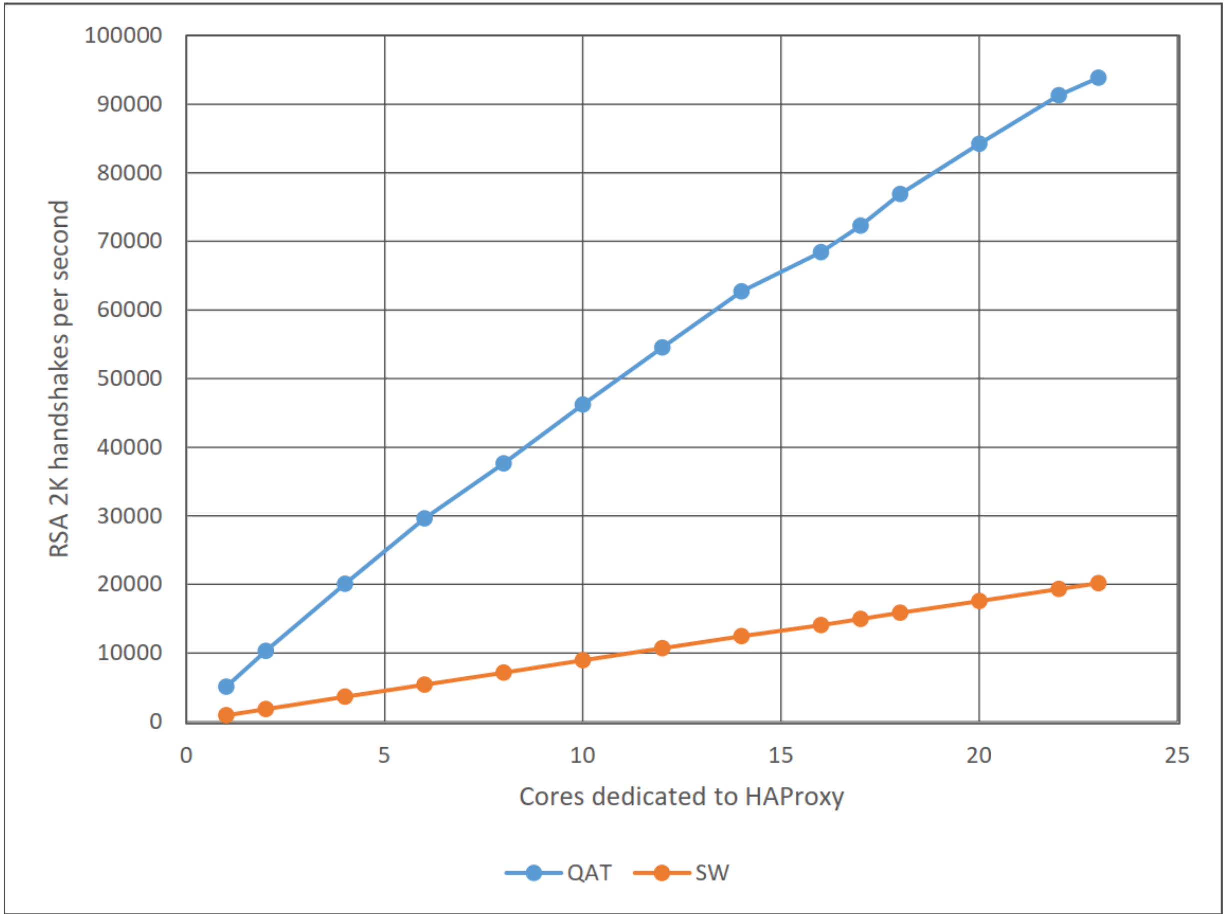
```
openssl speed -engine qatengine -async_jobs 8 rsa2048
```

Zatim usporedite oba testa!



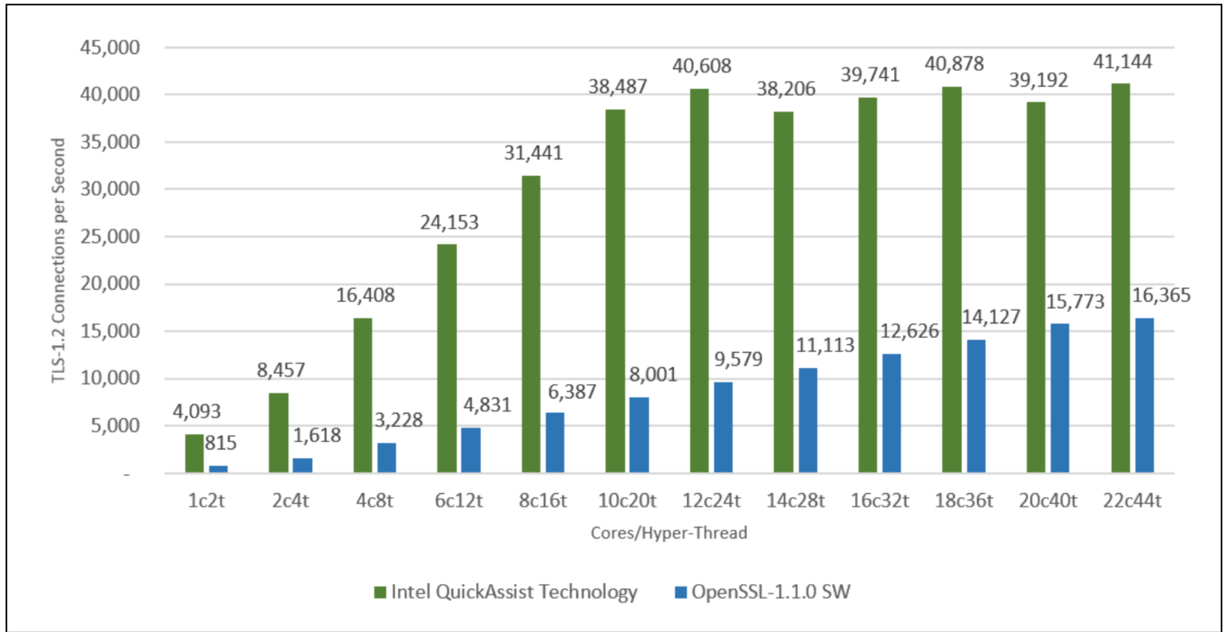
Važno je razumjeti i to da ***QAT*** može raditi *sinkrono* (sporije), ali od novijih inačica ***OpenSSL 1.1.x*** i *asinkrono* (brže) [*OpenSSL* opcija **`-async_jobs`**].

Potom su nam potrebne aplikacije poput: ***NGINX***, ***HAProxy*** ((**1291**),(**1292**)), koje su svjesne ***QAT*** preko *OpenSSL* sustava te su s njim i konfigurirane. Pogledajmo usporedbu [***HAproxy*** load balancera](), sa i bez upotrebe ***QAT*** tehnologije**:**




Izvor mjerenja (**1291**): ***Accelerating HAProxy* with Intel® QuickAssist Technology***.

Te pogledajmo usporedbu ***NGINX*** (web poslužitelja) sa i bez upotrebe ***QAT*** tehnologije:




Izvor mjerenja (**1290**): ***Intel® QuickAssist Technology (Intel® QAT) and OpenSSL-1.1.0: Performance.***

U oba mjerenja je jasno vidljivo ekstremno ubrzanje koje je dobiveno upotrebom ***QAT*** tehnologije.

**Izvori informacija:**
**(1268),(1269),(1270),(1271),(1272),(1273),(1274),(1275),(1277),(1278),(1279),(1280),(1281),(1282),(1283),(1284),(1285),(1286),(1287),(1288),(1289),(1290),(1291),(1292),** `man openssl`**,**`man certbot`**,**`man modprobe`**,**`man proc`**.**

## 26.4.5. Proxy poslužitelji

Posrednički odnosno Proxy poslužitelj je uobičajeno poslužiteljska aplikacija odnosno servis koji djeluje kao posrednik između klijenta koji zahtijeva resurs i poslužitelja koji taj resurs pruža. Proxy poslužitelj pri tome poboljšava privatnost, sigurnost i performanse. Naime, umjesto izravnog povezivanja na poslužitelj koji može ispuniti zahtjev za traženim resursom, kao što je određena datoteka ili web stranica (ili paketni repozitorij), klijent usmjerava zahtjev na proxy poslužitelj, koji procjenjuje zahtjev te izvodi potrebne radnje i mrežne transakcije. Proxy poslužitelj može se nalaziti bilo gdje u mreži: između korisničkog (našeg) računala i odredišnih poslužitelja na Internetu.
*Pojednostavljeno: Proxy poslužitelj je posrednik između (vašeg) web preglednika i Interneta. U mrežama koje ne koriste proxy poslužitelj, vaš web preglednik (pr. Mozilla Firefox) se direktno spaja na određenu web stranicu, kao i web preglednik svakog drugog računala u takvoj mreži.*

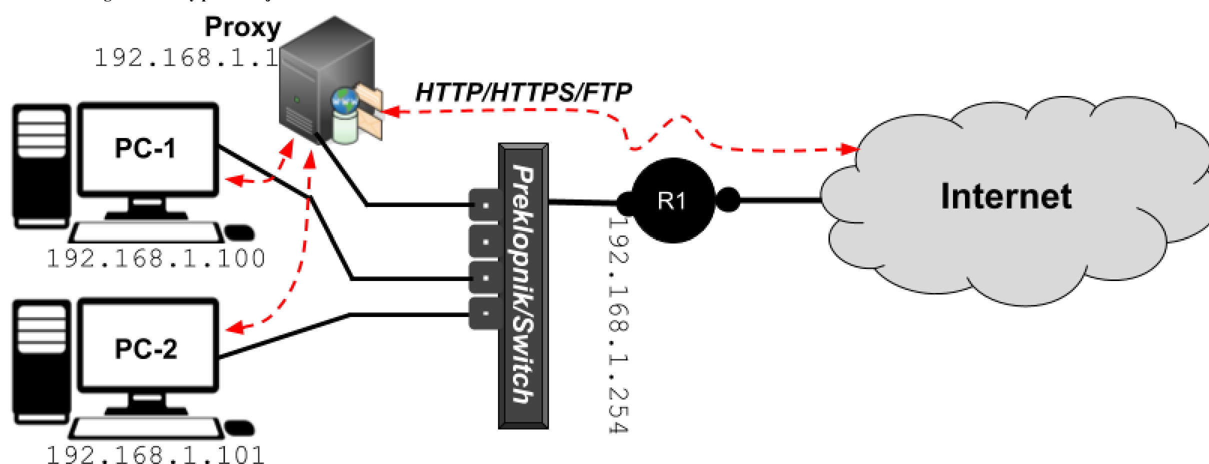
**Vrste Proxy poslužitelja**
U grubo postoji nekoliko vrsta odnosno kategorija Proxy poslužitelja:

- Proxy poslužitelj koji prosljeđuje nepromijenjene zahtjeve i odgovore obično se naziva ***Gateway*** ili ponekad proxy za tuneliranje odnosno "***Tunneling Proxy***".
- ***Forward Proxy*** odnosno proxy za prosljeđivanje je proxy s vezom prema internetu koji se koristi za dohvaćanje podataka iz širokog raspona izvora, a u većini slučajeva, bilo gdje s interneta.
- ***Reverse Proxy*** odnosno obično je interni proxy koji se koristi kao sučelje za kontrolu i zaštitu pristupa poslužitelju na privatnoj mreži. Reverse (obrnuti) proxy obično također obavlja zadatke kao što su uravnoteženje opterećenja (engl. *Load Balancing*), autentikacija, dekripcija i predmemoriranje sadržaja.
- ***Transparent Proxy*** - također poznat kao ***intercepting proxy***, ***inline proxy***, ili ***forced proxy***. Transparentni proxy presreće normalnu komunikaciju bez potrebe za posebnom konfiguracijom klijenta. Klijenti ne moraju biti svjesni postojanja ovakvog proxyja. Transparentni proxy obično se nalazi između klijenta i interneta, a ovaj proxy obavlja neke od funkcija usmjerivača. *Transparent proxy* obično se koriste u tvrtkama za provođenje sigurnosnih pravila (pristupa Internet sadržaju) te za ublažavanje administrativnih troškova budući da nije potrebna konfiguracija preglednika klijenta. On se može konfigurirati u grupnim pravilima; primjerice pomoću ***Active Directory*** pravila (*group policy*) ili na razini **DHCP** servisa za automatsko otkrivanje proxy poslužitelja. Presretanje se također može izvesti korištenjem *Ciscovog* **WCCP**-a (*Web Cache Control Protocol*). Ovaj protokol koristi se na usmjerivačima, definirajući koji portovi (i koji promet) će se slati na Proxy poslužitelj putem transparentnog preusmjeravanja na razini usmjerivača. **WCCP** protokl su implementirali i mnogi drugi proizvođaći mrežnih uređaja i mrežnih serisa poput primjerice: ***Citrix***, ***F5 Networks***, ***Fortinet***, ***Apache Traffic Server***, ***SQUID Proxy***, ***Sophos***, ***McAfee***, ***TrendMicro***, ali i mnogi drugi.

Po pitanju *Proxy poslužitelja* odnosno servisa **RFC 2616** nudi definiciju *Proxy* poslužitelja, koju možete proučiti!

Dakle računala na mreži na različite načine mogu koristiti Proxy poslužitelj za pristup mrežnim resursima; uobičajno web stranicama (HTTP/HTTPS) ili datotekama (pr. FTP) ili primjerice softverskim paketima na udaljenim mrežnim (paketnim) repozitorijima, kako je vidljivo na slici dolje.

**Slika 244.A Pogled na Proxy poslužitelj u mreži.**



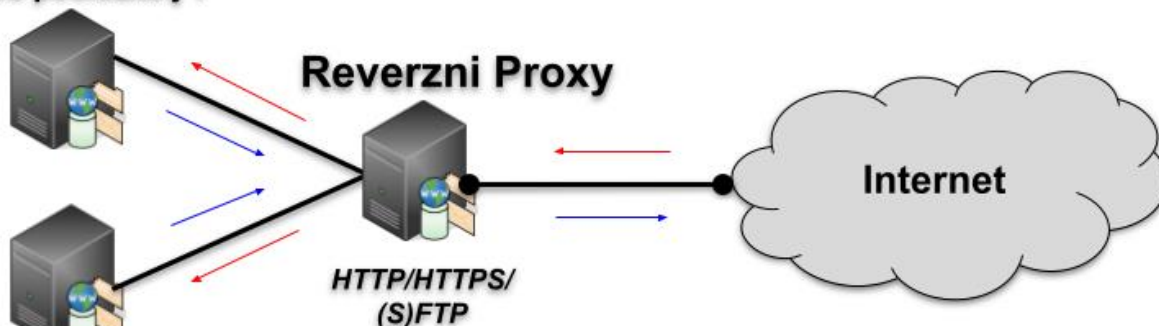



Za primjere upotrebe **Proxy** poslužitelja, za paketni menadžer (**YUM/DNF**) i web preglednike pogledajte poglavlje: **7.2.2.3. Upotreba posredničkog (*Proxy*) poslužitelja.**

## Reverzni Proxy

**Reverzni proxy** je proxy poslužitelj koji se klijentima čini kao običan poslužitelj. Reverzni proxy u radu prosljeđuju zahtjeve (uobičajeno s interneta) prema jednom ili više krajnjih poslužitelja (pr. web poslužitelja) koji obrađuju taj zahtjev. Odgovor izvornog poslužitelja vraća se kao da je došao izravno s *proxy* poslužitelja, ostavljajući klijenta bez znanja o izvornom poslužitelju.

**Slika 244.B Pogled na Reverzni Proxy poslužitelj u mreži.**




Kada se sve koriste klasični ***Proxy*** i ***Reverzni Proxy*** poslužitelji:

- Komprimiranje - *proxy poslužitelj* može optimirati i komprimirati sadržaj kako bi se ubrzalo vrijeme učitavanja.
- ***Međumemorija/predmemorija*** - *reverzni proxy* može rasteretiti web poslužitelje spremanjem u predmemoriju statičnog sadržaja poput slika i drugog statičnog grafičkog sadržaja, kako bi se ubrzalo dohvaćanje stranica s tih poslužitelja.
- Kod upotrebe enkripcije (SSL/TLS) i TLS akceleracije – kada se kreiraju sigurne/zaštićene web stranice, enkripciju upotrebom *Secure Sockets Layer* (**SSL**) ili **TLS** često ne obavlja sam web poslužitelj (pr. na slici **Web poslužitelj 1** i **Web poslužitelj 2**), već *Reverzni proxy* koji je često opremljen i hardverom za ubrzanje SSL/TLS odnosno enkripcije/dekripcije. To znači da se sva enkripcija i dekripcija odrađuju samo na *Reverznom TLS/SSL Proxy* poslužitelju, pa se samim time rasterećuju stvarni Web poslužitelji iza njega (**Web poslužitelji 1** i **2** na slici), prema kojima teče čisti (nekriptirani) HTTP promet. Ovakav način rada se naziva i ***TLS/SSL Offloading***. Nadalje, ovakav *Reverzni Proxy* koristi jedan TLS/SSL certifikat za pružanje SSL enkripcije za proizvoljan broj Web poslužitelja koje opslužuje, uklanjajući potrebu za zasebnim certifikatima na svakom unutarnjem Web poslužitelju. Loša strana je ta da svi Web poslužitelji iza Reverznog TLS/SSL Proxy poslužitelja (u ovom primjeru) moraju dijeliti DNS ime (FQDN) i IP adresu za sve SSL/TLS veze, a iznimka je korištenje tzv. *Server Name Indication-a* (**SNI**) (**1697**).
    - ***Load balancing*** odnosno balansiranje opterećenja - *reverzni proxy* može rasporediti opterećenje na nekoliko web poslužitelja, od kojih dodatno svaki može služiti vlastitom području primjene. U takvom slučaju, reverzni proxy možda će morati prepisati URL-ove na svakoj web-stranici: prijevod s vanjskih javnih URL-ova na unutarnje URL-ove.



Za **TLS/SSL** (HTTPS) i enkripciju pogledajte:
**26.4.4. Secure Socket Layer, Transport Layer Security i kriptografija.**

*Reverzni proxy* poslužitelji prema osnovnoj funkcionalnosti rada su zapravo ***Load Balanceri***.



Za **Load Balancing** pogledajte:
**23.4.5.1. Keepalived i balansiranje opterećenja (*Load Balancing*).**
**23.4.5.3. Balansiranje opterećenja (*Load Balancing*) – ručni rad.**
**23.4.5.4. Algoritmi za balansiranje opterećenja (Load Balancing).**

Neki od često korištenih ***Proxy*** i ***Reverznih Proxy*** poslužitelja otvorenog koda su:

- **SQUID Proxy** – on je ponokrvni Proxy poslužitelj s mnoštvo mogućnosti, opcija i parametara rada. On može raditi i kao *HTTP/HTTPS Mail Proxy* poslužitelj.
- **HAProxy** – iako je on primarno razvijan kao Load Balancer, on zapravo odrađuje funkcionalnosti *reverznog proxy* poslužitelja.
- **Apache web server** – iako je on je primarno web poslužitelj, pomoću dodatnih modula može imati najosnovnije funkcionalnosti *reverznog proxy* poslužitelja.
- **Nginx web server** – i on je primarno web poslužitelj, a pomoću dodatnih modula može imati najosnovnije funkcionalnosti *reverznog proxy* poslužitelja. On može raditi i kao *HTTP/HTTPS Mail Proxy* poslužitelj.

Postoje i Proxy poslužitelji za specifične protokole, poput specijaliziranih ***Mail Proxy*** poslužitelja, kao što su: **Proxmox Mail Gateway**, **Postfix** (mail poslužitelj, *Mail Proxy* i *Mail Relay*), **Dovecot** (mail poslužitelj, *Mail proxy* i *Mail Relay*) i drugi.

## Web Proxy Auto-Discovery Protocol i Proxy auto-config

Standardna metoda konfiguracije *Proxy* poslužitelja (pr. unutar web preglednika), jest ručna konfiguracija.



Za primjere konfiguracije **Proxy** poslužitelja, za paketni menadžer i web preglednike pogledajte poglavlje: **7.2.2.3. Upotreba posredničkog (*Proxy*) poslužitelja.**

Međutim, moguće je definirati (i konfigurirati) postavke upotrebe *Proxy* poslužitelja i na strani poslužitelja.
Naime *Web Proxy Auto-Discovery* (**WPAD**) je metoda koju koriste klijenti poput web preglednika za pronalazak web adrese (URL-a) konfiguracijske datoteke Proxy poslužitelja, koristeći DHCP i/ili DNS metode traženja.
*WPAD* protokol samo opisuje mehanizam za otkrivanje lokacije ove datoteke, ali najčešće primjenjivani format konfiguracijske datoteke je tzv. ***Proxy auto-config*** format (tzv. **PAC**) koji je izvorno dizajnirao *Netscape* 1996. godine za *Netscape Navigator* 2.0. *Proxy Auto-Configuration* (**PAC**) datoteka je definirana pomoću *JavaScript* koda.
Ova *PAC* datoteka određuje kako web preglednik treba odabrati proxy poslužitelj (ili više njih) na temelju ciljanog URL-a.
*Proxy Auto-Configuration* (**PAC**) datoteku je moguće i ručno konfigurirati unutar web preglednika ili drugog softvera; primjerice u postavkama web preglednika **Firefox**, kako je vidljivo na slici dolje.

**Slika 244.C. Pogled na Firefox i postavke Proxy poslužitelja.**

Connection Settings

Configure Proxy Access to the Internet

Automatic proxy configuration URL

http://server1.domena.com/proxy.pac    Reload

U ovom slučaju `proxy.pac` datoteka će se dohvatiti s URL-a: http://server1.domena.com/, a tom prilikom će se iz nje isčitati postavke Proxy poslužitelja. Prema konvenciji, *PAC* datoteka se obično zove `proxy.pac` dok ***WPAD*** standard koristi datoteku imena: `wpad.dat`.

Pogledajmo primjer jedne jednostavne *PAC* datoteke:

```
function FindProxyForURL(url, host) {
    if (shExpMatch(host, "*.tvrtka.com")) {
        return "PROXY proxy1.domena.com:7777";
    }
    if (shExpMatch(host, "*.internal")) {
        return "DIRECT";
    }
    return "PROXY proxy2.domena.com:8888";
}
```

Ovdje imamo primjer koji ima tri sekcije:

- U prvoj sekciji definiramo Proxy poslužitelj (`proxy1.domena.com` na portu `7777`), ali samo za URL-ove koji sadrže "`tvrtka.com`". dakle za domenu `tvrtka.com` će se koristiti ovaj Proxy poslužitelj.
- U drugoj sekciji deifiniramo da ako se pristupa lokalnim resursima unutar naše tvrtke, koji imaju URL koji sadrži "`internal`", da se Proxy poslužitelj uopće ne koristi, već da se tim resursima (poslužiteljima) pristupa direktno.
- U trećoj sekciji definiramo Proxy poslužitelj (`proxy2.domena.com` na portu `8888`) za sve ostale slučajeve odnosno za sve druge domene koje nismo naveli. To znači da će se ovaj Proxy koristiti za sve ostalo.

**WPAD**

Standard *WPAD* definira dvije metode koje je mguće koristiti za objavu lokacije proxy konfiguracijske datoteke (***PAC***), koristeći *Dynamic Host Configuration Protocol* (**DHCP**) ili *Domain Name System* (**DNS**) sustave:

- **DHCP** – prije dohvaćanja svoje web prve stranice, web preglednik koji implementira ovu metodu šalje **DHCPINFORM** upit lokalnom DHCP poslužitelju te koristi URL iz WPAD opcije iz odgovora poslužitelja. Ako DHCP poslužitelj ne pruži željene informacije, koristi se DNS metoda. Ako je, na primjer, FQDN naziv korisničkog računala **pc1.domena.com**, preglednik će prvo pokušati sljedeći URL dok ne pronađe proxy konfiguracijsku datoteku unutar domene klijenta: http://wpad.domena.com/wpad.dat a potom će pokušati sa svim poddomenama (ako postoje).



Na Windows računalima, ako je DNS upit neuspješan tada će se koristiti *Link-Local Multicast Name Resolution* (**LLMNR**) i/ili Windows NetBIOS.
Važno je razumjeti da **DHCP** ima veći prioritet od **DNS**-a: ako DHCP vraća WPAD URL, ne provodi se DNS traženje.
To radi samo s DHCPv4. U DHCPv6 nije definirana WPAD opcija.

Da bi *WPAD* radio, mora biti ispunjeno nekoliko uvjeta:

- Kako bi se koristila ova DHCP opcija, DHCP poslužitelj mora biti konfiguriran za opciju *site-local* **252** odnosno *auto-proxy-config* s definiranom vrijednosti ove opcije; pr. **http://domena.com/wpad.dat** gdje je "**domena.com**" adresa web poslužitelja.
  Primjerice **dnsmasq** DHCP opcija **252** izgleda ovako:
  ```
  dhcp-option=252,http://domena.com/wpad.dat
  ```

  Za primjere konfiguracije **dnsmasq** servisa pogledajte poglavlja:
  **25.8.3.3. Konfiguracija DHCP poslužitelja.**

- Kako biste koristili samo DNS metodu, potreban je DNS unos za računalo pod nazivom **WPAD**. Računalo na WPAD adresi mora biti u mogućnosti opsluživati web stranicu s pripadajućom *WPAD* (*PAC*) datotekom.



Za automatsku konfiguraciju **Proxy** poslužitelja preko **DHCP** poslužitelja pogledajte **DHCP** opciju **252:**
**25.8.3.1. Oblik DHCP poruke.**

## SOCKS i SOCKS Proxy

SOCKS je protokol za razmjenu mrežnih paketa između klijenta i poslužitelja putem specijaliziranog *Proxy* poslužitelja. Praktično, SOCKS poslužitelj preuzima TCP veze na željenu IP adresu i osigurava sredstva za prosljeđivanje tih paketa. Postoji nekoliko inačica SOCKS protokola (v,1. – v.5.) iako se SOCKS 5 najviše koristi, a opcionalno pruža i autentikaciju tako da samo ovlašteni korisnici mogu pristupiti poslužitelju.

SOCKS poslužitelj prihvaća dolaznu klijentsku vezu uobičajeno na TCP portu 1080, kako je definirano u **RFC 1928**. Međutim, moguće je koristiti i neki drugi port prema potrebi (kao na slici dolje).
Na slici je vidljiva upotreba **SOCKS 5** Proxy poslužitelja iz web preglednika *Firefox*.

**Slika 244.D. Pogled na Firefox i postavke SOCKS Proxy poslužitelja.**

Connection Settings

Configure Proxy Access to the Internet

Manual proxy configuration

SOCKS Host 127.0.0.1 Port 8888

SOCKS v4 SOCKS v5

U ovom primjeru konfiguracije mi koristimo SSH tunel do udaljenog Linux poslužitelja te u našem SSH klijentu (**PuTTY**) koristimo *SOCKS 5 Proxy* funkcionalnost. Odnosno mi se preko Web preglednika **Firefox** spajamo na SSH klijent na IP adresi: 127.0.0.1 odnosno na naše vlastito računalo na portu 8888, a koje je konfigurirano da bude SOCKS 5 Proxy.
Na taj način prolazimo kroz SSH tunel na udaljeni poslužitelj i kroz njega preko *Firefox-a* dolazimo do udaljenih web poslužitelja. SSH servisi podržavaju samo SOCKS 5 inačicu protokola.



Za primjer i opis ovakve konfiguracije **SSH + SOCKS** (**Dinamičko prosljeđivanje porta**) pogledajte poglavlje:
**26.2.1.6. SSH tuneliranje.**



***SOCKS**, a posebice **SOCKS 5** uobičajeno koriste: Web preglednici, Torrent klijenti i SSH klijenti.*
*U pravilu **SOCKS Proxy** ima manju latenciju od klasičnih HTTP/S Proxy poslužitelja.*

**Izvori informacija: (949),(950),(951),(1588),(1696),(1697),(1698),(1699), RFC 1928, RFC 2616, RFC 2965**

# 26.5. Dohvaćanje web sadržaja (naredbe wget i curl)

Za dohvaćanje sadržaja preko mreže (weba) na raspolaganju imamo dostupne dvije korisne naredbe:

- `wget` – namijenjena je za dohvaćanje sadržaja upotrebom: HTTP, HTTPS i FTP protokola.
- `curl` – namijenjena je za dohvaćanje, ali i postavljanje sadržaja upotrebom: *HTTP, HTTPS, FTP, FTPS, SCP, SFTP, TFTP, DICT, TELNET, LDAP* i mnogih drugih protokola. Lista s novim protokolima za ovaj program se stalno povećava.

## Wget

U slučajevima kada imamo potrebu dohvatiti (kopirati) sadržaj neke web stranice, za tu potrebu nam je najzgodnija naredba `wget`. Prvo ju instalirajmo na sljedeći način:

```
yum -y install wget
```

Sintaksa upotrebe ove naredbe je: ***wget OPCIJE URL***, pri čemu je URL adresa web stranice koju želimo dohvatiti (kopirati).

Naredba `wget` nam nudi stotine i stotine opcija od kojih ćemo navesti samo nekoliko osnovnih:

- `-O ime datoteke` – ako želimo da se lokalno ime datoteke zove drugačije od izvorišne datoteke.
- `-P ime direktorija` – ako želimo da se datoteka snima u neki drugi lokalni direktorij (mapu).
- `--limit-rate=Xz` – želimo li ograničiti brzinu dohvaćanja (download) datoteka. Pri tome je `X` veličina, a `y` može biti prefiks (`k`=kB/s, `m`=MB/s, `g`=GB/s).
- `--show-progress` – prikazuje nam postotak napretka kopiranja, tijekom kopiranja.
- `-i lista.txt` – naznačava da će se kopirati više datoteka (s više adresa), navedenih u datoteci imena: `lista.txt`.
- `--ftp-user=FTP_korisnk –ftp-password=FTP_lozinka` – naznačava da ćemo koristiti FTP protokol za dohvaćanje datoteka, te navodimo korisničko ime i lozinku s kojim će se ***wget*** spajati na udaljeni FTP poslužitelj.
- `-W XY` – pauzira se dohvaćanje na `XY` sekundi između svakog novog dohvaćanja.
- `-m` – naznačava da radimo download cijele web stranice (Engl. *Mirror*).
- `-k` – naznačava ***wget***-u da konvertira *linkove (datoteka koje su simboličke poveznice)* u valjane datoteke.
- `-p` – označava da će ***wget*** preuzeti sve datoteke potrebne za pravilno prikazivanje zadane HTML stranice. To uključuje stvari kao što su umetnute slike, audio (zvukovi) i *stylesheets* podaci.
- `-b` – označava da će ***wget*** preuzeti sadržaj u pozadini (Engl. *Background*), bez intervencije korisnika.
- `--no-check-certificate` – ovime definiramo da se valjanost TLS/SSL certifikata neće provjeravati (za *HTTPS*).
- `--user-agent="XY"` – ovdje možemo definirati korisničkog agenta koji definira emulirani Web preglednik s kojim dohvaćamo web sadržaj. Primjerice: `„Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"` s
- `--proxy-user „USERName" -proxy-password „PASSWORD"` – ako na Internet moramo izlaziti preko *PROXY* poslužitelja, ovdje definiramo korisnički račun i lozinku za *PROXY* poslužitelj. Pogledajte kako se definira *PROXY* poslužitelj; kako je vidljivo u poglavlju: **7.2.2.3. Upotreba posredničkog (*Proxy*) poslužitelja.**

Osim klasične uporabe naredbe `wget`, moguće je sve opcije njenog rada definirati konfiguracijskoj datoteci, specifičnoj za svakog korisnika. Stoga je za takav rad moguće unutar našeg kućnog (*home*) direktorija kreirati skrivenu datoteku imena: `.wgetrc`. Takva datoteka bi primjerice za korisnika `hrvoje` onda bila datoteka: `/home/hrvoje/.wgetrc`.
Ona može sadržavati sljedeće opcije:

```
# Definicija PROXY poslužitelja (IP:PORT) za HTTP, HTTPS i FTP
https_proxy = http://192.168.1.1:3128/
http_proxy = http://192.168.1.1:3128/
ftp_proxy = http://192.168.1.1:3128/

# Definicija korisnika PROXY poslužitelja za http i ftp protokole i njegove lozinke
user=hrvoje
password=SuperDobraLozinka2762783672!#&/()!1
```

Ovdje smo definirali *Proxy* poslužitelj na IP adresi 192.168.1.1, na portu `3128` i to za HTTP, HTTPS i FTP protokole. Zatim smo definirali s kojim korisničkim računom će se naredba `wget` spajati na navedeni *Proxy* poslužitelj i s kojom lozinkom, za izlaz na Internet. Naime, ako u našoj mrežni na Internet možemo izaći sâmo i isključivo preko *Proxy* poslužitelja, tada ga je potrebno definirati ovdje (za naredbu `wget`) ili koristiti onaj koji smo postavili za cijeli sustav.

Za primjer kopirajmo Linux kernel generacije 5.10, s adrese: https://cdn.kernel.org/pub/linux/kernel/v5.x/

```
wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.10.25.tar.xz
```

Ova kernel datoteka (`linux-5.10.25.tar.xz`) će biti snimljena na naš lokalni disk, u direktorij (mapu) u kojoj se trenutno nalazimo.

Pogledajmo još primjer u kojem ćemo kopirati sadržaj cijele Web stranice: https://www.opensource-osijek.org.

```
wget -m -k -p https://www.opensource-osijek.org
```

## Curl

U slučajevima kada imamo potrebu dohvatiti (kopirati) sadržaj neke web stranice ili postaviti nešto na nju, kao i poslati određenu naredbu specifičnu za određeni protokol [HTTP] (PUT,GET,POST,…), za tu namjenu nam je najpraktičnija naredba `curl`. Naime naredba `curl` prenosi podatke na mrežni poslužitelj ili s njega, koristeći jedan od podržanih protokola: *DICT, File, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET* i *TFTP*. Naziv ove naredbe je nastao od naziva „*Client for URLs*" koji označava klijenta za URL. Prvo ju instalirajmo na sljedeći način:

```
yum -y install curl
```

Sintaksa upotrebe ove naredbe je: ***curl OPCIJE URL***, pri čemu je URL (web) adresa stranice kojoj želimo pristupiti.
Naredba `curl` nam nudi tisuće opcija od kojih ćemo proći sâmo nekoliko kroz primjere koji slijede. Naredba `curl` također može koristiti već definirani *Proxy* poslužitelj ili ga možemo definirati s opcijom: `-x PROXY:PORT`.

**Slijede primjeri, prvo s upotrebom HTTP/HTTPS protokola.**
Kopirajmo lokalno sâmo datoteku `index.html`, s našeg web poslužitelja i zadržimo njeno izvorno ime (prekidač `-O`):

```
curl -O https://www.opensource-osijek.org/index.html
```

Naime, ako sâmo pokušamo pokrenuti `curl` na sljedeći način:

```
curl https://www.opensource-osijek.org
```

Dobit ćemo ispis HTML dokumenta (*index.html*) u terminalu odnosno na „ekranu", što je ovdje standardan način rada.

Moguće je dohvatiti (kopirati) i više datoteka s web poslužitelja istovremeno, nîzanjem pomoću prekidača (`-O`) nakon kojega mora slijediti URL stranice odnosno datoteke na njoj.
Neke web stranice rade preusmjeravanje; poput primjerice stranice www.google.com, koja ovisno o zemlji (državi) iz koje se poziva, odnosno radi se tzv. referenciranje na domenu unutar države (www.google.hr).
To se radi umetanjem „**`href`**" HTML oznake za referenciranje na drugu URL adresu.
Ovdje je moguće naložiti programu `curl` da dohvati sadržaj sa preusmjerene stranice (prekidač `-L`), primjerice sa:

```
curl -L http://www.google.com
```

Moguće je primjerice dohvatiti samo HTML zaglavlje web stranice, poput ovoga u primjeru (koristimo prekidač `-I`):

```
curl -I http://www.google.com
```

U prethodno navedenim primjerima rada s HTTP protokolom, koristila se **HTTP GET** metoda. Međutim moguće je koristiti i druge metode poput: **PUT**, **POST**, **DELETE** i **PATCH**. Pogledajmo primjer s metodom **HTTP PUT** (slanja na poslužitelj).
Slati podatke upotrebom HTTP metoda, možemo raditi u tri formata:

- *Field=value (multipart/form-data)* s prekidačem `-F`.
- *Application/x-www-form-urlencoded* formatu s prekidačem `-d`.
- Ili u JSON formatu, sa primjerice: `-X POST -H „Content-Type: application/json"`.

Pogledajmo sintaksu primjera slanja **HTTP POST** metode na web poslužitelj:

```
curl -X POST -d 'name=test' -d 'email=test@test.com' https://test.com/stranica.php
```

Dakle s ovime možemo simulirati slanje podataka odnosno određenog upita na web poslužitelj, od kojeg ćemo potom dobiti odgovor. Pogledajmo i kako poslati određenu datoteku (`wallpaper.jpg`) na web poslužitelj, pomoću **HTTP PUT** metode:

```
curl -X POST -F 'image=@/home/hrvoje/slike/wallpaper.jpg' http://test.com/slike
```

Provjerimo podržava li određena web stranica odnosno poslužitelj ***HTTP/2*** protokol:

```
curl -I -http2 https://google.com/
```

**Rad s FTP protokolom**.
Naredba `curl` omogućava i rad s ***FTP*** protokolom, pogledajmo sintaksu kako kopirati datoteku s ***FTP*** poslužitelja:

```
curl -u ftpkorisnik:ftplozinka -O ftp://ftp-server.org/dokumenti/dokument.pdf
```

Dakle u ovom primjeru dohvaćamo PDF datoteku sa ***FTP*** poslužitelja, koristeći korisnički račun i lozinku (definiraju se nakon prekidača `-u`) za FTP poslužitelj.

Moguće je postaviti (`-T`) i datoteku na ***FTP*** poslužitelj, sa sljedećom sintaksom:

```
curl -u ftpkorisnik:ftplozinka -T myfile.txt ftp://ftp-server.org
```

**Rad sa SMTP protokolom**.
Naredba `curl` omogućava i rad sa **SMTP** protokolom.
Dakle možemo poslati elektroničku poštu na poslužitelj elektroničke pošte (mail poslužitelj):

```
curl –mail-from hrvoje@test.com –mail-rcpt admin@firma.com smtp://mailserver.com
```

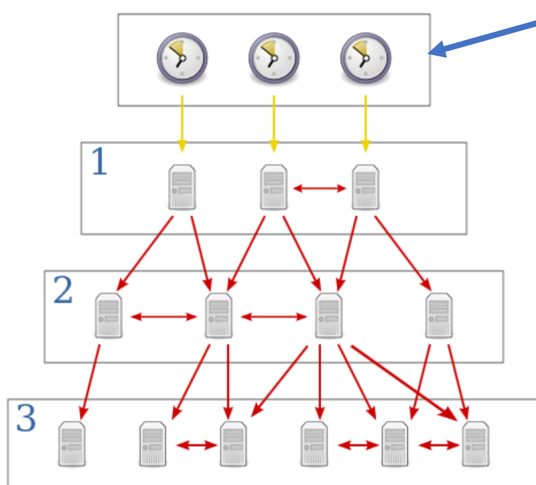
Naša adresa elektroničke pošte slijedi nakon opcije `-mail-from`, dok je adresa primatelja elektroničke pošte definirana nakon opcije `-mail-rcpt`. I na kraju definiramo na koji poslužitelj elektroničke pošte ju šaljemo, s parametrom: `smtp://`.

**Izvori informacija: (970),(971),(972),(973),(974), `man wget`, `man curl`, CURL knjiga: https://curl.se/book.html.**

# 26.6. NTP

Za osnove rada **NTP** protokola (engl. *Network Time Protocol*), pročitajte poglavlje: **10.4.2. NTP**.
Pojednostavljeno *NTP* protokol koristi se za postavljanje točnog vremena odnosno datuma i sata, kontaktiranjem servisa točnog vremena koji se nalazi negdje na mreži, a koji kontaktira naš *NTP* klijent kako bi lokalno postavio točno vrijeme i datum.
Namjena upotrebe *NTP* protokola je uskladiti sva računala unutar vremenskih okvira od nekoliko milisekundi, korištenjem takozvanog ***UTC** vremena* odnosno takozvanog *Engl. Coordinated Universal Time*. *NTP* je pri tome dizajniran za ublažavanje učinaka varijabilne latencije odnosno kašnjenja koja se događaju u mreži. *NTP* obično može održavati točno vrijeme unutar desetak milisekundi preko (javnog) interneta, a može postići bolju točnost od jedne milisekunde točnosti u lokalnim mrežama pod idealnim uvjetima. Međutim asimetrične mreže kao i zagušenja mreže mogu uzrokovati pogreške od 100 milisekundi ili više.
*NTP* protokol radi na principu modela: klijent-poslužitelj, ali se lako može koristiti u takozvanoj *peer-to-peer* komunikaciji gdje oba partnera smatraju da je onaj drugi potencijalni izvor točnog vremena. NTP protokol radi tako da se za transport koristi **UDP** protokol na portu broj **123**. U komunikaciji se dodatno mogu koristiti i *multicast* ili *broadcast* komunikacija, gdje klijenti pasivno slušaju ažuriranja vremena nakon početne kalibracijske izmjene. NTP šalje i upozorenje o bilo kakvom odstupanju tijekom prilagodbe, ali ne nosi informacije o lokalnim vremenskim zonama ili ljetnom odnosno zimskom računanju vremena. Trenutno aktivna inačica protokola je inačica 4 (*NTPv4*) koja je definirana standardima:
RFC 5905 i RFC7822. Ova inačica je kompatibilna s inačicom 3, koja je pak definirana u starijem standardu RFC 1305.
Naime *NTP* koristi hijerarhijski višeslojni sustav vremenskih izvora. Svaka razina ove hijerarhije zove se ***STRATUM*** i dodjeljuje mu se broj koji počinje s nulom, za referentni sat na vrhu hijerarhije. Poslužitelj sinkroniziran s poslužiteljem *stratuma **n*** radi na *stratumu **n** + **1***. Vrijednost ***n*** predstavlja udaljenost od referentnog sata i koristi se za sprečavanje cikličkih ovisnosti u hijerarhiji. Međutim *stratum* nije uvijek pokazatelj kvalitete ili pouzdanosti; tako recimo nije neuobičajeno pronaći vremenski izvor *stratum* (sloja) tri (3) koji je kvalitetniji od ostalih vremenskih izvora stratum (sloja) dva (2). Pogledajmo sliku koja predstavlja *stratum* (slojeve) u *NTP* hijerarhiji. Možemo reći kako je mjera ili jedinica udaljenosti od *Atomskog* sata (na vrhu slike **244**) upravo *stratum* broj, a što je on veći, to je poslužitelj udaljeniji od njega. Pogledajmo značenje *stratum* brojeva:

**Slika 244. NTP poslužitelji u hijerarhiji.** Autor fotografije: **Benjamin D. Esham** (bdesham)




- **Stratum 0** predstavlja najprecizniji *Atomski* sat, poput satova do kojih se može doći preko *GPS* sustava ili posebnog radio sata (pr. DCF77). Ovi uređaji se nazivaju i referentnim (*atomskim*) satovima.
- **Stratum 1** NTP poslužitelji; pošto su spojeni na referentni sat (*atomski* u konačnici) imaju očekivano odstupanje od sâmo nekoliko mikro sekundi
- **Stratum 2** NTP poslužitelji imaju nešto veće kašnjenje, ali se oni mogu spajati (sinkronizirati) na više **Stratum 1** poslužitelja istovremeno, kao i na više **Stratum 2** poslužitelja kako bi dobili točnije vrijeme.
- **Stratum 3** NTP poslužitelji imaju još malo veće kašnjenje, ali se oni također mogu spajati (sinkronizirati) na više **Stratum 2** poslužitelja istovremeno, kao i na više **Stratum 3** poslužitelja.
- **Stratum 4** NTP poslužitelji imaju još nešto veće kašnjenje, ali se i oni mogu spajati (sinkronizirati) na više **Stratum 3** poslužitelja istovremeno, kao i na više **Stratum 4** poslužitelja i tako sve do **Stratum 15** koji je zadnji mogući element u hijerarhiji.



**Stratum 16** se koristi kao poruka kada je nemoguća sinkronizacija preko NTP poslužitelja

Pogledajmo naše računalo koje ima konfiguriran *NTP* klijent, koji se spaja na vanjski NTP poslužitelj: `os.ntp.carnet.hr`.
Sve ćemo pratiti s naredbom `ntpq -p` na sljedeći način:

```
ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
================= ============= ===== ==== ==== ====== ======= ======== =======
*os.ntp.carnet.hr 161.53.123.8    2 u   96  1024  340    8.244   -0.110   0.293
```

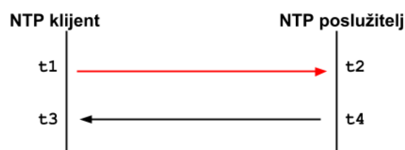
Na ovom ispisu vidimo sljedeće:

- Udaljeni NTP poslužitelj (`os.ntp.carnet.hr`) ima oznaku `*` na početku prvog stupca (`remote`), a koja označava kako je on aktivan i u upotrebi je. Pogledajmo i sve oznake koje se mogu ovdje pojaviti:
  - `*` označava NTP izvor (poslužitelj) koji je trenutno u upotrebi i aktivan je (ovo je normalno stanje).
  - `#` označava kako je ovaj NTP izvor odabran, ali se neće koristiti jer je previše udaljen.
  - `o` označava kako je ovaj NTP izvor odabran te da se koristi metoda: „*Pulse Per Second* (***PPS***)".
  - `+` označava kako je ovaj NTP izvor odabran i uključen u izbor, ali tek kao sekundarni izvor koji će se koristiti sâmo, ako je primarni (`*`) nedostupan.
  - `x` označava NTP izvor koji ima preveliko odstupanje vremena (`offset` polje) pa neće biti u upotrebi. Ova oznaka se zove i „***false ticker***".
  - `.` označava NTP izvor s kraja liste NTP poslužitelja (kao ispravan kandidat).
  - `-` označava NTP izvor koji je odbačen od strane NTP algoritma.
  - ` ` (prazno mjesto) označava NTP izvor koji je odbačen od strane višeg stratum izvora.

- U stupcu `refid` je vidljivo za naš udaljeni NTP poslužitelj: (`os.ntp.carnet.hr`) na koji se on NTP poslužitelj viši u hijerarhiji spaja; konkretno se on spaja na: `161.53.123.8`.
- Udaljeni NTP poslužitelj `os.ntp.carnet.hr` je u stratumu dva (stupac `st`).
- S udaljenim NTP poslužiteljem `os.ntp.carnet.hr` komuniciramo (stupac `t`) nekom od metoda komunikacije:
  - **`u`** – *unicast* poruke: direktno na IP adresu udaljenog NTP poslužitelja (ovo je zadana metoda rada).
  - **`b`** – *broadcast* poruke: slanjem na cijelu mrežu odnosno mrežni segment.
  - **`m`** – *multicast* poruke: komunikacija preko *multicast* grupe.
  - **`l`** - lokalno – direktno na **GPS** ili radio uređaj (pr. DCF77) za spajanje s atomskim satom.
  - - - adresa mreže.
- Potom slijede stupci:
  - `when` – koji označava prije koliko sekundi smo se zadnji puta spojili na udaljeni NTP poslužitelj.
  - `poll` – koji označava, svakih koliko sekundi će se naš NTP klijent spajati na udaljeni NTP poslužitelj.
- `reach` – označava koliko puta je uspješno ili neuspješno kontaktiran udaljeni NTP poslužitelj (izvor) prema sljedećem modelu oznake, koji predstavlja bit maska za zadnjih osam (8) poruka. Ovih zadnjih osam poruka predstavlja osam (8) bitova od kojih jedinica (1) predstavlja uspješan kontakt, a nula (0) neuspješan. Dakle ovo je praktično *FIFO* bit maska za zadnjih osam poruka koja se čita od desno na lijevo. Pogledajmo neke primjere:
  - $377_{oktalno}$ znači kako je svaki puta uspješno kontaktiran: $377_{oktalno} = 11111111_{binarno}$
  - $376_{oktalno}$ znači kako je svaki puta uspješno kontaktiran, usim prvog pokušaja: $376_{oktalno} = 11111110_{binarno}$
  - $375_{oktalno}$ znači kako je svaki puta uspješno kontaktiran, usim drugog pokušaja: $375_{oktalno} = 11111101_{binarno}$
- `delay` – označava vrijeme koje je bilo potrebno da mrežni paket dođe od izvora (NTP poslužitelja na mreži) do nas. Ovo vrijeme se naziva i prolazno vrijeme (*round trip time)*. Navedeno vrijeme je iskazano u milisekundama i ne utječe na pogrešku u korekciji vremena jer se ono uzima u obzir tijekom izračuna vremena. U normalnom radu ovisno o mreži preko koje se pristupa određenom izvoru (*NTP*), vrijeme pristupa (ovo vrijeme) može biti i nekoliko desetaka milisekundi. U vršnim slučajevima moguće je u slučaju kada je to vrijeme preveliko, da se taj izvor više neće koristiti odnosno dobiti će oznaku `#`.
- `offset` – označava razliku između lokalnog vremena (sata) i vremena (sata) koji pokazuje udaljeni NTP izvor odnosno NTP poslužitelj. Ovo vrijeme se koristi za prilagodbu lokalnog vremena odnosno sata. U vršnim slučajevima kada tijekom sinkronizacije udaljeni izvor (NTP poslužitelj) ponovno pokazuje veliku vrijednost odnosno ovo odstupanje (*offset*), a pogotovo u odnosu na neki drugi NTP poslužitelj istog *stratum broja*, tada će taj NTP izvor biti označen kao nepouzdan i biti vidljiv kao `x` ili proglašen kao ***falsetick***. Isto stanje se može dogoditi i u slučaju kada imamo dva ili više NTP izvora koji se prijavljuju da pripadaju istom stratumu (pr. *stratum 1*) iako jedan u odnosu na drugi imaju veliku razliku u vremenu (satu) što je vidljivo i u razlici u ovoj vrijednosti (*offset*).
  U navedenom slučaju kada oba izvora daju različit *offset,* a pripadaju istom stratumu, oba će biti odbačena i imati oznaku `x`, a tada će se koristiti neki drugi (treći) NTP izvori. Ovo vrijeme je iskazano u milisekundama.
- `jitter` – označava razliku u vremenu izraženu u milisekundama, između dva primljena paketa od strane NTP izvora. Uzroci vremenske razlike ovdje mogu ukazivati na problematičnu mrežu prema konkretnom NTP izvoru; u kojoj se stalno mijenja kašnjenje odnosno `delay` pa je ova vrijednost pokazatelj oscilacija u kašnjenju mreže prema tom NTP izvoru ili postoje problemi u radu tog NTP izvora (NTP poslužitelja).

Pogledajmo i klasičnu komunikaciju NTP klijent → NTP poslužitelj u slučaju kada imamo samo jedan NTP poslužitelj:

**Slika 245. NTP komunikacija**.




**1.** NTP klijent šalje prvi paket na NTP poslužitelj. Ovaj paket sadrži vremensku oznaku (***t1***) kada je paket poslan; prema njegovom (klijentovom) lokalnom satu, kao i ostale vremenske oznake i druge podatke. Ova vremenska oznaka se zapisuje u polje: *`Transmit Timestamp`*.
**2.** NTP poslužitelj sprema vrijeme kada je paket zaprimio od klijenta (***t2***) koje se zapisuje u polje `Receive Timestamp` kao i vrijeme koje je dobio od klijenta (***t1***), a ***t1*** sada sprema u polje *`Origin Timestamp`* te sprema i novo vrijeme u trenutku kada šalje ovaj paket nazad prema klijentu (***t3***), a ***t3*** vrijeme se sprema u polje `Transmit Timestamp`.
**3.** Klijent sada interno zapisuje vrijeme kada je paket od poslužitelja došao kao ***t4*** vrijeme.
NTP klijent pomoću ova četiri vremena izračunava relativnu korekciju vremena (sata) koju mora napraviti.

Za više detalja o izgledu NTP paketa na mreži, možete pogledati *PCAP* datoteku, koju možete analizirati u programu ***Wireshark***: https://wiki.wireshark.org/NTP

Međutim vremenski pomak koji klijent mora napraviti nije trenutan već se odrađuje u koracima. To znači da, ako NTP klijent (pr. naše računalo) primjerice kasni 15 minuta od NTP izvora (poslužitelja) algoritmi pomoću kojih će se vrijeme uskladiti napravljeni su tako da će se naše vrijeme (klijenta) polako prilagođavati NTP izvoru (poslužitelju). Završni dio NTP protokola u komunikaciji čini algoritam frekvencije slanja (*polling frequency*). Naime NTP klijenti u redovitim intervalima šalju poruke NTP poslužitelju. Ovaj redovni interval obično je postavljen na 64 sekunde. Ako poslužitelj nije dostupan, NTP će se odmaknuti od ovog broja sekundi slanja poruka, udvostručujući vrijeme odustajanja pri svakom neuspješnom pokušaju slanja kako bi se u konačnici postigla minimalna stopa slanja od jednog (1) slanja svakih 36 sati. Kada se NTP pokušava ponovno sinkronizirati s poslužiteljem, povećat će frekvenciju slanja poruka i poslat će se odjednom osam paketa u razmaku od dvije sekunde. Kada sat klijenta radi unutar dovoljnog malog odstupanja od sata poslužitelja, NTP produljuje interval slanja i šalje osam paketa svakih četiri do osam minuta (ili 256 do 512 sekundi).

Kada je klijent konfiguriran za upotrebu sâmo jednog NTP poslužitelja, sat klijenta se podešava pomicanjem vremena kako bi se odmak sa satom NTP poslužitelja sveo na nulu, pod uvjetom da je vrijednost *offseta* poslužitelja unutar prihvatljivog raspona. Kada je klijent konfiguriran s više NTP poslužitelja, klijent će koristiti algoritam odabira, za odabir željenog poslužitelja (Engl. *preffer server*) za sinkronizaciju između više poslužitelja kandidata. Međutim iako je odabran sâmo jedan primarni NTP poslužitelj, prati se više vremenskih signala iz više izvora odnosno s više NTP poslužitelja koje imamo konfigurirane na sustavu. Sve zbog mogućnosti da neki od njih (uključujući aktivni primarni NTP poslužitelj) postane nepodoban, a u tom slučaju algoritam odabire poslužitelj s najnižim stratum brojem i to s minimalnim `offset` i `jitter` vrijednostima.

Algoritam koji NTP koristi za izvođenje ove operacije je takozvani *Marzulloov* algoritam. Kada imamo konfiguriran NTP klijent, on pokušava zadržati sat klijenta (računala) usklađen s referentnim vremenskim satom (NTP poslužiteljem).
U svakom slučaju kako bi to učinio, NTP klijent standardno prilagođava lokalno vrijeme u malim pomacima (*offsetima*) jer bi veći pomaci mogli uzrokovati nuspojave tijekom pokretanja aplikacija ili kod već pokrenutih aplikacija koje su osjetljive na nagle promijene vremena. Naime neke aplikacije su vrlo osjetljive na promjene lokalnog sata (vremena).
Ovo malo podešavanje provodi se sistemskim pozivom koji smanjuje odnosno podešava lokalno vrijeme odnosno sistemski sat, na koji se oslanjaju svi programi (aplikacije) na sustavu, sve do postizanja ispravke vremena koja je tražena. Ovo pomicanje sata je spor proces za velike vremenske odmake; tipična brzina pomjeranja (usklađivanja) sata je 0,5 ms korekcije po sekundi.


Kod većih odstupanja između klijentovog sata i sata na referentnom ***NTP*** poslužitelju, usklađivanje klijentovog sata (točnog vremena) može potrajati i satima.


Postoji i jednostavniji protokol naziva *Simple Network Time protocol* (**SNTP**) čija implementacija je jednostavnija i primijenjenija ***embeded*** sustavima. **SNTP** je definiran u: RFC2030 i RFC4330 . Klijentska naredba mu je `sntp`
**SNTP** u pozadini koristi NTP protokol za komunikaciju.

**Izvori informacija: (438),(439),(441),(442),(444),(445), `man ntpq`, RFC 1305, RFC 5905, RFC7822, RFC2030,RFC4330**

## 26.6.1. Struktura NTP paketa

U ovom kratkom naprednom poglavlju vidjet ćemo strukturu NTP paketa u kojem je točno definirana struktura sâmog paketa:

| Leap Indicator | NTP Version Number | Mode | Stratum (Peer) | Poll | Precision (Peer clock) |
|---|---|---|---|---|---|
| Root Delay | | | | | |
| Root Dispersion | | | | | |
| Reference identifier (Reference ID) | | | | | |
| Reference Timestamp | | | | | |
| Origin Timestamp | | | | | |
| Receive Timestamp | | | | | |
| Transmit Timestamp | | | | | |

Pogledajmo i osnovne opise ovih polja:

- `Leap Indicator` – označava je li zadnja minuta trenutnog dana imala dodatnu sekundu koju treba dodati (*leap second*). Ova vrijednost može biti:
    - `0`: Označava kako nema potrebe za ovom korekcijom.
    - `1`: Označava kako zadnja minuta dana ima 61 sekundu.
    - `2`: Označava kako zadnja minuta dana ima 59 sekundi.
    - `3`: Označava kako sat nije sinkroniziran.

- `NTP Version Number` – označava inačicu NTP protokola, a može biti: `3` ili `4`

- `Mode` – označava način rada, što ovisi o vrsti poruke, a on može biti:
    - `0`: Rezervirano.
    - `1`: Simetrično aktivan način rada (*Symmetric active*).
    - `2`: Simetrično pasivan način rada (*Symmetric passive*).
    - `3`: *Client (ako klijent šalje poruku).*
    - `4`: *Server (ako poslužitelj šalje poruku).*
    - `5`: *Broadcast način rada.*
    - `6`: NTP kontrolna poruka.
    - `7`: Rezervirano za privatnu upotrebu.

- `Stratum` – označava stratum kojemu NTP poslužitelj pripada (on ga sam postavlja), moguće vrijednosti su:
  - `0`: Nespecificirano.
  - `1`: Označava primarni stratum 2 poslužitelj.
  - `2-15`: Označava sekundarni stratum poslužitelji (2-15).
  - `16`: Označava nesinkronizirano vrijeme odnosno označava grešku.
  - `17-255`: Rezervirano (za buduću upotrebu i posebne namjene).

- `Poll` – označava svakih koliko sekundi će se naš NTP klijent spajati na udaljeni NTP poslužitelj, Ovo vrijeme je definirano kao $log_2$ vrijednost u sekundama, pa tako pr. 6 znači 64 sekunde, 10 znači 1024 sekunde i tako dalje.

- `Precision` – označava preciznost sistemskog sata NTP izvora/poslužitelja, definirano u sekundama (pr. 0.000008).

- `Root Delay` – označava kašnjenje od poslužitelja do primarnog referentnog izvora. Vrijednost je 32-bitna u jedinici sekunde, s frakcijskom točkom između bitova 15 i 16. Ovo je polje značajno samo u porukama poslužitelja.

- `Root Dispersion` – označava maksimalnu pogrešku zbog tolerancije oscilacije frekvencije sata. Vrijednost je 32-bitna u jedinicama sekunde, s frakcijskom točkom između bitova 15 i 16. Ovo je polje značajno samo u porukama poslužitelja.

- `Reference identifier` – za sve osim stratum **1** poslužitelja označava IP adresu vršnog NTP poslužitelja. Dok za stratum **1** poslužitelje označava u ***ASCII*** kodu (do 4 slova), definiciju izvora točnog vremena:
  - `LOCL` - Nekalibrirani lokalni sat.
  - `GOES` – *Geosynchronous Orbit Environment Satellite* [RFC5905]
  - `GPS` – *Global Position System* [RFC5905]
  - `GAL` – *Galileo Positioning System* [RFC5905]
  - `PPS` – *Generic pulse-per-second* [RFC5905]
  - `IRIG` – *Inter-Range Instrumentation Group* [RFC5905]
  - `WWVB` – *LF Radio WWVB Ft. Collins, CO 60 kHz* [RFC5905]
  - `DCF` – *LF Radio DCF77 Mainflingen, DE 77.5 kHz* [RFC5905]
  - `HBG` – *LF Radio HBG Prangins, HB 75 kHz* [RFC5905]
  - `MSF` – *LF Radio MSF Anthorn, UK 60 kHz* [RFC5905]
  - `JJY` – *LF Radio JJY Fukushima, JP 40 kHz, Saga, JP 60 kHz* [RFC5905]
  - `LORC` – *MF Radio LORAN C station, 100 kHz* [RFC5905]
  - `TDF` – *MF Radio Allouis, FR 162 kHz* [RFC5905]
  - `CHU` – *HF Radio CHU Ottawa, Ontario* [RFC5905]
  - `WWV` – *HF Radio WWV Ft. Collins, CO* [RFC5905]
  - `WWVH` – *HF Radio WWVH Kauai, HI* [RFC5905]
  - `NIST` – *NIST telephone modem* [RFC5905]
  - `ACTS` – *NIST telephone modem* [RFC5905]
  - `USNO` – *USNO telephone modem* [RFC5905]
  - `PTB` – *European telephone modem* [RFC5905]

- `Reference Timestamp` – označava vrijeme kada je zadnji puta sat postavljen ili ispravljen, u 64-bitnom obliku vremenskog formata (pr. `Sep 21, 2004 17:44:05.064547999 UTC`).

- `Origin(Originate) Timestamp` – označava vrijeme u koje je zahtjev poslan od klijenta prema poslužitelju, u 64-bitnom obliku vremenskog formata (pr. `Sep 27, 2004 03:18:04.922896299 UTC`).

- `Receive Timestamp` – označava vrijeme u kojem je zahtjev klijenta stigao na poslužitelj u 64-bitnom obliku vremenskog formata (pr. `Sep 27, 2004 03:18:03.848507999 UTC`).

- `Transmit Timestamp` – označava vrijeme u kojem poslužitelj odgovara, u 64-bitnom obliku vremenskog formata (pr. `Sep 27, 2004 03:18:03.848500099 UTC`).

**Izvori informacija: (440),(1049), RFC5905.**

## 26.6.2. Osnovna konfiguracija

**NTP** servis obično standardno dolazi instaliran na sustav, kao ***ntpd*** servis.
Međutim u slučajevima kada nije instaliran, možemo ga instalirati sa sljedećom naredbom:

```
yum -y install ntp
```

Nakon instalacije ovaj servis neće se automatski aktivirati niti pokrenuti. Ali prvo ćemo prilagoditi njegovu konfiguracijsku datoteku, u kojoj ćemo prvo definirati vanjske NTP izvore (poslužitelje) na koje se on treba spajati.
Stoga otvorimo njegovu konfiguracijsku datoteku `/etc/ntp.conf`
Za sada nas zanima samo dio u kojemu se konfiguriraju NTP poslužitelj, odnosno dio koji počinje s ključnom riječi `server`.
Mi ćemo dodati sâmo sljedeći redak, a sve ostale (NTP) poslužitelje (*servere*) ćemo obrisati:

```
server os.ntp.carnet.hr primary
```

Dakle ovime smo dodali *CARNET*-ov NTP poslužitelj `os.ntp.carnet.hr` kao jedini (i primarni) mrežni NTP izvor.
Sada možemo pokrenuti NTP servis:
Za ***RedHat/CentOS* 6**.x. to postižemo sa:

```
service ntpd start
```

Odnosno za ***RedHat/CentOS* 7**.x/**8.**x+ NTP servis pokrećemo ovako:

```
systemctl start ntpd
```

U slučaju kada želimo da se naš NTP servis pokreće tijekom svakog pokretanja sustava, tada:
Za ***RedHat/CentOS* 6**.x moramo napraviti:

```
chkconfig ntpd on
```

Odnosno na ***RedHat/CentOS* 7**.x trebamo napraviti sljedeće:

```
systemctl enable ntpd
```

Sada kada smo pokrenuli NTP servis on će se pokušati spojiti na NTP poslužitelj koji smo upisali: `os.ntp.carnet.hr` čija IP adresa je: `161.53.30.104`.

Programom ***Wireshark*** ili `tcpdump` ćemo promatrati samo NTP promet (port 123). I dobit ćemo sljedeće inicijalno stanje u kojem naše računalo šalje NTP paket prema NTP poslužitelju. Bazirajmo se samo na (donji) NTP dio paketa:

```
Frame 1: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
Ethernet II, Src: 08:00:27:ec:36:32, Dst: d4:76:ea:2e:70:7f
Internet Protocol Version 4, Src: 192.168.1.162, Dst: 161.53.30.104
User Datagram Protocol, Src Port: 123, Dst Port: 123
Network Time Protocol (NTP Version 4, client)
   Flags: 0xe3, Leap Indicator: unknown (clock unsynchronized), Version number: NTP
Version 4, Mode: client
    Peer Clock Stratum: unspecified or invalid (0)
    Peer Polling Interval: 6 (64 sec)
    Peer Clock Precision: 0.000000 sec
    Root Delay: 0 seconds
    Root Dispersion: 0 seconds
    Reference ID: (Initialization)
    Reference Timestamp: Jan  1, 1970 00:00:00.000000000 UTC
    Origin Timestamp: Jan  1, 1970 00:00:00.000000000 UTC
    Receive Timestamp: Jan  1, 1970 00:00:00.000000000 UTC
    Transmit Timestamp: Nov  1, 2018 17:36:37.294384023 UTC
```

Vidimo kako smo i objasnili u teoriji: da naš NTP klijent šalje u `Transmit Timestamp` svoje vrijeme (svoj sat).

Potom nam *NTP* poslužitelj odgovara sa sljedećom porukom:

```
Frame 2: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
Ethernet II, Src: d4:76:ea:2e:70:7f, Dst: 08:00:27:ec:36:32
Internet Protocol Version 4, Src: 161.53.30.104, Dst: 192.168.1.162
User Datagram Protocol, Src Port: 123, Dst Port: 123
Network Time Protocol (NTP Version 4, server)
   Flags: 0x24, Leap Indicator: no warning, Version number: NTP Version 4, Mode: server
    Peer Clock Stratum: secondary reference (2)
    Peer Polling Interval: 6 (64 sec)
    Peer Clock Precision: 0.000001 sec
    Root Delay: 0.0040130615234375 seconds
    Root Dispersion: 0.0371551513671875 seconds
    Reference ID: 161.53.123.8
    Reference Timestamp: Nov  1, 2018 17:20:28.593177630 UTC
    Origin Timestamp: Nov  1, 2018 17:36:37.294384023 UTC
    Receive Timestamp: Nov  1, 2018 17:36:37.308346811 UTC
    Transmit Timestamp: Nov  1, 2018 17:36:37.308381102 UTC
```

Na dnu vidimo polja koja smo objasnili: `Origin Timestamp`, `Receive Timestamp` i `Transmit Timestamp` s kojima nam *NTP* poslužitelj daje podatke o svom točnom vremenu (satu) kao i kašnjenju našeg paketa.

Sveukupno iz svih vidljivih polja se izračunavaju sve potrebne vrijednosti odnosno odstupanja u vremenu.

A sada pogledajmo sve to, ali preračunato od strane našeg NTP klijenta, koje će nam dati naredba ntpq, na sljedeći način:

```
ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
================== ============== ==============================================
*os.ntp.carnet.hr   161.53.123.8    2 u   57   64  377   22.669    1.087   0.246
```

Vidljivo je kako je naš NTP poslužitelj `os.ntp.carnet.hr` odmah odabran za upotrebu, što je vidljivo prema oznaci `*` u stupcu `remote`. Vidimo i da se navedeni NTP poslužitelj predstavio da pripada stratumu dva (`st 2`) odnosno iz mrežnog paketa je to vidljivo iz `Peer Clock Stratum`.

Većina ostalih vidljivih vrijednost iz mrežnog paketa se preračunavaju i ispisuju pomoću naredbe `ntpq -p`.

Međutim, ako želimo vidjeti više podataka o konkretnom NTP poslužitelju na koji se spaja naš NTP klijent to možemo napraviti i iz naredbe ntpq na sljedeći način. Dakle prvo ju pokrenimo:

```
ntpq
```

Sada ćemo dobiti novi naredbeni redak u kojem trebamo pozvati naredbu `as` koja će nam ispisati sve dostupne NTP poslužitelje koje smo konfigurirali (uz svaki od njih stoji njegov identifikacijski broj koji se nalazi u drugom stupcu [`assid`]):

```
ntpq>as

ind assid status  conf reach auth condition  last_event cnt
===========================================================
  1 38892  963a   yes   yes  none  sys.peer    sys_peer  3
```

Vidimo da je naš *CARNET*-ov NTP poslužitelj koji je aktivan (`condition` je `sys.peer`) i ima identifikacijski broj: `38892`.

Sada ćemo za taj identifikacijski broj (`38892`) zatražiti više detalja, pomoću naredbe `rv` uz navođenje tog broja:

```
ntpq> rv 38892
associd=38892 status=963a conf, reach, sel_sys.peer, 3 events, sys_peer,
srcadr=os.ntp.carnet.hr, srcport=123, dstadr=192.168.1.162, dstport=123,
leap=00, stratum=2, precision=-20, rootdelay=4.013, rootdisp=28.717,
refid=161.53.123.8,
reftime=df85bca6.f6edd1cd  Thu, Nov  1 2018 18:55:50.964,
rec=df85bf4b.88e3413a  Thu, Nov  1 2018 19:07:07.534, reach=377,
unreach=0, hmode=3, pmode=4, hpoll=6, ppoll=6, headway=20, flash=00 ok,
keyid=0, offset=1.678, delay=22.955, dispersion=1.408, jitter=0.508,
xleave=0.059,
filtdelay=    22.95   23.06   23.62   23.81   23.47   23.17   23.76   23.59,
filtoffset=    1.68    1.39    1.36    1.20    1.26    1.07    1.18    0.90,
filtdisp=      0.00    0.99    1.98    2.96    3.93    4.89    5.85    6.86
```

Za sve one koji žele više istraživati, ovo je pravo mjesto. Mi ćemo ipak krenuti dalje s primjerima i drugim detaljima.

Pogledajmo slučaj u kojemu imamo problematični sekundarni NTP poslužitelj kako je vidljivo s ntpq naredbom:

```
ntpq -p

     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*os.ntp.carnet.hr   161.53.123.8    2 u   57   64  377   22.669    1.087   0.246
xzg1.ntp.carnet.hr  161.53.123.5    2 u   57   64  42    26.123    125841  0.246
```

Naime za drugi NTP poslužitelj (`zg1.ntp.carnet.hr`) vidimo da ima početnu oznaku (**x**) koja označava kako je on proglašen kao „*false ticker*“ odnosno nepouzdan izvor, po tome što (trenutno) ima nenormalno veliki ***offset*** (`125841`).
Ovaj izvor stoga neće biti u upotrebi sve dok se ne stabilizira, a na što će paziti sâm NTP klijent. Dodatno ovaj NTP izvor ima i mrežnih problema odnosno potencijalno mi imamo problema prema njemu, a što je vidljivo u stupcu `reach`. Naime u ovom polju imamo vrijednost `42` oktalno što je binarno 00100010, a što znači kako je od zadnjih osam paketa od njega zaprimljeno njih sâmo dva, i to ona dva s jedinicama (drugi i šesti konkretno). Što kada se bez obzira na potencijalne probleme s aplikacijama koje su problematične, naglo promijeni sistemski sat?. Što ako ipak na silu odnosno naglo želimo uskladiti naš sistemski sat s referentnim NTP izvorom?.

Zamislim slučaj u kojem naš sistemski sat kasni deset (10) minuta u odnosu na NTP referentni sat i ne želimo čekati da se normalnom NTP metodom sat polako usklađuje jer će to trajati previše vremena.
Ne zaboravimo kako je usklađivanje vrlo postepeno kako se ne bi poremetili programi kojima bi nagla promjena vremena uzrokovala probleme. Stoga možemo napraviti sljedeće. Prvo zaustaviti *NTP servis*:

```
service ntpd stop
```

Potom pomoću naredbe ntpdate ručno i trenutno uskladiti naš sistemski sat s udaljenim referentnim NTP izvorom: primjerice s os.ntp.carnet.hr, što ćemo postići s naredbom:

```
ntpdate os.ntp.carnet.hr
```



*Moguće je i sâmo provjeriti status udaljenog NTP izvora, ne bi li vidjeli šalje li nam ispravno točno vrijeme, ali bez korekcije lokalnog sata. To možemo postići pomoću naredbe:* `ntpdate -q os.ntp.carnet.hr`

A sada ponovno možemo pokrenuti *NTP* servis, kako bi on nastavio usklađivati točno vrijeme:

```
service ntpd start
```

Sada kada smo usklađeni, naš NTP servis svako malo će se usklađivati s udaljenim referentnim NTP izvorom u malim koracima, što više neće biti problem, jer tada više neće biti velikih odstupanja.



U slučaju kada je razlika u vremenu prevelika, NTP klijent se uopće neće moći uskladiti s udaljenim referentnim NTP izvorom pa je gore navedenu metodu svakako potrebno odraditi. Maksimalna standardna vrijednost odstupanja klijenta je 1.000 sekundi (~16.minuta). Dakle, ako sat na klijentu odstupa više od 16 minuta NTP standardno neće **NITI** pokušati uskladiti vrijeme.

Međutim moguće je narediti *ntp* servisu da sâmo i isključivo u trenutku pokretanja ignorira ovo ograničenje od 1.000 sekundi i to s prekidačima `-g` i `-x` koje je potrebno dodati u njegovu inicijalizacijsku skriptu: /etc/sysconfig/ntpd pa bi onda ona izgledala ovako (kao na ispisu dolje). Iako je ona obično tako i postavljena, bolje ju provjerite.

```
OPTIONS="-x -u ntp:ntp -p /var/run/ntpd.pid -g"
```

Dodatno postoje dvije opcije koje možemo navesti u konfiguraciji za svaki udaljeni NTP izvor (poslužitelj) zasebno u konfiguracijskoj datoteci /etc/ntp.conf, a to su opcije koje definiraju minimalni i maksimalni *poolling* interval koji će NTP koristiti za povezivanje s udaljenim referentnim NTP izvorom (poslužiteljem). Naime NTP klijent kontaktira udaljeni NTP poslužitelj obično svake 64 sekunde definirane na početku i onda kako se sve više usklađuje sa udaljenim NTP izvorom odnosno poslužiteljem, smanjuje vrijeme kontaktiranja na nekih 17 minuta ($2^{10}$ sekundi).

Donje granično vrijeme od kojeg se kreće je definirano u opciji `minpoll`, a gornje granično vrijeme do kojega se dolazi kasnije, definirano je u opciji `maxpoll`. Pogledajmo ove opcije postavljene za naš poslužitelj `os.ntp.carnet.hr` u datoteci: /etc/ntp.conf

```
server os.ntp.carnet.hr minpoll 4 maxpoll 10
```

Ovdje smo definirali da će se krenuti s $2^4$ odnosno 16 sekundi i postepeno će se doći do $2^{10}$ odnosno 17 minuta.

Postoji još nekoliko opcija koje ćemo ovdje spomenuti:

- `prefer` – označava primarni NTP poslužitelj. Važno je postaviti barem jedan od NTP poslužitelja kao *prefer* odnosno onaj koji će se uvijek prvi kontaktirati i to ako ih imamo više definiranih koji pripadaju istom *stratumu*.
- `iburst` – označava kako će se stalno, a pogotovo važno u slučaju nedostupnosti nekog udaljenog NTP poslužitelja, poslati „*burst*" odnosno nîz od osam (8) paketa na mrežu, prema njemu, u razmaku od dvije (2) sekunde. Dakle svake dvije sekunde će se poslati po jedan paket, a ukupno njih osam. S time smo osigurali da, ako je udaljeni NTP poslužitelj bio preopterećen i nije nam odgovorio na naš standardno poslan jedan paket, da će sigurno odgovoriti na barem jedan od njih osam poslanih, pa neće biti izbačen s liste NTP poslužitelja koje smo konfigurirali. U slučaju normalne dostupnosti udaljenog NTP poslužitelja ova metoda osigurava brže usklađivanje lokalnog vremena

Sada pogledajmo konačnu konfiguraciju našeg referentnog NTP poslužitelja u datoteci: /etc/ntp.conf:

```
server os.ntp.carnet.hr iburst minpoll 4 maxpoll 10 prefer
```



Novije inačice **RedHat/CentOS 7.x** umjesto ntpd servisa koriste servis chronyd i pripadajuću naredbu chronyc, a koji se konfigurira u datoteci: /etc/chrony.conf.

***Chrony servis bolje reagira na velika odstupanja u vremenu i brže se sinkronizira u tom slučaju.***
***O njemu više u narednom poglavlju.***

**Izvori informacija: (440),(441),(442),(443),(444),(445),(446),(447),(448),(449),(450), `man ntpd`, `man ntpq`, `man ntp.conf`.**

### 26.6.3. Chrony servis

***hrony*** je zadani poslužitelj za *Network Time Protocol* (**NTP**) u operativnim sustavima baziranim na **Red Hat/CentOS *8x***.
On je zamjena za stari ***ntpd*** servis koji se koristio u starijim inačicama operativnih sustava Linux (i Unix).
Koristi se za sinkronizaciju sata sustava s udaljenim internetskim poslužiteljima vremena odnosno NTP poslužiteljima.

***Chrony*** bolje i brže reagira na veća odstupanja u vremenu te se brže sinkronizira s udaljenim NTP poslužiteljima, što je osobito korisno za virtualna računala. Osim toga ima bolju stabilnost kada se radi o privremenim asimetričnim kašnjenjima u mreži, primjerice kada je veza zasićena velikim preuzimanjem sadržaja (*download*). ***Chrony*** se sastoji od dvije komponente: `chronyc` i `chronyd`.
Pri tome je ***Chronyd*** servis koji se pokreće prilikom pokretanja sustava. Dok je ***chronyc*** sučelje naredbenog retka koje se koristi za praćenje performansi i provjeru rada ***chronyd*** i NTP poslužitelja.

U slučajevima kada ***chrony*** nije instaliran, možemo ga instalirati sa sljedećom naredbom:

```
yum -y install chrony
```

Nakon instalacije ovaj servis možda se neće automatski aktivirati niti pokrenuti. Stoga ga aktivirajmo i pokrenimo:

```
systemctl enable chronyd
systemctl start chronyd
```

Opcije i parametri ***chrony*** servisa spremaju se u konfiguracijsku datoteku `/etc/chrony.conf`.

Druge opcije su iste kao i za ***ntpd*** servis, od kojih ćemo navesti samo njih nekoliko:

- `server` ***XY***- označava NTP poslužitelj na IP adresi ili s *FQDN* imenom ***XY***.
- `pool` ***XY***– označava *FQDN* adresu iza koje se može nalaziti više NTP poslužitelja. Naime ova opcije slična je onoj za direktivu poslužitelja (`server`), osim što se koristi za specificiranje skupa NTP poslužitelja umjesto jednog NTP poslužitelja. Očekuje se da će se ime (*FQDN*) razriješiti na više adresa koje bi se mogle promijeniti tijekom vremena.

Slijede opcije, koje možemo koristiti i za `server` i za `pool`:

- `prefer` – označava primarni NTP poslužitelj. Važno je postaviti barem jedan od NTP poslužitelja kao *prefer* odnosno onaj koji će se uvijek prvi kontaktirati i to ako ih imamo više definiranih koji pripadaju istom *stratumu*.
- `iburst` – označava kako će se stalno, a pogotovo važno u slučaju nedostupnosti nekog udaljenog NTP poslužitelja, slati nîz od četiri do osam paketa na mrežu, prema njemu, u razmaku od dvije (2) sekunde. Dakle svake dvije sekunde će se poslati po jedan paket, a ukupno njih osam. S time smo osigurali da, ako je udaljeni NTP poslužitelj bio preopterećen i nije nam odgovorio na naš standardno poslan jedan paket, da će sigurno odgovoriti na barem jedan od njih osam poslanih, pa neće biti izbačen s liste NTP poslužitelja koje smo konfigurirali. U slučaju normalne dostupnosti udaljenog NTP poslužitelja ova metoda osigurava brže usklađivanje lokalnog vremena.
- `burst` – uz ovu opciju, ***chronyd*** će poslati niz do četiri NTP zahtjeva prema NTP poslužiteljima. Ova opcija je blaža od opcije `iburst`.
- `minpoll` ***XY***– označava donje granično vrijeme (***XY***) dohvaćanja od kojeg se kreće. Primjerice `minpoll 4` znači da se kreće s $2^4$ odnosno već nakon 16 sekundi.
- `maxpoll` XY – označava gornje granično vrijeme (***XY***) dohvaćanja do kojega se dolazi kasnije. Primjerice `maxpoll 10` znači da će se postepeno doći do $2^{10}$ odnosno do vremena dohvaćanja od maksimalno 17 minuta.

Mi ćemo sada prilagoditi njegovu konfiguracijsku datoteku, u kojoj ćemo prvo definirati vanjske NTP izvore (poslužitelje) na koje se on treba spajati, (gotovo) na identičan način i sa sličnim opcijama i parametrima koje je koristio ***ntpd*** servis.
Stoga otvorimo njegovu konfiguracijsku datoteku `/etc/chrony.conf`. Nakon izmjena ne zaboravite restartati `chronyd` servis: **`systemctl restart chronyd`**. Za sada nas zanima samo dio u kojemu se konfiguriraju NTP poslužitelj.
Za razliku od ***ntpd*** servisa, preporuka je koristiti *pool* NTP poslužitelje, a ne pojedine poslužitelje odnosno dio koji počinje s ključnom riječi `server`.

Mi ćemo dodati sâmo sljedeće retke, a sve ostale (NTP) poslužitelje (*servere*) ćemo obrisati:

```
pool 0.hr.pool.ntp.org iburst prefer
pool 2.europe.pool.ntp.org iburst
```

Dakle ovime smo dodali navedene NTP poslužitelje kao mrežne NTP izvore.

Nadalje, više se ne koristi naredba `dtpdate`, a također više nema potrebe niti za upotrebom naredbe `ntpq`.

Umjesto njih, sve njihove značajke su ujedinjene u naredbu `chronyc`.

S njom provjerimo na koje NTP poslužitelje smo spojeni.

```
Chronyc  sources

MS  Name/IP address          Stratum Poll Reach LastRx Last sample
===============================================================================
^+  m03.colo.bontekoe.techno>     3    6   177     40  -1204us[-1204us] +/-   60ms
^+  time.cloudflare.com           3    6   377     64  +1056us[+1056us] +/-   42ms
^*  ntp1.cratis.cc                2    6   377     64  +2312us[+3280us] +/-   35ms
^+  m02.colo.bontekoe.techno>     3    6   377     64  -2018us[-2018us] +/-   71ms
^-  185.83.169.27                 1    6   377     64  -2648us[-1679us] +/-   30ms
^-  tilia.zsx.hu                  2    6   377      7  +4192us[+4192us] +/-   50ms
^-  inter.tyjo.eu                 2    6   377      7  +5255us[+5255us] +/-   24ms
^-  82-64-45-50.subs.proxad.>     1    6   377      8   +705us[ +705us] +/-   29ms
```

Pogledajmo što znače vidljivi stupci:

- `MS` – označava status veze prema udaljenom NTP poslužitelju, ukratko: * označava aktivni, a + pričuvne.
- `Stratum` – označava kojem *Stratumu* pripada određeni NTP poslužitelj.
- `Pol` – označava nakon koliko sekundi će se dohvatiti podaci s određenog NTP poslužitelja.
- `Reach` – označava dostupnost određenog NTP poslužitelja. Slijede drugi stupci identični starim naredbama.



Za sve navedene statusne oznake i detalje, pogledajte poglavlje:
**26.6. NTP**.

Naredba `chronyc` ima desetak parametara s kojima ju možemo pozvati, a sve dostupne možete dobiti pokretanjem naredbe `chronyc` s pritiskanjem tipke ***TAB*** nakon toga. Neki od njenih korisnih parametara su: `clients` (*za ispis NTP klijenata*), `ntpdata` (*za detalje o NTP poslužiteljima*), `sources` (*za listu NTP poslužitelja*), `sourcestats` (*za statistike za svaki NTP poslužitelj*), `tracking` (*za statistike od aktivnog NTP poslužitelja*) i druge.

Osim toga nju možemo pokrenuti i bez parametara, pa će nam se otvoriti njen naredbeni redak, slično kao kod naredbe `ntpq`.

**Chrony** je moderni NTP (*Network Time Protocol*) klijent i poslužitelj koji pruža nekoliko prednosti u odnosu na tradicionalni **NTP** servis (***ntpd***).

Neke od ključnih prednosti Chrony servisa su:

- ***Brža i točnija sinkronizacija*** – *Chrony* može brže sinkronizirati sistemski sat s referentnim NTP poslužiteljem. To je posebno korisno za sustave koji se često isključuju ili imaju neredovite veze s mrežom. On postiže veću točnost u sinkronizaciji vremena, čak i u okruženjima s nestabilnim mrežnim vezama.
- ***Bolja podrška za mrežna okruženja s visokim latencijama i jitterom*** – *Chrony* je dizajniran da radi učinkovito u mrežnim okruženjima s velikim latencijama i fluktuirajućim kašnjenjima (*jittering)*, što može biti izazov (problem) za tradicionalni NTP servis.
- ***Rad u offline modu*** – *Chrony* može održavati točno vrijeme čak i kada je sustav offline, koristeći hardverski sat (**RTC**) za korekcije. Ovo je korisno za prijenosne uređaje i poslužitelje koji se često isključuju.
- ***Brza prilagodba velikih promjena vremena*** – *Chrony* može brzo prilagoditi velike razlike između lokalnog i referentnog vremena, bez potrebe za naglim skokovima koji bi mogli uzrokovati probleme u aplikacijama osjetljivim na vrijeme.
- ***Bolje upravljanje vremenskim odmacima*** – *Chrony* može preciznije upravljati vremenskim odmicanjem sistemskog sata, koristeći statističke metode za predviđanje i korekciju odmicanja (*drift-a*).
- ***Manja potrošnja resursa*** – *Chrony* često manje opterećuje sistemske resurse u usporedbi s **ntpd**, što ga čini pogodnim za upotrebu na ugrađenim sustavima i poslužiteljima s ograničenim resursima.
- ***Napredne funkcionalnosti za kontrolu i nadzor*** – *Chrony* pruža bogatiji skup alata za praćenje i upravljanje, uključujući mogućnost ispitivanja statusa sinkronizacije i prilagođavanja postavki u hodu.
- ***Široka podrška za razne operativne sustave*** - *Chrony* podržava širok spektar operativnih sustava, uključujući različite verzije Linux-a, BSD-a i drugih UNIX-olikih sustava.
- …

**Izvori informacija: (440),** `man chronyc`, `man chronyd`, `man chrony.conf`.

# 26.7. Vatrozid (*Firewall*)

Vatrozid ili engl. *firewall* predstavlja uređaj ili tehnologiju, a u našem slučaju linux servis koji prati i kontrolira mrežni promet na razini svakog mrežnog paketa. Sljedeći korak rada i osnovna funkcionalnost vatrozida je donošenje odluke na osnovi zadanih kriterija: što napraviti sa svakim mrežnim paketom koji dođe do našeg računala odnosno preciznije, svake pojedine mrežne kartice odnosno mrežnog sučelja: logičkog (`br0`, `bond0`,…) ili fizičkog (pr. `eth0`, `eno0`, …).
Naime ovisno o postavkama vatrozida, on svaki mrežni paket može:

- Proslijediti dalje, odbaciti ga ili blokirati.
- Napraviti neku dodatnu radnju odnosno akciju.

Možemo reći da je funkcionalnost vatrozida kontroliranje mrežnih paketa koji dolaze, prolaze ili izlaze kroz naše računalo odnosno kroz mrežne slojeve unutar kernela operativnog sustava Linux. Vatrozid koristimo kada želimo ograničiti pristup na neki mrežni servis odnosno mrežni protokol. Primjerice želimo li ograničiti pristup za *SSH* protokol na našem poslužitelju i to tako da dozvoljavamo *SSH* promet sâmo s određenih IP adresa koje su unutar naše mreže. Drugi primjer može biti ograničenje broja otvorenih konekcija na recimo njih 100 u sekundi na naš web poslužitelj na TCP portu 80 (*http* protokol). Nadalje možemo zabraniti sav mrežni promet prema našem poslužitelju, sa strane interneta, osim za HTTP protokol (TCP port 80) i tako dalje. U pozadini je za svo filtriranje i obradu paketa na najnižoj razini zadužen linux kernel koji za svaki pojedini mrežni protokol koristi ***Netfilter*** programski okvir (*framework)* koji omogućava dublje baratanje s mrežnim paketima. ***Netfilter*** potom za svaki pojedini mrežni protokol ili funkcionalnost poziva zasebne *Netfilter* kernel module koji potom i odrađuju specifične zadatke. Mi ćemo u ovoj cjelini raditi sa servisom i naredbom `iptables` koja se također u pozadini oslanja na ***Netfilter*** sustav i pripadajuće kernel module koji su potrebni za njegov rad, poput centralnog modula `ip_tables` i drugih specifičnih, kao što je: `iptable_nat`.


Za napredne parametre pri inicijalizaciji ovog servisa, pogledajte poglavlje:
**7.4. Upravljanje konfiguracijom servisa** i to konfiguracijsku datoteku `/etc/sysconfig/iptables-config`.

## 26.7.1. Kako koristiti vatrozid

Pošto je vatrozid već instaliran s vašim Linuxom, potrebno ga je sâmo pokrenuti i koristiti. Primjeri koji slijede koriste takozvani ***iptables*** vatrozid, a o drugim implementacijama vatrozida ćemo govoriti u nekim od sljedećih poglavlja. Za ručno pokretanje vatrozida potrebno je napraviti sljedeće.Za **RedHat/CentOS 6.x** za pokretanje ***iptables*** vatrozida napravite sljedeće:

```
service iptables start
```

Za ručno zaustavljanje ***iptables*** vatrozida, potrebno je pokrenuti sljedeću naredbu:

```
service iptables stop
```

Odnosno za ručni restart ***iptables*** vatrozida:

```
service iptables restart
```

Ako želimo da se ***iptables*** vatrozid pokreće tijekom pokretanja Linuxa, napravimo sljedeće:

```
chkconfig –level 345 iptables on
```

Možete se podsjetiti kako se upravlja sa servisima jer je ovo standardna metoda automatskog pokretanja.
Ili ako **ne** želimo da se *iptables* vatrozid pokreće kod podizanja Linuxa, napravimo sljedeće:

```
chkconfig –level 345 iptables off
```

***Iptables*** servis vatrozida ne može raditi bez kernel modula `ip_tables`, koji ako se nije učitao, možete ga ručno učitati sa:

```
modprobe ip_tables
```

Za **RedHat/CentOS 7.x+/8.x/9.x+** stanje je malo drugačije.
Naime *RedHat/CentOS* 7.x standardno ne koristi ***iptables*** servis već noviji koji se zove `firewalld`, a koji u pozadini isto koristi pristup ***Netfilter programskom okviru (****Netfilter framework****)***, kao i `iptables`. Osim toga on nudi i neke naprednije mogućnosti koje *iptables* nije nudio (niti nudi): poput dinamičkih promjena (učitavanja) pravila, upotrebu zona ili takozvanu ***ipset*** funkcionalnost. Mi ćemo proći kroz primjere konfiguracije **iptables** vatrozida, zbog kompatibilnosti unatrag, kao i kompatibilnosti s drugim distribucijama Linuxa koje ga koriste.
Pošto u ovom primjeru radimo na **RedHat/CentOS 7.x+/8+**, stariji servis `iptables` je moguće naknadno instalirati, što ćemo i napraviti sa sljedećom naredbom:

```
yum -y install iptables-services
```

Potom ćemo (za trenutne potrebe) trajno zaustaviti noviji servis `firewalld`:

```
systemctl disable firewalld
systemctl stop firewalld
```

Zatim ćemo pokrenuti servis `iptables` kako bismo u potpunosti bili kompatibilni (u uputama) sa *RedHat/CentOS* 6.x.

```
systemctl start iptables
```

Potom ćemo učiniti da se ovaj servis i sâmim time (ovaj stariji [*iptables*]) vatrozid pokreće sa svakim pokretanjem sustava:

```
systemctl enable iptables
```

**Izvori informacija: (721),(722),** **`man iptables`.**

## 26.7.2. Kako radi Linux vatrozid *iptables*

*Sve implementacije Linux vatrozida oslanjaju se na **Netfilter** podsustav (o njemu kasnije).*
***Iptables*** funkcionalnost vatrozida se sastoji od nekoliko vrsta tablica (engl. *tables*) koje se potom sastoje on nîzova ili tokova podataka (engl. *chain*), a na svaki od njih se primjenjuju određena pravila rada (engl. *rules*).
Ta pravila su u konačnici ono što čini jezgru vatrozida. Navedene tablice su podijeljene u nekoliko kategorija:

- **Raw** – koje uglavnom sadrže mehanizme za označavanje i/ili ranu obradu paketa.
- **Mangle** – radi razne promjene IP zaglavlja paketa (pr. polja: **ToS**, **TTL** fragmentacija i slično).
- **NAT** – koji je zadužen za translaciju adresa poput **NAT**, ***SNAT*** i ***DNAT***.
- **Filter** – koji je zadužen za odluke o filtriranju paketa. Sadrži pravila poput: *INPUT*, *OUTPUT* ili *FORWARD*.
- **Security** – zadužen za sigurnosne postavke te eventualno označavanje paketa za potrebe ***SELinux*** podsustava.

Operacije odnosno nîzovi (engl. *chains*) su podijeljeni u nekoliko grupa:

- **PREROUTING** – (*za radnje prije usmjeravanja*) – koriste se za **Raw**, **Mangle** i **NAT** tablice.
- **INPUT** – (*za ulazni promet*) – koriste se za **Mangle** i **Filter** tablice.
- **FORWARD** – (*za prosljeđivanje prometa*) – koriste se za **Mangle** i **Filter** tablice.
- **OUTPUT** – (*za izlazni promet*) – koriste se za **Raw**, **Mangle**, **NAT** i Filter tablice.
- **POSTROUTING** – (*za radnje nakon usmjeravanja*) – koriste se za **Mangle** i **NAT** tablice.

Za sada ćemo se fokusirati na **Filter** tablice. Filter tablica može koristiti tri operacije odnosno filtera ili lanca (engl. *chain*):

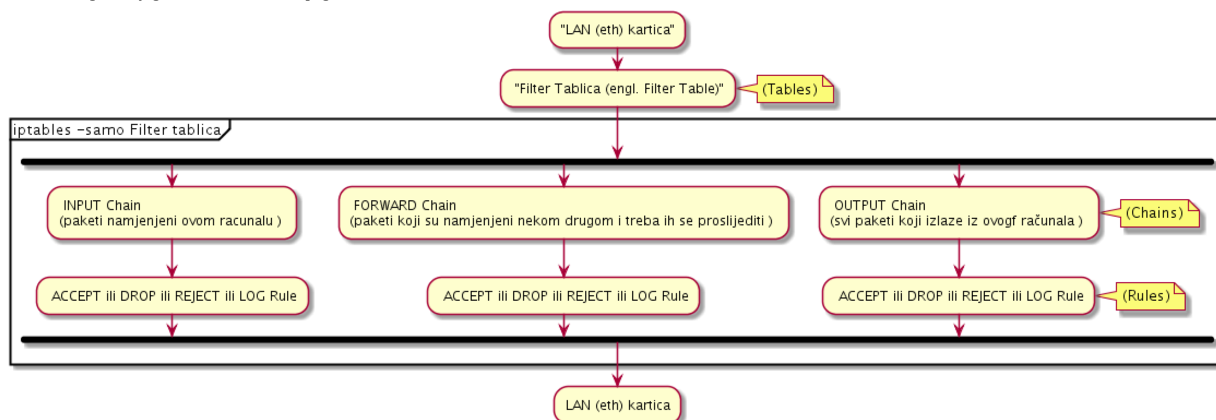
- **INPUT** – on je namijenjen za sve pakete namijenjene ovom računalu.
- **OUTPUT** – on je namijenjen za sve pakete koji izlaze iz ovog računala.
- **FORWARD** – on je namijenjen za sve pakete koji su zaprimljeni, ali nisu namijenjeni ovom računalu već bi ih ovo računalo trebalo proslijediti (engl. *Forward*) negdje dalje, odnosno nekom drugom računalu na mrežu.

Sada svaki paket mora proći kroz provjeru pravila (engl. *rules*) odnosno kriterije koji mogu biti:

- **ACCEPT** – paket je prihvaćen i ide dalje prema aplikaciji.
- **DROP** – paket se odbacuje i nikakve informacije se ne šalju pošiljaocu paketa.
- **REJECT** – paket se odbacuje, ali se pošiljaocu paketa šalje poruka o grešci.
- **LOG** – detalji o paketu se šalju na centralno mjesto za logiranje (obično `syslog` servis).
- **DNAT** – prepisuje se odredišna IP adresa "*Destination IP*" paketa (Engl. *Destination NAT*).
- **SNAT** – prepisuje se izvorišna IP "*Source IP*" paketa (Engl. *Source NAT*).

Pogledajmo logički dijagram tôka Linux vatrozida i to sâmo dijela **Filter** tablice:

**Slika 246. Logički dijagram linux vatrozida: pogled na** Filter **tablice.**




Potrebni su nam i prekidači kojima definiramo radnju za određeno pravilo:

- `-j` - (engl. *Jump*); podržava jednu od četiri naredbe: ***ACCEPT***(prihvati), ***REJECT***(NE prihvati), ***DROP***(odbaci) ili ***LOG.***
- `-m` - (engl. *Match*); provjerava jesu li zadovoljeni daljnji uvjeti (ima ih veći broj), a oni rade detaljniju provjeru stanja paketa, ograničenja i slično.

Osim gore navedene provjere postoje i dodatni kriteriji provjere:

- `-p` <protocol>; dodaje dodatnu definiciju protokola: *tcp*, *udp*, *icmp* i *all.*
- `-s` <ip_addr>; provjerava izvorišnu IP adresu odnosno "*Source* IP".
- `-d` <ip_addr>; provjerava odredišnu IP adresu odnosno "*Destination* IP".
- `--sport` <port>; provjerava izvorišni port odnosno "*Source* port".
- `--dport` <port>; provjerava odredišni port odnosno "*Destination* port".
- `-i` <interface>; provjerava mrežno sučelje s kojeg je paket **došao**.
- `-o` <interface>; provjerava mrežno sučelje s kojeg je paket **otišao**.

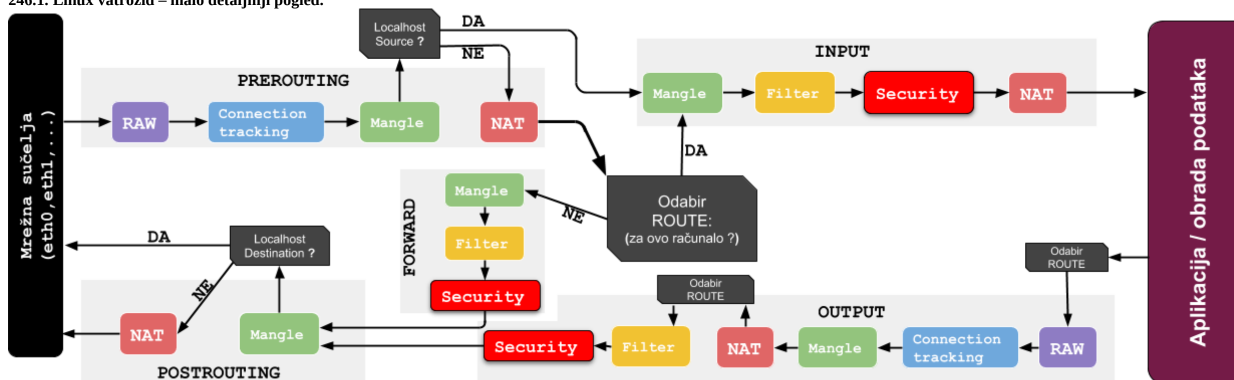
Sada kada znamo osnove, potrebno je razumjeti i načine dodavanja, brisanja i mijenjanja pravila *vatrozida*.

To postižemo sa:

- `-A` – nadodaj na postojeće pravilo vatrozida (engl. *Append*).
- `-D` – obriši pravilo vatrozida (engl. *Delete*).
- `-I` – dodaj/ubaci pravilo vatrozida (engl. *Insert*).
- `-R` – zamjeni pravilo vatrozida (engl. *Replace*).
- `-L` – izlistaj/ispiši pravila vatrozida (engl. List); `--line-numbers` – kod izlistavanja ispiši i redni broj pravila.

Kada kreiramo pravila (engl. *rule*) ona se dodaju prema redoslijedu prema kojem smo ih kreirali, prema brojevima.
Stoga nije svejedno koje pravilo je postavljeno prije kojega. Ako recimo prvo imamo pravilo da se svi paketi odbacuju (**DROP**), sva pravila nakon toga će postati nevažna jer će se svi paketi prvo odbacivati. Ali, ako s druge strane prvo dodamo pravila za pojedine protokole i slično, a tek na kraju pravilo da se sve ostalo odbacuje, tada imamo potpuno drugu priču.
Pogledajmo malo detaljniju logičku shemu rada Linux vatrozida, kako je vidljivo na slici **246.1.**

**246.1. Linux vatrozid – malo detaljniji pogled.**




**Izvor logičke sheme na osnovu koje je napravljena slika: (1050): https://stuffphilwrites.com/2014/09/iptables-processing-flowchart/**

**Izvori informacija: (725), (1050),(1051),(1311).**

---

## 26.7.2.1. Primjeri

**U ovoj cjelini vidjet ćemo kako ručno dodavati *IPTABLES* pravila.**
Dodavanje prava pristupa za pojedinu IP adresu za sve protokole. Dodajmo pravo IP adresi 10.10.10.1 da smije pristupiti našem Linux računalo preko bilo kojeg protokola (porta). Pogledajmo kako to napraviti pomoću naredbe `iptables`:

```
iptables -A INPUT -s 10.10.10.1/32 -j ACCEPT
```

Dodavanje prava pristupa za pojedinu IP adresu za sâmo točno određeni protokol (SSH), TCP port 22.
Dodajmo pravo IP adresi 10.10.10.1 da smije pristupiti našem Linux računalo, ali sâmo preko protokola SSH (TCP porta 22).
Dakle ovdje koristimo odredišni port odnosno "*Destination port*" 22 koji označava i mrežni servis/protokol (SSH).

```
Iptables -A INPUT -s 10.10.10.1/32 -p tcp -m tcp –dport 22 -j ACCEPT
```

Dodavanje prava pristupa za cijelu mrežu (20.20.20.0/24 [tj. mrežu 20.20.20.0 s maskom: 255.255.255.0]), za sve protokole.

```
Iptables -A INPUT -s 20.20.20.0/24 -j ACCEPT
```

Na kraju kada smo završili dodavanje prava svime kojima smo željeli, moramo kreirati pravilo da se sve ostalo odbacuje.
To ćemo napraviti sa sljedećom naredbom:

```
iptables -A INPUT -j DROP
```

Moramo biti svjesni činjenice da se pravila dodaju redoslijedom kojim ih i upisujemo.
Prema tome važno je koje pravilo primjenjujemo prije kojega, jer ih vatrozid i provjerava u istom redoslijedu.
**Sada probajmo obrisati sva pravila, privremeno sve do restarta računala:**

```
iptables --flush
```

Potom ćemo dodati nekoliko ulaznih pravila (`INPUT`) iz primjera **2.** i **3.** gore:

```
iptables -A INPUT -s 10.10.10.1/32 -p tcp -m tcp –dport 22 -j ACCEPT
iptables -A INPUT -s 20.20.20.0/24 -j ACCEPT
```

Pogledajmo kako sada izgledaju sva trenutno postavljena pravila, s ispisom njihovih rednih brojeva:

```
iptables -L --line-numbers -n

Chain INPUT (policy DROP)
num  target     1217roto pt source               destination
1    ACCEPT     tcp  --  10.10.10.1/32         0.0.0.0/0            tcp dpt:22
2    ACCEPT     all  --  20.20.20.0/24         0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     1217roto pt source               destination
Chain OUTPUT (policy ACCEPT)

num  target     1217roto pt source               destination
```

Na prijašnjem ispisu, vidljiv je redoslijed pravila (num) i pravila koja smo dodali (1 i 2). FORWARD i OUTPUT nismo dirali prema tome nema niti pravila za njih, te je za njih sve dozvoljeno bez ograničenja. Probajmo sada ubaciti novo pravilo između pravila 1 i 2. Razlika je u tome da nismo koristili prekidač -A (*Append* – dodaj) već -I (*Insert* – ubaci) te smo naveli redni broj pravila na čije mjesto se ubacujemo a staro previlo s tim rednim brojem pomičemo dolje.

Dodajmo i pravila u kojem računalu 10.10.10.1 dodajemo prava pristupa na TCP port 80 (HTTP).

```
Iptables -I INPUT 2 -s 10.10.10.1/32 -p tcp -m tcp -dport 80 -j ACCEPT
```

Pogledajmo sada našu novu trenutnu tablicu (ako želimo i ispis brojača paketa, možemo koristiti i prekidač -v):

```
iptables -L --line-numbers -n

Chain INPUT (policy DROP)
num  target     1218roto pt source               destination
1    ACCEPT     tcp  --  10.10.10.1/32         0.0.0.0/0           tcp dpt:22
2    ACCEPT     tcp  --  10.10.10.1/32         0.0.0.0/0           tcp dpt:80
3    ACCEPT     all  --  20.20.20.0/24         0.0.0.0/0
Chain FORWARD (policy ACCEPT)
num  target     1218roto pt source               destination
Chain OUTPUT (policy ACCEPT)
num  target     1218roto pt source               destination
```

**Brisanje pojedinog pravila**

Sada ipak obrišimo pravilo koje smo dodali (sada je to br.2.). Prekidač -D govori da želimo obrisati pravilo i to INPUT br. 2.

```
iptables -D INPUT 2
```

Pogledajmo ponovno tablicu s pravilima vatrozida:

```
iptables -L --line-numbers -n

Chain INPUT (policy DROP)
num  target     1218roto pt source               destination
1    ACCEPT     tcp  --  10.10.10.1/32         0.0.0.0/0           tcp dpt:22
2    ACCEPT     all  --  20.20.20.0/24         0.0.0.0/0
Chain FORWARD (policy ACCEPT)
num  target     1218roto pt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     1218roto pt source               destination
```

Vidljivo je da više nema starog pravila br. 2 već da je na njegovo mjesto došlo pravilo koje je bilo br. 3 a sada je postalo br.2.

**Snimanje *firewall* pravila koja smo ručno upisali, možemo postići sa:**

```
service iptables save
```

S gore navedenom naredbom snimili smo sva pravila koja smo ručno unijeli, te će biti aktivna kod sljedećeg restarta računala. Ova pravila se snimaju u datoteku /etc/sysconfig/iptables i to u malo drugačijem formatu nego ručno upisane naredbe. Konkretno, samo bez sâme iptables naredbe, uz istu sintaksu.

## NAT (*Network Address Translation*) + PAT ili SNAT (*Source NAT*) pravila

**Upotreba NAT-a**

U slučaju kada je naš Linux i centralni usmjerivač na kojemu želimo raditi ***NAT+PAT***, pri čemu mu je eth0 ***WAN*** sučelje prema internetu, a eth1 ***LAN*** sučelje (lokalna mreža [***192.168.2.0/24***]), napravimo sljedeće:

```
iptables -t nat -A POSTROUTING -s 192.168.2.0/24 -o eth0 -j MASQUERADE
```

Nakon ove konfiguracije, naš Linux kao usmjerivač će skrivati sve unutarnje IP adrese na mrežnom sučelju eth1 iz opsega 192.168.2.0/24 i pretvarati ih u svoju vanjsku IP adresu definiranu na **WAN** mrežnom sučelju eth0. Pogledajmo ***NAT*** tablicu:

```
iptables -t nat -L -n

Chain POSTROUTING (policy ACCEPT)
target      prot  opt    source              destination
MASQUERADE  all   --     192.168.2.0/24      0.0.0.0/0
```

Dakle vidimo da se u *postrouting* djelu radi ***NAT+PAT*** (Linux terminologija MASQUERADE) sa svih unutarnjih IP adresa iz opsega: 192.168.2.0/24 prema svim odredišnim odnosno vanjskim (javnim) adresama, što označava: 0.0.0.0/0.

---

**Upotreba SNAT-a**

Sa sljedećom konfiguracijom ćemo napraviti **SNAT** pravilo koje će za sav vanjski promet koji ulazi u lokalnu mrežu iz sučelja eth0 preko sučelja eth1 promijeniti izvorišnu (*source*) IP adresu u adresu eth1 sučelja: 192.168.2.1, tako da će računalima iza ovog Linux usmjerivača izgledati kao da promet dolazi s njega to jest s IP adrese 192.168.2.1.

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT -to 192.168.2.1
```

---

***Upotreba SNAT POOL-a***

Moguće je koristiti i više IP adresa istovremeno kao tzv. ***SNAT pool***, za povećanje broja mogućih aktivnih konekcija, ali pri tome ne možemo koristiti primarnu IP adresu. Zamislimo ***SNAT pool*** (192.168.2.2→192.168.2.20); koji se konfigurira ovako:

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT -to 192.168.2.2-192.168.2.20
```

---

Pokrenimo naredbu s kojom ćemo pogledati dio ***SNAT*** tablice (čisti ***SNAT*** bez ***SNAT poola***), s kojom ćemo vidjeti da se upravo odrađuje navedeni ***SNAT***:

```
iptables -t nat -L -n
```

Slijedi ispis navedene naredbe u kojem vidimo da se sav dolazni promet (s interneta) sakriva kao da dolazi s IP: ***192.168.2.1***.

```
Chain POSTROUTING (policy ACCEPT)
target     1219roto pt source               destination
SNAT       all  --  0.0.0.0/0            0.0.0.0/0           to:192.168.2.1
```

## Restart iptables vatrozida

*Iptables* vatrozid možemo restartati sljedećom naredbom za **RedHat/CentOS 6.x.**

```
service iptables restart
```

Odnosno za **RedHat/CentOS 7.x/8.**+ (i to ako koristimo ***iptables*** servis umjesto standardnog ***firewalld*** servisa)

```
systemctl restart iptables
```

U procesu restarta *iptables* servisa vatrozida on ponovno učitava sadržaj datoteke `/etc/sysconfig/iptables` te se primjenjuju sva pravila koja su upisana unutar ove datoteke (ako ih ima).



Servis `iptables` zadužen je sâmo za IPv4 protokol, dok za IPv6 protokol postoji servis koji se zove `ip6tables`.

Za dodatne funkcionalnosti Linux vatrozida, potrebno je učitati kernel module koji nam te funkcionalnosti donose.
Dakle ovdje se radi o kernel modulima koji se vežu za ***Netfilter*** podsustav linuxa, na koji se Linuxov vatrozid veže.
Ako želimo izlistati sve dostupne kernel module koji pripadaju ***Netfilter*** podsustavu, to možemo napraviti na sljedeći način:

```
ls -al /lib/modules/`uname -r`/kernel/net/netfilter/
```

Tako za primjerice iole detaljniju kontrolu i ograničavanje pristupa sustavu, definitivno želimo imati učitan kernel modul imena `nf_conntrack` jer nam on daje i dodatne mogućnosti praćenja mrežnog prometa. Sada ga i učitajmo na sljedeći način:

```
modprobe nf_conntrack
```

Tek sada su nam dostupne mnoge dodatne, a među njima i jedna vrlo važna funkcionalnost, na sustavima koji moraju baratati s vrlo velikim brojem (10.000+) aktivnih konekcija na sustav; što je važno za primjerice za *load balancere* (pr. ***Haproxy***), usmjerivače, *proxy* poslužitelje i slično. Naime tek sada možemo vidjeti, ali i mijenjati ograničenje na maksimalan broj aktivnih konekcija na naš sustav vatrozida. Ovdje se radi o *sysctl* varijabli imena: `net.netfilter.nf_conntrack_max`. Prvo pogledajmo njenu trenutno postavljenu vrijednost pomoću sljedeće *sysctl* naredbe:

```
sysctl net.netfilter.nf_conntrack_max

net.netfilter.nf_conntrack_max = 32768
```

Vidimo da se na naš sustav (računalo/poslužitelj) može ostvariti maksimalno ukupno `32.768` aktivnih konekcija, što se tiče kernela, a vezano za funkcionalnost vatrozida. Ovu vrijednost možemo povećati na slijedeći način (primjerice na `200.000`):

```
sysctl -w net.netfilter.nf_conntrack_max=200000
```

Za trajnu konfiguraciju napravimo novi unos u datoteci `/etc/sysctl.conf`. Ova vrijednost je vidljiva i u */proc* pseudo datotečnom sustavu u datoteci `/proc/sys/net/nf_conntrack_max`.



Vrijednost `net.netfilter.nf_conntrack_max` se odnosi na apsolutno sve aktivne mrežne konekcije na sustavu, ako koristimo vatrozid uz učitan kernel modul `nf_conntrack`. Međutim čak i ako ne koristimo vatrozid, a navedeni kernel modul je učitan, sustav će i dalje pratiti svaku konekciju pa će stoga i imati ovo ograničenje na njihov broj(772).

Ako pređemo njenu granicu, dobit ćemo poruku od sustava kako ne može otvoriti nove konekcije s porukom:

```
nf_conntrack: table full, dropping packet.
```



Vezano za `nf_conntrack` kernel modul pogledajte i poglavlja:


**7.4. Upravljanje konfiguracijom servisa.**
**19.8.1. TCP/UDP Portovi.**
**24.2.16.1. Timeri i stanja veze.**




Servisi: ***iptables***, ***iptables-nft*** i ***ipset*** su od inačice 9.x. ***RedHat*** Linuxa zastarjeli i zamijenjeni s: ***nftables***.

**Izvori informacija: (721),(722),(769),(770),(776),(1050), `man iptables`.**

## 26.7.3. Pogled na mrežni model vatrozida u linuxu i nove tehnologije

Vezano za vatrozid, u pozadini je za svo filtriranje i obradu paketa na najnižoj razini zadužen Linux kernel koji za svaki pojedini mrežni protokol koristi ***Netfilter*** programski okvir (*framework*) koji omogućava dublje baratanje s mrežnim paketima. ***Netfilter*** potom za svaki pojedini mrežni protokol ili funkcionalnost poziva zasebne *Netfilter* kernel module koji potom i odrađuju specifične zadatke.

Međutim komunikacija između aplikacija i svake druge komponente mrežnog sustava odvija se u slojevima. Na početku dolazimo do takozvanog ***Netlink*** sloja koji je specifičan po tome što se koristi za prijenos informacija između izoliranog prostora kernela i korisničkog adresnog prostora odnosno pokrenutih programa/aplikacija. Osim *Netlink* sučelja aplikacije s mrežnim sustavom mogu komunicirati i pomoću *sistemskih poziva*. Dakle i ***Netlink*** i ***sistemski pozivi*** sučelja su između korisničkih programa/aplikacija i internih funkcija kernela prema kernel modulima za specifične namjene, ali i drugim dijelovima kernela zaduženim za mrežu, te u konačnici i s upravljačkim programima za mrežne kartice i samom mrežom. Možemo to reći i ovako: ***Netlink*** utičnice (engl. ***socket***) su sučelja Linux kernela koje se koriste za komunikaciju između procesa (**IPC**) i to između procesa kernela i korisničkih programa (procesa), te između različitih procesa unutar korisničkog prostora sustava, na način sličan Unix domenskim utičnicama (engl. *Unix domain socket*). Nadalje ***Netlink*** nam daje i sučelje za komunikaciju prema ***Netfilter*** sustavu i drugim mrežnim sustavima (pr. mrežnom stôgu Linuxa) i mreži. ***Netlink*** komunikacija ne može preko mreže komunicirati s vanjskim entitetima, što znači da se koristi i može djelovati samo unutar računala. ***Netlink*** je definiran u **RFC3549**.

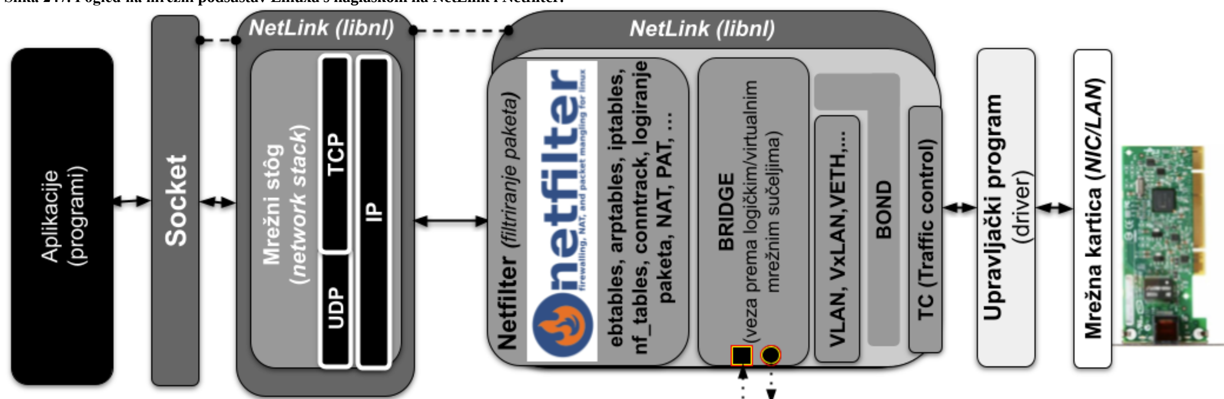
***Netfilter*** komponenta unutar mrežnog podsustava daje programski okvir (engl. *framework*) koji pruža Linux kernel, a koji omogućuje implementaciju raznih operacija vezanih za baratanje s mrežnim paketima. Konkretno *netfilter* nudi razne funkcije i operacije za filtriranje paketa, prevođenje mrežnih adresa (**NAT**) ili prevođenja portova (**PAT**) i slično, koje pružaju funkcionalnosti potrebne za usmjeravanje paketa kroz mrežu ili zabranu paketima da dosegnu osjetljiva mjesta unutar mreže.

To radi tako da ***Netfilter*** preko ***Netlink*** biblioteka (i njihovih funkcija) omogućava spojnice prema Linux kernelu, dopuštajući određenim kernel modulima da registriraju funkcije s mrežnim stôgom unutar kernela. Te funkcije, koje se obično primjenjuju na promet u obliku pravila filtriranja i modifikacije (ali i drugih specifičnih radnji), pozivaju se za svaki mrežni paket koji prolazi odgovarajuću spojnicu to jest takozvanu *kuku* (engl. *hook*), koja se nalazi unutar mrežnog stôga Linuxa.
To u praksi znači i da se primjerice mrežno sučelje ***Bridge*** može spojiti (preko svog kernel modula [`bridge`]), s *Netfilter* sustavom, to jest preko njega indirektno povezati s mrežnim stôgom Linuxa s jedne strane, a s druge strane s mrežnom karticom računala. Tako će sav promet koji će prolaziti kroz ***Bridge*** mrežno sučelje ili druga logička/virtualna sučelja spojena na njega, prvo prolaziti kroz *Netfilter* komponentu.


Vezano za ***Bridge*** mrežno sučelje, pogledajte poglavlje:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**
Moguće je filtriranje (oblikovanje) mrežnog prometa i na razini tzv. ***queuing*** mehanizama, za više detalja pogledajte poglavlje:
**25.1.4.2. Oblikovanje mrežnog prometa i naredba tc.**

Naposlijetku, ovisno o tome hoće li *Netfilter* propustiti ili odbaciti pojedine pakete u ovoj komunikaciji, oni će se dalje slati prema mrežnom stôgu Linuxa i/ili prema aplikacijama ili dalje na mrežu.
Pogledajte ovakvu, pojednostavljenu shemu, na slici 247.

**Slika 247. Pogled na mrežni podsustav Linuxa s naglaskom na NetLink i Netfilter.**




Bez obzira koji softver za vatrozid u Linuxu koristili (***iptables***, ***ipset***, ***nftables***, ***firewalld***, **UFW**, …) on se oslanja na ***Netfilter*** sustav. Svi vatrozidi rade tako da centralni kernel modul vatrozida; primjerice za ***iptables*** je to `ip_tables`, za ***nftables*** je to `nf_tables` i slično, s korisničkim naredbama, primjerice: `iptables`, `ipset`, `nft` i drugima, komuniciraju preko *Netlink* biblioteka. Nadalje, centralni kernel modul vatrozida (pr. `nf_tables`) oslanja se na druge kernel module od kojih je svaki zadužen za određenu dodatnu zadaću (pr. `nf_nat` je zadužen za **NAT**). S druge strane ovi kernel moduli u interakciji su sa spojnicama (kukama) *Netfilter* sustava.

Ove spojnice prema kernelu koriste **Netfilter** programsko sučelje. Naime, svaki paket koji uđe u mrežni sustav (dolazni ili odlazni) aktivirat će ove spojnice (kuke) kako prolazi kroz mrežni stôg, dopuštajući programima koji se registriraju pomoću tih kuka, interakciju s mrežnim prometom na ključnim točkama. Kernel moduli povezani s centralnim kernel modulom vatrozida registriraju se na ovim spojnicama kako bi osigurali da je promet u skladu s uvjetima postavljenim prema pravilima vatrozida.

Pri tome postoji pet *Netfilter* kuka s kojima se program može registrirati. Kako paketi napreduju kroz mrežni stôg, oni će aktivirati kernel module koji su se registrirali s ovim kukama. Priključnice koje će paket aktivirati ovise o tome je li paket dolazni ili odlazni, o odredištu paketa i je li paket ispušten ili odbijen u prethodnoj točki. Podsjetimo se mrežnog modela Linuxa.
Pogledajte detaljniji dio mrežnog sustava s naglaskom na ***Netlink*** i ***Netfilter*** podsustave (slika 248. dolje):

**Slika 248. Linux: pogled na mrežni podsustav s naglaskom na Netfilter i Netlink te kernel module.**

Aplikacije (programi/naredbe)
thunderbird
ip
route
nft
ipset
firefox
ifconfig
iptables
...
Sistemski pozivi (socket i ostalo)
NetLink pozivi (NetLink Socketi i ostalo)
Netlink sloj
libnl-genl
Za pristup Netlink socketima.
libnl-route
Pristup podsustavu za usmjeravanje (Routing).
libnl
(Netlink suite) rukovanje s NetLink socketima (utičnicama), upravljanje mrežnim sučeljima i vezama te njihovim IP parametrima, praćenje veza, ...
libnl-xfrm
Sučelje za RAW netlink socket.
libnl-nf
Za kreiranje i spajanje na Netfilter podsustav.
libnl-idiag
Za dijagnostiku.
Mrežni promet
kernel API
Interni kernel API
Korisnički prostor (User space)
Kontrola i postavke
iptable_filter
Rad s tablicama za filtriranje
ip_tables
Jezgra iptables
Kernel moduli (pr. za vatrozid)
nfnetlink
Za vezu s Netlink
iptable_nat
NAT funkcionalnost
nf_tables
Jezgra nftables
iptable_mangle
Rad s Mangle tablicom
nf_conntrack
Praćenje konekcija
nf_nat
NAT funkcionalnost
Mrežni stôg (Network stack)
TCP
UDP
IP
RFS, RPS, GRO, GSO, scheduler, buffers. . .
Netfilter podsustav
Packet filtering
Filtriranje paketa na osnovu pravila filtriranja
NAT (i PAT)
Translacija IP adresa (pr. promjene izvorišne i odredišne IP, ...)
Packet mangling
Promjene unutar paketa (pr. označavanje, ToS, TTL, ...)
TC (traffic control)
Upravljački program (driver)
Kernel prostor (Kernel space)
Mrežna kartica (NIC/LAN)



***Sumirajmo: Netfilter*** je softverski okvir (Engl. *framework*) koji nam pruža Linux kernel, koji omogućava provođenje različitih operacija s mrežnim paketima to jest s filtriranjem mrežnih paketa.
***Netfilter*** nam preko ***Netlinka*** daje skup programskih poveznica/kuka unutar Linux kernela, koji preko specijaliziranih kernel modula koriste funkcije koje se potom primjenjuju na mrežni promet (pakete) u obliku pravila filtriranja i preinaka.

## NetLink i sistemski pozivi detaljnije

Vratimo se na trenutak na sliku s prethodne stranice (248). Odnosno vratimo se na aplikacije i *Netlink* i *sistemske pozive*. Naime, što se tiče komunikacije između aplikacija i mreže, ona je na Linux sustavu moguća na dva načina:

- Upotrebom navedenih ***Netlink socketa*** (odnosno *Netlink* poziva). Važno je znati da se *Netlink* koristi za upravljanje komponentama mrežnog stoga ili mrežnih sučelja, ali ne i za prijenos stvarnih korisničkih podataka odnosno mrežnih paketa. To znači da pomoću *Netlink socketa* možete konfigurirati kako mrežni stog obrađuje pakete (npr. postavljanje politika rutiranja, vatrozida i sl.), ali ne možete direktno slati ili primati mrežne pakete pomoću njega.
- Upotrebom ***Sistemskih poziva*** koji se koriste za slanje/primanje paketa na mrežu, ali i druge stvari.



Vezano za ***sistemske pozive*** pogledajte poglavlje:
**9.2.2. Signali i sistemski pozivi.**

*Netlink socketi* i *sistemski pozivi* su dva različita mehanizma za komunikaciju između kernela i korisničkih programa u Linux sustavu. Neke od ključnih razlika među njima su:

- Njihova namjena:
  - ***Netlink socketi*** – specifično su dizajnirani za komunikaciju između kernela i korisničkog prostora odnosno aplikacija, za područje mreže i mrežnih protokola, ali ne i za sâmo slanje ili primanje mrežnih paketa.
  - ***Sistemski pozivi*** su opći mehanizam za komunikaciju između kernela i korisničkog prostora (aplikacija) koji se može koristiti za različite svrhe, uključujući i mrežnu komunikaciju: slanje/primanje paketa na mrežu.
- Vrste komunikacije:
  - ***Netlink socketi*** – omogućuju strukturiranu i specifičnu komunikaciju za određene mrežne podsustave, poput upravljanja vezama, kontrolu i upravljanje mrežnim sučeljima i IP parametrima, rutiranje, rad s tablicama filtriranja vatrozida, NAT, praćenje konekcija ili upravljanja mrežnim uređajima.
  - ***Sistemski pozivi*** – opći mehanizam za izvršavanje operacija unutar kernela, a rezultati (obrade/rada) se često prenose korisničkom prostoru odnosno aplikacijama, poput slanje/primanja podataka na mrežu.
- Podržani protokoli:
  - ***Netlink socketi*** - koriste točno određeni protokol, poput `NETLINK_ROUTE`, `NETLINK_NETFILTER` itd., kako bi identificirali vrstu poruka i komunikacije koja se obavlja. Neki od njih su i:
    - `socket(↓)` – za rad sa *NetLink soketima*; moguće je koristiti cijeli niz mogućnosti, poput:
      - `AF_NETLINK` – kreira *Netlink soket* koji ne komunicira direktno s mrežnim paketima, već s mrežnim stôgom na razini konfiguracijskih i kontrolnih operacija (npr. dodavanje ruta, upravljanje mrežnim sučeljima, konfiguriranje vatrozida i sl.)
    - `nft_add_rule()` – za kreiranje pravila vatrozida prema **Netfilter** komponenti.
    - `nft_add_nat_rule()` – za dodavanje **NAT** pravila prema **Netfilter** komponenti, ….
  - ***Sistemski pozivi*** – mogu uključivati različite sistemske pozive, ovisno o željenoj operaciji. Primjerice, za mrežnu komunikaciju su to jednostavniji pozivi, poput:
    - `socket(AF_INET,SOCK_STREAM` ili `SOCK_DGRAM` ili `SOCK_RAW)`: `SOCK_STREAM` se koristi za TCP pakete, `SOCK_DGRAM` se koristi za UDP pakete, dok se `SOCK_RAW` može koristi za IP pakete direktno (pr. za prilagođene TCP/UDP pakete), ali i primjerice ICMP.
    - `bind(),listen(),sendmsg(),recvmsg(),setsockopt(),getsockopt(), …`
- Njihova primjena:
  - ***Netlink socketi*** – često se koriste u mrežnom programiranju za konkretna upravljanja i događaje, poput konfiguracije mrežnih sučelja i parametara rada sučelja, praćenja ili filtriranja mrežnog prometa i slično.
  - ***Sistemski pozivi*** – koriste se za širok raspon operacija poput slanja ili primanja paketa na mrežu, ali i operacije koje nisu uvijek specifične za mrežu, kao što su upravljanje datotekama, procesima, resursima sustava i sve drugo.

Sada se u razmatranju možemo vratiti na početak priče o ***Netlink socketima***.



Vezano za ***Netfilter*** komponentu, na nju se oslanjaju i ***iptables*** te ***firewalld*** kao i njihove pripadajuće naredbe, ali i naredbe poput `ip` i mnogih drugih. I novija implementacije vatrozida ***nftables*** se također oslanja na ***Netfilter*** sustav.

## Libnl i NetLink detaljnije

***Libnl*** je skup biblioteka za rad s ***netlink*** protokolom i nekim od protokola više razine implementiranih povrh njega. Cilj *NetLink* protokola je pružiti API-je (programsko sučelje) na različitim razinama apstrakcije. Kako smo spomenuli, a vidljivo je na slici **248**, *libnl* se sastoji od nekoliko osnovna biblioteka, a primarna `libnl` pruža temeljni skup funkcija za rad s *NetLink* mrežnim utičnicama (*NetLink socketima*), konstruiranje poruka i slanje ili primanje tih poruka prema kernelu (ne i za slanje/primanje podataka na mrežu), predmemoriranje podataka i slično.

Dodatna programska sučelja više razine nalaze se unutar nekoliko pojedinačnih *netlink* protokola, a dostupna su u zasebnim bibliotekama; na primjer: `libnl-route`, `libnnl-genl`, … . Ovaj sustav biblioteka osmišljen je kako bi se osiguralo da su sve komponente izborne, tj. iako osnovna biblioteka pruža sustav predmemoriranja koji omogućuje jednostavno upravljanje objektima bilo koje vrste, nijedna aplikacija nije potrebna za korištenje ovog sustava predmemoriranja, ako za to nema potrebe.

Upoznajmo se s ***netlinkom*** to jest ***Libnl*** bibliotekama i njihovom vezom s ostatkom Linuxa; pa ih pogledajmo redom:

- Jezgra ***libnl*** sustava je biblioteka imena `libnl`. Ona je zadužena za rukovanje s *NetLink* utičnicama (*socketima*), upravljanje vezama, slanje i primanje podataka (ne preko mreže), konstrukciju i parsiranje poruka, sustav predmemoriranja objekata i sl.
- Sljedeća biblioteka zadužena je za rad s *NetLink* mrežnim utičnicama (*socketima*) odnosno daje nam ***API*** (programsko sučelje) za pristup na njih, a zove se: `libnl-gen`.
- Slijedi `libnl-diag` koja je zadužena za razne dijagnostičke potrebe to jest daje nam ***API*** do tih komponenti.
- Komponenta `libnl-nf` zadužena je za pristup na ***Netfilter*** sustava za filtriranje mrežnih paketa to jest daje nam ***API*** za pristup do tog sustava. Dakle pomoću ove komponente odrađuju se sve funkcije vatrozida, pomoću dodatnih kernel modula, primjerice:
    - Za ***Iptables*** vatrozid se prvo koristi centralni kernel modul: `ip_tables`, koji potom koristi i:
        - Kernel modul imena: `iptable_filter` za filtriranje mrežnih paketa.
        - Kernel modul imena: `iptable_nat` za ***NAT*** funkcionalnost (translaciju IP dresa).
        - Kernel modul imena: `iptable_mangle` je za *mangle* tablice koje se koriste za promjene paketa unutar određenog lanca. Primjerice ako se to primjenjuje unutar ***PREROUTING*** lanca, tada je on za izmjenu dolaznih paketa prije usmjeravanja, a ako se primjenjuje unutar ***OUTPUT*** lanca, tada se izmjene rade za lokalno generirane pakete prije usmjeravanja i tako dalje.
        - Kernel modul imena: `iptable_filter` za filtriranje mrežnih paketa.
        - Kernel modul imena: `nf_conntrack` za praćenje konekcija, …
    - Za ***nftables*** vatrozid se prvo koristi centralni kernel modul: `nfnetlink` za pristup na ***Netlink*** sustav.
        - Potom se koristi kernel modul: `nf_tables`, za osnovnu funkcionalnost, koji potom koristi i:
            - Kernel modul imena: `nf_nat` za ***NAT*** odnosno translaciju IP dresa.
            - Kernel modul imena: `nf_conntrack` za praćenje konekcija, …

Svi ***netlink*** kernel moduli za funkcionalnosti vatrozida su vidljivi unutar vršnog direktorija (mape):

```
ls -al  /lib/modules/`uname -r`/kernel/net/ipv4/netfilter/
```

- I zadnja komponenta zove se `libnl-route`, a ona ima malo širi spektar funkcija. Ona nam daje ***API*** prema ***Netlink*** sustavu za usmjeravanje (*routing*), pravila usmjeravanja i rute, adrese, kontrolu prometa, ali i drugim komponentama. Neke od njih su:
    - Mrežna sučelja niže razine (nabrojana od više do najniže razine):
        - ***Bridge*** sučelje (mrežni most/premosnik), za koje je odgovoran kernel modul: `bridge`.


Vezano za **Bridge** sučelje pogledajte poglavlje:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**

- ***MACVLAN*** sučelje, za koje je odgovoran kernel modul: `macvlan`.


Vezano za **MACVLAN** sučelje pogledajte poglavlje:
**26.8.2. Druga mrežna sučelja (*MACVLAN,IPVLAN,VXLAN,MACVTAP/IPVTAP*).**

- ***VLAN*** (prema 802.1Q standardu) sučelje, za koje je odgovoran kernel modul: `8021q`.


Vezano za **VLAN** sučelje pogledajte poglavlje:
**20.6.2. VLANovi odnosno virtualne lokalne mreže.**

- **VxLAN** sučelja, za koje je odgovoran kernel modul: `vxlan`.


Vezano za **VxLAN** sučelje pogledajte poglavlje:
**26.8.2. Druga mrežna sučelja (*MACVLAN,IPVLAN,VXLAN,MACVTAP/IPVTAP*).**

- ***VETH*** sučelje, za koje je odgovoran kernel modul: `veth`.


Vezano za **VETH** sučelje pogledajte poglavlje:
**20.6.4. VETH - posebno mrežno sučelje.**

- ***Bond*** sučelje (team/agregacija [pr. ***LACP***]), za koje je odgovoran kernel modul: `bond`.


Vezano za **Bond** sučelje pogledajte poglavlje:
**20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).**

**Izvori informacija: (1051),(1311),(1312),(1313),(1342), `man 7 netlink`,`man 2 syscalls`, `man 7 rtnetlink`,`man iptables`, RFC3549 .**

### 26.7.3.1. Nova funkcionalnost: *ipset*

Iako standardno s ***RedHat/CentOS*** 6.x dolazi *iptables* servis i naredba, moguće je kao nadogradnju koristiti i noviju naredbu `ipset` koja doduše koristi isti ***Netfilter*** *softverski okvir* koji u konačnici odrađuje filtriranje paketa. Međutim *ipset* uvodi mogućnost kreiranja pravila vatrozida koja se podudaraju s cijelim skupovima ili „*setovima*" IP adresa odjednom.
Promatrajmo *ipset* kao nadogradnju osnovne *iptables* funkcionalnosti. Naime za razliku od uobičajenih *iptables* lanaca, koji se linearno pohranjuju i kod svake provjere prolaze sekvencijalno (redom),
IP skupovi se pohranjuju u optimiziranoj indeksiranoj strukturi, što prolazak i pretraživanje čini vrlo učinkovitim, čak i kada se radi o vrlo velikim skupovima. U slučaju upotrebe većih *iptables* lanaca, u odnosu na *ipset*, u korist *ipseta* se postižu značajnija ubrzanja[(723)]. Važno je razumjeti da se *ipset* poput *iptablesa* sastoji i od alata u korisničkom prostoru kernela i od kernel modula, pa vam trebaju oba kako bi on pravilno funkcionirao. Osnovni kernel modul potreban za *ipset* je kernel modul imena `ip_set` koji pak ovisi o kernel modulu `nfnetlink`.
Prvo instalirajmo potrebni softverski paket za *ipset* funkcionalnost, ali i sistemski servis `ipset-service`.

```
yum -y install ipset ipset-service
```

Sada privremeno pokrenimo ovaj servis (*RedHat/CentOS* 6.x ili 7.x+). Za trajno pokretanje servisa pogledajte poglavlje: **7.3.**

```
service ipset start
```

Ako je potrebno, odnosno, ako nije učitan, učitajmo i kernel modul `ip_set`

```
modprobe ip_set
```

Sada pogledajmo jesu li se učitali svi kernel moduli kako treba:

```
lsmod | grep ip_set

ip_set                 49152  0
nfnetlink              16384  2 nf_tables,ip_set
```

Dakle oba kernel modula: `ip_set` i `nfnetlink` su tu, stoga smo spremni za rad.



Na *RedHat/CentOS* 7.x i novijim, `ipset` već dolazi instaliran pa nije potrebno ništa dodatno instalirati.

Navest ćemo sâmo nekoliko opcija i parametara naredbe `ipset`:

- `create` ili `-N` ***IMESKUPA METODA:VRSTA*** – za kreiranje novog seta odnosno skupa. Pri tome ***metoda*** i ***vrsta*** ovise o unosima koje dodajemo u skup, pri tome:
    - ***METODE*** mogu biti: `bitmap`, `hash` i `list`
    - ***VRSTE*** mogu biti: `ip`,`net`,`mac`,`port` i `iface`
  - `add` ili `-A` ***IMESKUPA UNOS*** – za dodavanje unosa: IP adresa, portova i slično, u postojeći skup.
  - `del` ***IMESKUPA UNOS*** – za brisanje unosa unutar kreiranog skupa.
  - `test` ***IMESKUPA UNOS*** – za testiranje pripada li određeni unos traženom skupu.
- `destroy` ***IMESKUPA*** – za brisanje cijelog skupa sa svim unosima (ili svih skupova, ako nije naveden skup).
- `list` ***IMESKUPA*** – za ispis svih unosa koje smo dodali u traženi skup (prošireno sa stvarnim opsegom skupa).
- `save` ***IMESKUPA*** – za snimanje skupa (ispis je primarno na *stdout*). Ako želimo ispis u datoteku; možemo koristiti dodatno: `-file` *DATOTEKA*
- `restore` ***IMESKUPA*** – za povratak snimljenog skupa; primjerice iz datoteke: `-file` *DATOTEKA.*
- `flush` ***IMESKUPA*** – za brisanje svih unosa iz željenog skupa ili ako nije naveden skup, iz svih skupova.
- `rename` ***STAROimeSKUPA NOVOimeSKUPA*** – za preimenovanje imena postojećeg skupa.

Pogledajmo i nekoliko primjera upotrebe naredbe `ipset` kao nadogradnje na naredbu odnosno *iptables* funkcionalnost.
Kreirajmo skup imena `serveri1` koji će sadržavati opsege IP adresa: `192.168.10.0/24` i `192.168.20.0/24`:

```
ipset create severi1 hash
ipset add serveri1 192.168.10.0/24
ipset add serveri1 192.168.20.0/24
```

Sada kreirajmo *iptables* poveznicu odnosno pravilo na ovaj skup u kojem zabranjujemo pristup s navedenih mreža:

```
iptables -A INPUT -m set -match-set serveri1 src -j DROP
```

Kreirajmo novi skup imena `serveri2` u kojem za cijelu mrežu: `192.168.1.0/24` dozvoljavamo portove od `80` do `84` za IP adresu `192.168.2.1` i to sâmo za `UDP` port `53` (DNS protokol).

```
ipset create serveri2 hash:ip,port
ipset add serveri2 192.168.1.0/24,80-84
ipset add serveri2 192.168.2.1,udp:53
```

I na kraju dodajmo *iptables* pravilo za dozvolu pristupa s IP adresa i portova navedenih u našem skupu imena `serveri2`:

```
iptables -A INPUT -m set --match-set serveri2 src -j ACCEPT
```

Sada snimimo konfiguraciju, koja će biti snimljena u direktorij `/etc/sysconfig/ipset.d/`, i to po jedna datoteka za svaki pojedini skup.

```
service ipset save
```

**Izvori informacija: (723),(724),(725), `man ipset`.**

### 26.7.3.2. Nftables

U novije vrijeme došlo je do promjena načina na koji se filtriraju mrežni paketi. Naime od linux kernela 3.13 više se ne koristi *Netfilter*/***iptables*** *softverski okvir* za filtriranje paketa. Novi *softverski okvir* za obradu i filtriranje mrežnih paketa se zove ***Nftables***, a on mijenja određene dijelove *netfiltera* sustava uz zadržavanje i ponovno korištenje velikog dijela njegovih funkcionalnosti. Osnovna prednost *nftables* u odnosu na klasični *netfilter* je znatno manje dupliciranje programskog kôda i funkcija, a sâmim time i veća propusnost kao i pojednostavljenje Linux kernel ***ABI*** sučelja (*application binary interface*).

Sve to uz dodatno učinkovitije izvršavanje i pohranu pravila filtriranja kao i inkrementalne promjene pravila filtriranja. *Nftables* također nudi poboljšani ***API*** u korisničkom prostoru kernela (*user space*), koji omogućuje zamjenu jednog ili više pravila vatrozida unutar jedne transakcije prema *Netlink* sučelju. To znatno ubrzava promjene konfiguracije vatrozida, pogotovo za sustave s velikim brojem pravila filtriranja i/ili s kompleksnijim pravilima filtriranja.
Zbog svega navedenog se više ne koristi niti naredba `iptables` već imamo novu naredbu `nft` s kojom koristimo nove mogućnosti i novi *nftables softverski okvir*. Osim što više nemamo naredbu `iptables` nemamo niti stare naredbe: `ip6tables`, `arptables` i `ebtables`, koje su sada sve objedinjene u novoj naredbi `nft` i pripadajućem novom *softverskom okviru*.


Servisi: ***iptables***, ***iptables-nft*** i ***ipset*** su od inačice 9.x. ***RedHat*** Linuxa zastarjeli i zamijenjeni su s ***nftables***.

Naime ***nftables*** kernel mehanizam u Linux kernel dodaje jednostavnu virtualnu mašinu koja je sposobna izvršiti takozvani bajt-kôd za pregled svakog mrežnog paketa i donošenje odluka o tome kako se taj paket treba prihvatiti i obraditi. Operacije koje provodi ovaj virtualni stroj su namjerno razvijene tako da sadrže samo osnovne (temeljne) funkcionalnosti. Dakle on (virtualni stroj za *nftables*) je u stanju izvući podatke iz sâmog mrežnog paketa, pogledati povezane metapodatke poput primjerice ulaznog mrežnog sučelja, ali i upravljati podacima za praćenje veze koja se odnosi na taj paket. On također može koristiti aritmetičke i druge operatore za donošenje odluka na temelju tih podataka. Virtualni stroj je također sposoban manipulirati skupovima podataka (obično se to odnosi na IP adrese) dopuštajući da se višestruke operacije usporedbe, zamijene jednim traženjem skupa podataka odnosno jednim jedinim pravilom. Dodatno unutar jednog pravila (*firewall rule*) moguće je definirati više akcija (pr. ***log***, ***drop*** i sl.). Osim navedenog, *nftables* pravila su lakša za čitanje i slijede sintaksu sličnu kao kod naredbe `tcpdump` ili sličnih naredbi. Dakle nova sintaksa je čitljivija i jednostavnija, ali i znatno naprednija s mogućnostima.
Gore opisani način rada je u potpunoj suprotnosti s klasičnim „*iptables firewalling*" programskim kôdom koji ima izgrađene mehanizme koji su svjesni svakog protokola tako duboko u logici rada, da se programski kôd morao replicirati četiri puta:

- Za IPv4 i za IPv6 protokol.
- Za ARP protokol.
- I za *Ethernet* mostove/premosnike odnosno *bridge* uređaje ili *bridge* funkcionalnost.

Sve to zato što je sve bilo implementirano previše specifično za svaki od navedenih protokola kako bi se moglo koristiti na generički način. Iz navedenog je jasno kako se upotrebom novog modela sve pojednostavilo i ubrzalo te se omogućio lakši razvoj i održavanje programskog kôda.

**Mi ćemo koristiti kernel v.4.20 i `nft` v.0.8 jer neke opcije u starijim inačicama kernela ili `nft` programa nisu dostupne.**

## Kako se konfigurira i koristi *nftables?*

Prije nego uopće krenemo, moramo provjeriti imamo li novi Linux kernel modul potreban za ***nftables***:

```
modinfo nf_tables
```

Ako imamo dostupan ovaj kernel modul, dobiti ćemo i neke informacije o njemu. Sada možemo učitati ovaj kernel modul:

```
modprobe nf_tables
```

Kernel modul `nf_tables` ovisi i kernel modulu `nfnetlink` koji će se u našem primjeru odmah i učitati.
Napomena: Kernel modul zadužen za staru funkcionalnost (*iptables*) je: `ip_tables` i on nam više nije potreban.
Dakle preporuka je koristiti ili stari (*iptables*) ili novi sustav (*nftables*). U primjerima dalje, koristit ćemo novi sustav (***nftables***) stoga ćemo isključiti stari. Međutim prvo instalirajmo program `nft` i njemu potrebne biblioteke:

```
yum -y install nftables
```

I sada ga postavimo da se pokreće tijekom svakog pokretanja sustava kao servis (***RedHat/CentOS* 7.x+**). Zatim ga pokrenimo odmah potom. Kod ***RedHat*** 9.x. ovo je već standardno postavljeno.

```
systemctl enable nftables
systemctl start nftables
```

I još zaustavimo i isključimo stare *iptables* i *firewalld* (ako i on postoji) servise jer ih više ne želimo koristiti:

```
systemctl stop iptables
systemctl stop firewalld
systemctl disable iptables
systemctl disable firewalld
```

I konačno obrišimo sva trenutna *iptables* pravila (i za IPv4 i za IPv6) jer *iptables* više nećemo koristiti:

```
ip6tables -F
iptables -F
```

## Tablice (*Tables*)

Prvo kreirajmo novu tablicu. Važno je znati kako tablice sadrže lance (Engl. *chains*), a u njima se definira vrsta protokola (***family***) koji može biti:

- `ip` - za IPv4 protokol (lance).
- `ip6` - za IPv6 protokol (lance).
- `inet` - za miješane IPv4 ili IPv6 (lance).
- `arp` - za ARP lance.
- `bridge` - za *bridge* (mrežni most) lance.
- `netdev` - za sav promet kada dolazi na mrežni sučelje.

Konačno kreirajmo novu tablicu imena `tablica-1` za IPv4 protokol, upotrebom naredbe `nft` na sljedeći način:

```
nft add table ip tablica-1
```



Tablice možemo: dodavati (`add`), brisati (`delete`), ispisati (`list`) ili isprazniti (`flush`).

Izlistati našu tablicu sa svim lancima i pravilima možemo sa sljedećom naredbom:

```
nft list table tablica-1
```

## Lanci (*Chains*)

Sada trebamo kreirati lanac pravila (*chain*). Lanci sadrže pravila vatrozida. Lanci se definiraju prema sljedećoj sintaksi:

```
nft add chain familiy ime-tablice ime-lanca type hook prio policy
```

Lanci po vrsti (***type***) mogu biti:

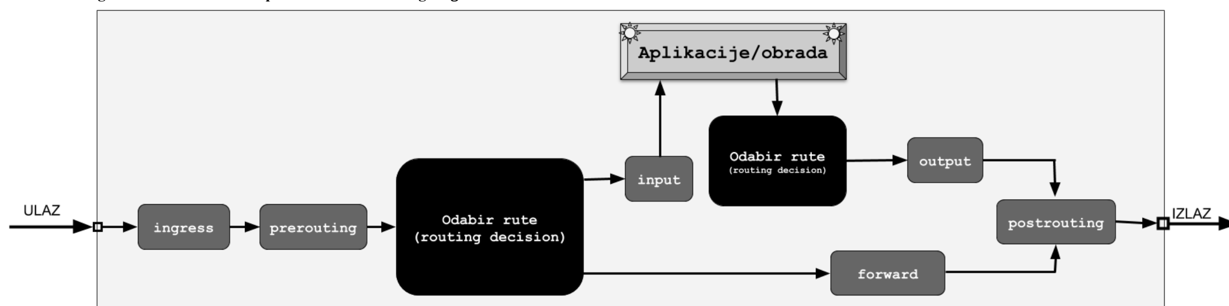
- `filter` - za filtriranje prometa.
- `rute` - za preusmjeravanje paketa (*rerouting*), koji pri tome modificiraju IP zaglavlje paketa: moguće ih je primijeniti za IPv4 i IPv6.
- `nat` - za NAT (Engl. *Network Address Translation*); pri tome samo prvi paket tôka podataka određene konekcije ulazi u ovaj lanac pa nije pogodno za filtriranje. Umjesto toga za filtriranje je potrebno koristiti filtere.

Poveznica/*kuka* (***hook***) je poveznica na točku na kojoj će se obrada paketa vezati na mrežni stôg linux kernela, pa može biti:

- `ingress` - povezuje se na ulaz prije *preruting* točke; za *netdev* (mrežne uređaje/sučelja).
- `prerouting` - povezuje se na ulaz prije nego se počne primjenjivati *ruting*. Primjenjuje se za IPv4 i IPv6.
- `input` - povezuje se s točkom u kojoj se analiziraju svi paketi kojima je odredište naš sustav. Primjenjuje se za ARP, IPv4 i IPv6.
- `output` - povezuje se s točkom u kojoj se analiziraju svi paketi koji se šalju s našeg sustava. Primjenjuje se za ARP, IPv4 i IPv6.
- `forward` - povezuje se s točkom u kojoj se analiziraju svi paketi kojima nije niti odredište niti izvor s našeg sustava (dakle oni koji se prosljeđuju). Primjenjuje se za IPv4 i IPv6.
- `postrouting` - povezuje se s točkom u kojoj se analiziraju svi paketi koji napuštaju naš sustav i to tek nakon odabira rute (nakon *routing odluke*). Primjenjuje se za IPv4 i IPv6.

Pogledajmo i kako paketi prolaze kroz mrežni stôg linuxa (na slici 249).

**Slika 249. Pogled na obradu mrežnih paketa unutar mrežnog stôga Linuxa.**




Prioriteti (***prio***) se odnose na prioritet paketa, a postoji ih cijeli nîz, počevši od -400 do +300 s time da najniži broj (-XY) ima najveći prioritet. Međutim, neki od brojeva prioriteta su već definirani. Dodatno je važno znati kako se lanac s nižim prioritetom izvršava prvi, a onda onaj s višim prioritetom, a pri tome se može dogoditi da lanac s višim prioritetom koji se kasnije izvršio, pregazi onaj prije njega (ako nismo pazili što koji od njih radi).

I na kraju (***policy***) može biti:

- drop - za odbacivanje paketa.
- accept - za prihvaćanje paketa.
- ...

Dodat ćemo novi lanac imena lanac-tcp u našu tablicu tablica-1, a on će biti za **filtriranje** prometa

```
nft add chain ip tablica-1 lanac-tcp {type filter hook input priority 0\;}
```



Lance možemo: dodavati (add), brisati (delete), ispisati (list) ili isprazniti (flush).



**Iptables** standardno dolazi s jednim lancem za svaku poveznicu (*hook*), s lancima nazvanim isto kao i poveznice. Međutim ***nftables*** nema standardno postavljeni (*default*) lanac.
Dodatno ***nftables*** lanci mogu ili ne moraju biti nazvani po poveznicama (*hook*) sve ovisno o vašim željama ili potrebama.

## Pravila odnosno *rules*

Tek sada dolazimo do **pravila** (***rules***) koja dodajemo u željeni lanac. Ali prvo se upoznajmo s time kako dodajemo pravila (*rules*).
Pravila se dodaju prema principu, uz primjer za dodavanje (add).
Pri tome se nadodavanje i zamjena rade malo drugačije, prema principu:

```
nft add rule familiy ime-tablice ime-lanca matches/meta statement
```



Pravila (*rules)* možemo: dodavati (add), brisati (delete), nadodati (insert) ili zamijeniti (replace).

Objasnit ćemo sâmo osnovne (vršne kategorije):

- ***matches*** - označava vršnu kategoriju protokola, od kojih svaki protokol sadrži cijeli nîz opcija.
  Vršna kategorija može biti (navesti ćemo samo neke):
  - ip – za IPv4 protokol; primjerice: protocol, saddr,daddr, …
  - ip6 – za IPv6 protokol; primjerice: protocol, saddr,daddr, …
  - tcp – za TCP protokol; primjerice: dport, sport,sequence, …
  - udp – za UDP protokol; primjerice: dport, sport, …
  - udplite – za UDP lite protokol; primjerice: dport, sport, …
  - sctp – za SCTP protokol; primjerice: dport, sport, …
  - dccp – za DCCP protokol; primjerice: dport, sport, …
  - ah – za AH protokol; primjerice: hdrlength, spi, …
  - esp – za ESP protokol; primjerice: sequence, spi, …
  - icmp – za ICMP protokol; primjerice: type, id, …
  - vlan – za VLAN mreže (802.1Q protokol); primjerice: cfi, id, …
  - arp – za ARP protokol; primjerice: plen, ptype, …

- ***meta*** – označava vršnu kategoriju koja označava meta podatke prema kojima se mogu raditi određene akcije.
  Vršna kategorija može biti (navesti ćemo samo neke):
  - iifname – prema vrsti ulaznog mrežnog sučelja; primjerice: meta iifname „eth0‟
  - oifname – prema vrsti izlaznog mrežnog sučelja; primjerice: meta oifname „eth0‟

- ***statement*** - označava radnju koja će se odraditi za određeni paket koji ulazi u ovo pravilo:
  - accept – paket se prihvaća.
  - drop – paket se ne prihvaća te se dalje provjere paketa ne rade.
  - queue – paket se ubacuje u nîz (*queue*) i više ništa se po njemu ne radi jer se tada prepušta kontroli na aplikaciji na korisničkoj razini (*user space*).
  - continue – paket se prihvaća i prosljeđuje na slijedeće pravilo.
  - return – izlazi se s izvršavanja trenutnog lanca (*chain*) i nastavlja na izvršavanje sljedećeg pravila (*rule*)
  - jump – nastavlja se od prvog pravila (*rule*) točno definiranog lanca (*chain*).
  - goto – slično kao prethodno ali nakon prelaska na novi definirani lanac (*chain*) nastavlja se od zadnjeg definiranog umjesto onoga definiranog u ***goto*** izjavi.

- Moguće je koristiti i pravila za:
  - *NAT* – (translaciju adresa) i to:
    - `dnat` - odredišni (*destination*) ***NAT*** (pr. `dnat 192.168.1.1`)
    - `snat` - izvorišni to jest takozvani *source **NAT*** (pr. `snat 192.168.1.1`)
    - `masquerade` - *masquerade **NAT*** odnosno *klasični NAT* (pr. `masquerade`)
  - `log` – logiranje (spremanje/čuvanje) poruka.
  - `reject` - odbacivanje paketa.
  - `counter` - pravila za brojanje odnosno za takozvane brojače.
  - `limit rate` - pravila za ograničavanje broja paketa, i to (navesti ćemo ih samo par):
    - `/minute/hour/day/week` - po minuti/satu/danu/tjednu.
    - `over` - preko određenog broja.
    - `bytes/kbytes/mbytes` - prema prometu (bajta/kilobajta/megabajta, ..).

I sada konačno dodajmo jedno pravilo (*rule*) u tablicu `tablica-1` i to u njen postojeći lanac `lanac-tcp` u kojem će se odbaciti svi paketi kojima je odredišni port broj 80, koji se odnosi na *http* protokol:

```
nft add rule tablica-1 lanac-tcp tcp dport 80 drop
```

Sva upisana pravila (*rules*) možemo vidjeti s naredbom:

```
nft list ruleset
```

I dobit ćemo nešto poput:

```
table ip tablica-1 {
        chain lanac-tcp {
                type filter hook input priority 0; policy accept;
                tcp dport http drop
        }
}
```

Vidimo kako je standardno pravilo da se sve propušta (`policy accept`), a poslije mi dodajemo pravilo s kojim odbacujemo pakete kojima je odredišni port **80** (`http`). Toliko o osnovnim primjerima rada s naredbom `nft` i *nftables* vatrozidom.

## NAT (*Network Address Translation*) + PAT ili SNAT (*Source NAT*) pravila

Zamislimo slučaj kada je naš Linux i centralni usmjerivač na kojemu želimo raditi NAT+PAT, pri čemu mu je `eth0` WAN sučelje prema internetu. Stoga napravimo sljedeće korake:

**8.** Kreiranje tablice imena **`nat`**:

```
nft add table nat
```

**2.** Kreiranje **prerouting** i **postrouting** lanaca (podsjetite se slike 249. nekoliko stranica prije):

```
nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

**3.** Potom dodajmo pravilo unutar **postrouting** lanca s kojim nalažemo *masquerade* (*NAT+PAT*) način rada:

```
nft add rule nat postrouting oifname "eth0" masquerade
```

Nakon ove konfiguracije, naš Linux kao usmjerivač će skrivati sve unutarnje IP adrese na mrežnom sučelju `eth1` iz opsega 192.168.2.0/24 i pretvarati ih u svoju vanjsku IP adresu definiranu na **WAN** mrežnom sučelju `eth0`.

## *SNAT*

Sa sljedećom konfiguracijom ćemo napraviti SNAT pravilo koje će za sav vanjski promet koji ulazi u lokalnu mrežu iz sučelja `eth0` preko sučelja `eth1` promijeniti izvorišnu (*source*) IP adresu u adresu `eth1` sučelja: `192.168.2.1`, tako da će računalima iza ovog Linux usmjerivača izgledati kao da promet dolazi s njega to jest s IP adrese `192.168.2.1`.

Dakle napravimo korake 1,2 i 3 iz prethodnog primjera, nakon kojih slijedi novi četvrti korak. Specifičan za SNAT:

```
nft add rule nat postrouting oifname "eth0" snat to 192.168.2.1
```

## Dalji rad s NFT i novosti od RedHat 9.x

Servisi: **iptables**, **iptables-nft** i **ipset** od inačice 9.x. *RedHat* Linuxa zastarjeli su i zamijenjeni su s novim servisom **nftables**. Međutim servis ***firewalld*** je i dalje ostao aktivan te ga se preporuča i koristiti ukoliko želite koristiti jednostavnije konfiguracije vatrozida. Ipak, za iole napredniji rad, preporuča se koristiti noviji servis **nftables**.

**Konverzija iptables (i/ili ip6tables) pravila vatrozida u nft pravila**

Srećom po korisnike **iptables** pravila (konfiguracije) vatrozida, postoji metoda konverzije **iptables** u **nft** pravila, u nekoliko koraka. Prvo ćemo snimiti stara **iptables** pravila. Dajemo primjere samo za IPv4 pravila vatrozida:

```
iptables-save > /root/iptables.dump
```

Zatim ćemo konvertirati **iptables** u **nftables** pravila:

```
iptables-restore-translate -f /root/iptables.dump > /etc/nftables/migracija.nft
```

Sada imamo **nftables** pravila koja možemo privremeno učitati:

```
nft -f /etc/nftables/migracija.nft
```

**Snimanje NFT pravila i konfiguracija**

Pošto smo prethodno zaustavili sve prijašnje (**iptables** i **firewalld**) vatrozide i onemogućili njihov rad, te pokrenuli **nftables**, možemo nastaviti dalje.



Inicijalne postavke servisa **nftables** nalaze se u datoteci: `/etc/sysconfig/nftables.conf`.
**Nftables** pravila se obično pohranjuju u datoteke ekstenzije `.nft` unutar vršnog direktorija: `/etc/nftables/`.

Važno je razumjeti da se konfiguracija ***nftables*** pravila unutar `.nft` datoteke malo razlikuje od naredbi, pa će tako gore navedene naredbe (**1-4**) koje smo koristili za ***NAT*** i ***SNAT***, zapisane u *nft* datoteku izgledati ovako:

```
table ip nat {
        chain prerouting {
                type nat hook prerouting priority dstnat; policy accept;
        }

        chain postrouting {
                type nat hook postrouting priority srcnat; policy accept;
                oifname "eth0" masquerade
        }
}
```

Zapisivanje **nftables** naredbi u *nft* datoteku ne odrađuje se automatski, već to prema potrebi treba napraviti ručno.

Primjerice kreirajmo datoteku `/etc/nftables/nat.nft` s gornjim sadržajem. To možemo napraviti (nakon koraka **1-4**):

```
nft list table ip nat > /etc/nftables/nat.nft
```

Zatim je za trajnu aktivaciju potrebno navesti ovu *nft* konfiguracijsku datoteku u `/etc/sysconfig/nftables.conf` dodavanjem sljedećeg retka:

```
include "/etc/nftables/nat.nft"
```

Moguće je navoditi i pozivanje svih *nft* datoteka unutar određenog direktorija, poput primjerice:

```
include "/etc/nftables/moja-pravila/*.nft"
```

Međutim ovdje treba biti oprezan, te obratiti pažnju na to da ono pravilo koje se prvo učitava, inicijalno i obriše sva prethodna pravila to jest da napravi čišćenje svih pravila vatrozida, što je vrlo važno kod primjerice restarta ***nftables*** servisa.

Dio konfiguracije *nft* pravila to jest naredba za čišćenje svih prijašnje unesenih pravila vatrozida je sljedeća:

```
flush ruleset
```

**Nftables + eBPF**

**Nftables** može i brže. Naime moguće je koristiti i *Extended Berkeley Packet Filter* (**eBPF**) u kombinaciji s *Nftables*. ***eBPF*** radi tako da je pomoću njega moguće obradu mrežnih paketa spustiti na razinu izvršavanja samog Linux kernela, dajući vam pri tome fleksibilnost izrade mrežnih filtera uz drastično veću brzinu izvršavanja (obrade) na razini Linux kernela.

***Za primjere pogledajte izvor informacija: (1127)***.



Za više detalja o ***eBPF*** (i ***XDP***) sustavu, pogledajte sljedeće poglavlje:
**26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP* + *eBPF*).**

**Izvori informacija: (556),(557),(558),(621),(622),(623),(624),(625),(1127),(1296),(1297),(1298),(1299),** `man nft`.

### 26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP* + *eBPF*)

Negdje tijekom 1995 godine s kernelom 2.2.10 razvijen je vatrozid i alat `ipfwadm` koji je koristio pripadajuće metode filtriranja paketa naziva ***ipchains***. On se s vremenom pokazao kao manjkav te je razvijena nova metoda filtrirana paketa koja koristi `iptables` naredbu, koja je uvedena s kernelom 2.4.0. tijekom 2001 godine, a koja se često koristi i danas.
Ipak uočene su neke slabosti i ovog dizajna te je tijekom 2014. godine s kernelom 3.13. uveden novi sustav koji koristi `nftables` funkcionalnost. Važna novost ovog sustav je bilo uvođenje virtualnog stroja unutar kernela koji je zapravo implementirao pravila filtriranja (Engl. *Firewall rules*), ali je implementacija pokazala svoje slabosti jer je cijeli proces radio sporije od očekivanog, ali ipak brže od *iptables*-a. Ako pogledamo ovaj standardni model dohvaćanja i obrade mrežnih paketa, s nekoliko stranica prije (slika 249) sve izgleda logično. Međutim kada upravljački program dohvati mrežni paket (pogledajte poglavlje: **10.5.2.1.1. Kako to (NAPI) radi, još malo detaljnije**, slika 52), a nakon što je signal prekida (***IRQ***) bio okidač za to, svi koraci koji slijede, čak i ako se koriste ***DMA*** i ***NAPI*** modeli, oduzimaju prilično procesorskog vremena za obradu, a ponajviše operacija koje se tiču kreiranja i stavljanja paketa u mrežnu međumemoriju na razini operativnog sustava.



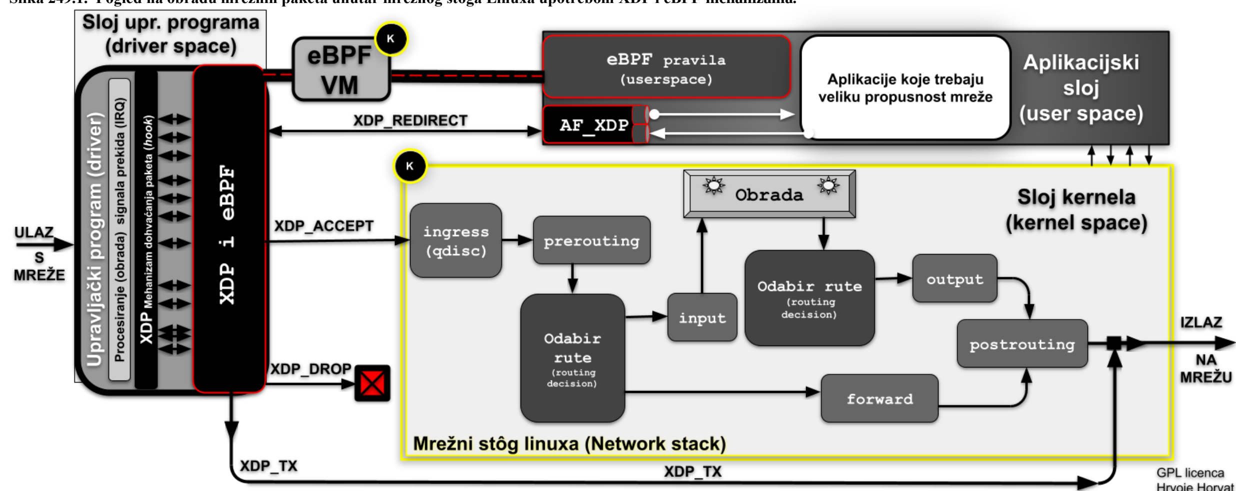
Pogledajte i poglavlje:
**25.1.1. Raspodjeljivanje mrežnih paketa (*network queuing/scheduler*).**

Stoga je u najnovije vrijeme osmišljen potpuno novi mrežni model koji se aktivno koristi od ***RedHat/CentOS*** Linuxa inačice **8.1**.x ili novijih. Ovaj model je u produkcijsku primjenu došao praktično tek s Linux kernelom 4.17. On se naziva ***XDP*** odnosno engl. *Express Data Path* tj. ekspresni put podataka. On dodatno može koristiti i novi način obrade i filtriranja paketa koji je također znatno brži od prijašnjih, a zove se ***eBPF*** (engl. *Extended Berkeley Packet Filter*) i s kojim je usko povezan. Ideja koja stoji iza ***XDP***-a nalazi se unutar sâmog upravljačkog programa mrežne kartice, logički nakon dijela za obradu signala prekida (***IRQ***) s kojim se i okida dohvaćanje mrežnih paketa, ali prije alociranja memorijskog međuspremnika na razini operativnog sustava. Dakle ovdje ne govorimo o ***RX*** međumemoriji na sâmoj mrežnoj kartici, kako je upisano u **poglavlju: 25.2.1.**.

Naime kreiranje i baratanje s međumemorijama za rad s mrežnim paketima na razini operativnog sustava je i najsporiji proces. Stoga je kod *XDP*-a bilo potrebno unutar upravljačkog programa dodati takozvani mehanizam kuke (engl. *Hook*) s kojom se mrežni paket u dolazu, odmah nakon što ga je mrežna kartica zaprimila, može proslijediti dalje na ***eBPF*** mehanizme za obradu.

Zbog ovog dizajna **XDP** postiže drastična ubrzanja u dohvaćanju mrežnih paketa, ali i u kasnijoj obradi pomoću ***eBPF*** filtera.

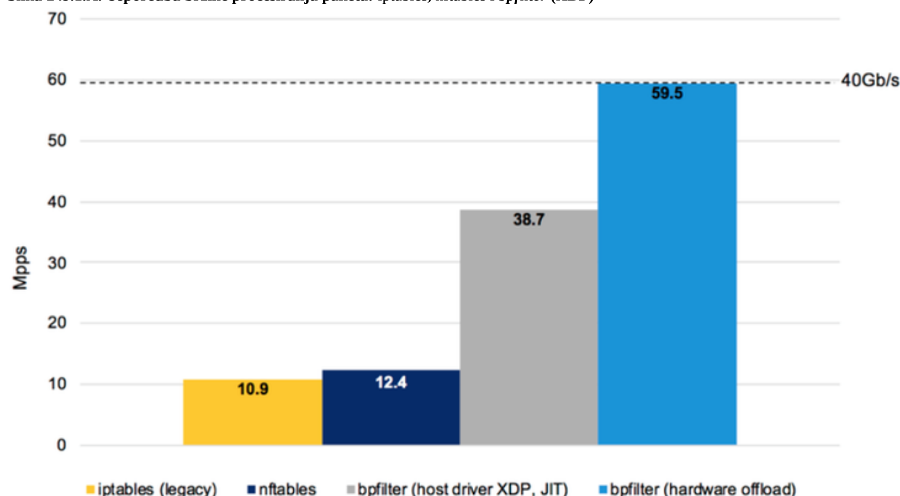
**Slika 249.1. Pogled na obradu mrežnih paketa unutar mrežnog stoga Linuxa upotrebom XDP i eBPF mehanizama.**




**BPF**
Čak i bez **XDP**, kada imamo potrebu maksimalno iskoristiti mogućnost obrade mrežnih paketa, na datom hardveru, *Extended Berkeley Packet Filter* (**eBPF**) je prirodan izbor. ***eBPF*** radi tako da je s njim moguće umetnuti program za obradu mrežnih paketa na razinu izvršavanja samog Linux kernela, dajući vam pri tome fleksibilnost izrade mrežnih filtera uz brzinu izvršavanja koja se dobiva na razini Linux kernela.

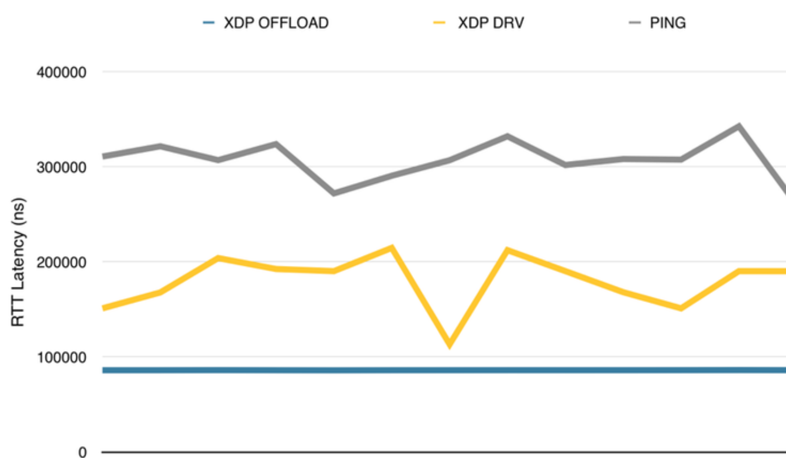
**XDP + BPF**
Prema nekim mjerenjima[830] ***XDP*** na prosječnom hardveru, na sâmo jednoj jezgri procesora postiže brzinu dohvaćanja i konkretno odbacivanja (osnovni filter [**XDP_DROP**]) 25 milijuna paketa u sekundi (25Mpps). A prema drugim mjerenjima[829] vidljivim na slici **249.1.A.** vidimo i konkretne rezultate usporedbe rada klasičnog ***iptables*** filtriranja (*vatrozida*) koji na navedenoj hardverskoj konfiguraciji uspijeva obraditi 10.9Mpps (milijuna paketa u sekundi [**poglavlje: 20.4.**]) dok novija implementacija Linux vatrozida ***nftables*** postiže malo bolje rezultate od 12.4Mpps.
Međutim, ako se u analizu uključi i ***XDP*** sa ***eBPF*** filterom, dakle sâmo s promjenom upravljačkog programa (Engl. *Drivera*) koji podržava ***XDP***, dobivamo povećanje na 38.7Mpps što je skok od tri i pol puta (x**3.5**) u odnosu na ***iptables*** odnosno staru funkcionalnost upotrebe klasičnog ***Netfilter*** podsustava Linuxa.

**Slika 249.1.A. Usporedba brzine procesiranja paketa:** iptables, nftables i ***bpfilter*** **(XDP)**




Na istom mjerenju, na slici 249.1.A. vidimo i još jednu statistiku označenu kao ***bpfilter (hardware offload)*** koja također koristi ***XDP*** sa ***eBPF*** filterom, ali na posebnim mrežnim karticama koje su u mogućnosti hardverski ubrzavati (odrađivati) ***eBPF*** filtriranje, pa su stoga i rezultati još bolji. Dakle ovdje se radi o posebnom hardveru integriranom u mrežne kartice.

Ovdje vidimo 59.5Mpps, što je skok od pet i pol puta (x**5.5**) u odnosu na ***iptables*** jer sve operacije filtriranja zapravo obavlja sâma mrežna kartica (njen hardver) bez prebacivanja podataka preko sabirnice ili posredovanja centralnog procesora (CPU-a).
Na drugoj statistici na slici **249.1.B.** vidimo kašnjenje (latenciju) koje standardno unosi obrada paketa u softveru (***XDP*** upravljački program + ***eBPF*** obrada). Dakle na slici **249.1.B.** imamo srednju vrijednost vidljivu kao ***XDP DRV*** u odnosu na onu koju unosi mrežna kartica, ako ima hardversku podršku za ***XDP+eBPF*** pa ne začuđuje da je prednost na strani hardverski ubrzanih operacija, koje daju specijalizirane mrežne kartice.




**Slika 249.1.B. Usporedba latencije: XDP softverski / XDP hardverski (*Offload*)**

Testovi sa slika **249.1.A.** i **249.1.B.** rađeni su na ***Netronome*** mrežnim karticama serije: *Agilio CX SmartNIC* **(832)** koje imaju takozvani **NFP** mrežni procesor (Engl. *Network Flow Processor*) istoimene tvrtke *Netronome* **(831)**.
Drugi veliki proizvođači mrežnih sklopova poput tvrtki ***Intel*** **(827),(833)** i ***Mellanox*** također rade na razvoju sličnih rješenja.

**Izvor mjerenja i fotografija 249.1.a, 249.1.B. je: (829)**

# Priča o DPDK i Netmapu

Iako smo do sada spominjali kombinacije softvera i hardvera odnosno upravljačkih programa, moguće su i dostupne razne tehnike i tehnologije ubrzavanja obrade mrežnih paketa, upotrebom specijaliziranih sistemskih biblioteka, koje su visoko optimizirane za brzu obradu mrežnih paketa, poput takozvanog **DPDK**.

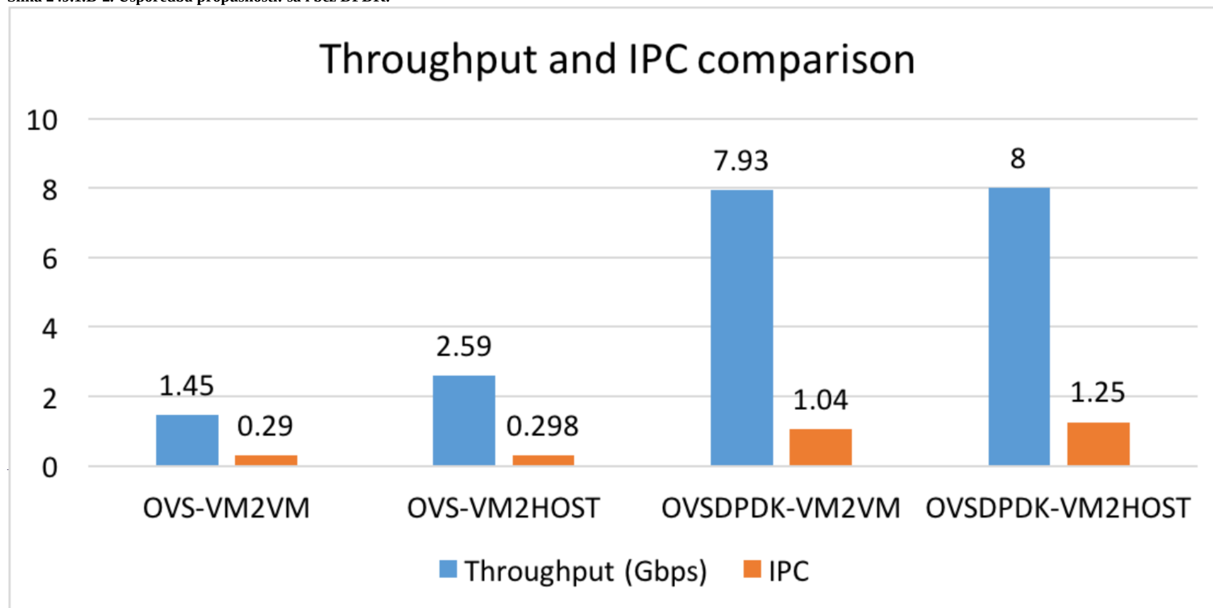
Naime upotrebom *DPDK* moguće je unutar sâmo **80 CPU** ciklusa, modernih procesora i mrežnih kartica s pripadajućim (modificiranim) upravljačkim programima, zaprimiti ili poslati pojedini mrežni paket. Dakle *Data Plane Development Kit* (***DPDK***) je softverski projekt otvorenog koda kojim upravlja *Linux fondacija*. On pruža skup programskih biblioteka u kombinaciji s upravljačkim programima koji podržavaju tzv. *pooling* način dohvaćanja mrežnih paketa (okvira) od fizičkog mrežnog sučelja, rasterećenjem obrade paketa s kernela operativnog sustava na procese (programe) koji se izvode u korisničkom prostoru operativnog sustava.

To znači da ***DPDK*** zaobilazi veći dio kernela i cijeli mrežni stôg Linuxa pa se s ovakvim rasterećenjem postiže veća računalna učinkovitost te drastično veća propusnost dohvaćanja mrežnih paketa nego što je to moguće korištenjem klasične metode rada s mrežnim paketima. Nadalje, *DPDK* ima bolju optimizaciju međumemorije, te optimizaciju za **NUMA** sustave, ali i mogućnost upotrebe velikih stranica memorije (tzv. **Hugepages**).

***DPDK*** programski okvir (*framework*) nudi skup programskih biblioteka za određena hardverska i softverska okruženja kroz stvaranje sloja takozvane apstrakcije okruženja (**EAL** [engl. *Environment Abstraction Layer*]). **EAL** sloj skriva specifičnosti okruženja sustava i pruža standardno programsko sučelje za biblioteke, dostupne hardverske akceleratore i druge elemente hardvera i operativnog sustava (pr. **Linux**, **FreeBSD**). Nakon što je **EAL** kreiran za određeno okruženje, programeri se povezuju s bibliotekom za izradu svojih aplikacija. **EAL** također pruža dodatne usluge uključujući vremenske reference, generički pristup sabirnici, funkcije praćenja i otklanjanja pogrešaka i operacije alarma. Sve navedeno ubrzava proces dohvaćanja mrežnih paketa. Primjerice napredni virtualni preklopnik ***Open vSwitch*** može koristiti ***DPDK.***

Pogledajmo i rezultate mjerenja propusnosti mreže, bez upotrebe i s upotrebom **DPDK** preko **Open vSwitch** sustava.

**Slika 249.1.B-2. Usporedba propusnosti: sa i bez DPDK.**




Iz mjerenja su vidljiva sljedeća povećanja propusnosti: ***Open vSwitch*** vs. ***Open vSwitch + DPDK***:

- Od **5.5.** puta: *virtualno računalo - virtualno računalo* (**VM2VM**).
- Dok je povećanje propusnosti od **3.** puta u smjeru: *fizičko računalo (Host)-virtualno računalo* (**VM2HOST**).
- Također je vidljiva i bolja učinkovitost po pitanju izvođenja procesorskih instrukcija (**IPC [**engl. *instructions per cycle***]**).



Vezano za **DPDK** i **Open vSwitch** pogledajte poglavlja:
**26.8.5. Open vSwitch servis.**
**26.8.6. Usporedba mrežnih tehnologija hipervizora za virtualizaciju.**

## Netmap

Sljedeći primjer je **netmap** *framework*, koji je također razvijen za sličnu namjenu, a koji omogućava programima mogućnost procesiranja i prosljeđivanja mrežnih paketa vrlo velikim brzinama (unutar **90 CPU** ciklusa), korištenjem *netmap* **API** poziva.
**Netmap** *Framework* se danas standardno koristi i na **FreeBSD Unixu** (još od inačice **9.1**), ali sve više i na Linuxu, na kojem je integriran i sa **KVM/QEMU** sustavom za virtualizaciju. Jedan od primjera je i **PF RING** koji je razvijen za potrebe brzog dohvaćanja, filtriranja i analize mrežnih paketa.
Najviše za potrebe programa za analizu mrežnog prometa, poput programa ***ntop*** odnosno ***ntopng***. Drugi primjer je upotreba kombinacije novog mehanizma za dohvaćanje mrežnih paketa, unutar upravljačkog programa mrežne kartice, već spomenutog naziva **XDP**, a dodatno i upotrebom pripadajućih eBPF filtera o kojima smo govorili.



Pogledajte poglavlje:
**9.3.1. Context switching ,** a posebno cjelinu **"Vrijeme obrade".**

**Postoji li veza između *eBPF*, *iptables* i *nft*?**
Zapravo postoji. Naime i ***iptables*** i ***nft*** u novijim inačicama mogu koristiti ***eBPF*** filtere, što u određenim okruženjima može pojednostaviti uporabu ***eBPF*** filtera. Konkretno ***iptables*** koristi ***xt_bpf*** proširenje za podudaranje s ***eBPF*** filterima.
Ovo proširenje koristi tip programa `BPF_PROG_TYPE_SOCKET_FILTER` ***eBPF***, koji nam omogućuje da učitamo informacije o paketu iz međuspremnika *socketa* (*socket buffer*) i vratimo vrijednost na temelju našeg (***eBPF***) filtera odnosno kôda.
Pogledajmo kako bi izgledalo pozivanje ***eBPF*** filtera iz programa odnosno naredbe `iptables`:

```
iptables -A INPUT -m bpf --object-pinned /sys/fs/bpf/filter1 -j DROP
```

Pri tome je `filter1` *eBPF* filter (**1126**) koji primjenjujemo.



***Isto je moguće koristiti i s novijim Nftables sustavom.***

## NFP procesori

Pogledajmo i referentni dizajn jednog **NFP** procesora (serije *NFP-6xxx*) tvrtke ***Netronome***, na slici: **249.1.c.**

**Slika 249.1.C. NFP 6xxx procesor tvrtke Netronome.**

**Izvor fotografije: (831)**

Ovaj specijalizirani mrežni procesor (**NFP**) osim hardverskog ubrzanja i podrške za **eBPF** filtere, hardverski ubrzava i ***OpenvSwitch*** značajke, ***OpenFlow*** te podržava ***DPDK*** i ***SR-IOV***, ali i operacije koje se tiču Linux **KVM** hipervizora (mrežni dio) te kriptografske i mnoge druge operacije i protokole, poput primjerice: ***VxLAN***, ***NVGRE***, ***MPLS***, … .



Vezano za kriptografsku akceleraciju, pogledajte i poglavlje:
**26.4.4.2. Upotreba hardverski ubrzanih kripto funkcija.**

U ovakvom dizajnu specijaliziranog (ko)procesora, sa slike 249.1.C., jasno je vidljivo da na ulazu imamo mrežna sučelja velikih propusnosti (10/40/100+Gbps) te brzu vezu prema *PCI Express* sabirnici. S druge strane imamo hardverski ubrzane operacije i mehanizme potrebne za spremanje, obradu i filtriranje (*Queue*, *Hash*, *CAM*, …). I na kraju je vidljiv integrirani memorijski kontroler, pa se daje naslutiti da mu je veća RAM memorija na drugom sklopu.

Dok je za sâmu obradu paketa zaduženo 96 specijaliziranih CPU jezgri („***Packet processing cores***“), odnosno 120 takozvanih „***Flow processing* cores**“ CPU jezgri. Sve zajedno, u jednom specijaliziranom sklopu, imamo nevjerojatno snažan hardver za prihvaćanje, obradu i filtriranje mrežnih paketa koji „*hranimo*“ s pravilima i filterima iz Linuxa odnosno iz ***eBPF*** pravila filtriranja. Detalje konfiguracije ***eBPF*** nećemo duboko objašnjavati, ali važno je razumjeti da od kernela ***4.18*** standardno dolaze i alati (naredbe) s kojima možete koristiti ***iptables*** pravila, koja se konvertiraju u ***eBPF*** pravila te se dalje primjenjuju, uz naravno standardno korištenje novih ***eBPF*** pravila odnosno filtera. Ovakav sklop sa svim pratećim komponentama se obično implementira u obliku specijalizirane *PCI express* kartice.

## DPU procesori (općenito)

Jedinica za obradu podataka to jest ***DPU*** (engl. *data processing unit*) je novost u svijetu obrade podataka, koja se sve više primjenjuje. I prethodno navedeni ***NFP*** procesor je zapravo podvrsta *DPUa*, ali specijaliziranog za mrežne protokole.
*DPU* je programabilni specijalizirani elektronički sklop s hardverskim ubrzanjem obrade podataka za računalstvo usmjereno na podatke. Kod njega odnosno do njega se podaci prenose kao ***multipleksirani**** paketi informacija.
*DPU* obično sadrži CPU, mrežnu karticu i programabilne mehanizme za ubrzanje obrade podataka. To omogućuje DPU-ovima da imaju općenitost, ali i mogućnost programiranja središnjih procesorskih jedinica dok su specijalizirani za učinkovit rad sa mrežnim paketima, zahtjevima za pohranu podataka ili analitičkim zahtjevima.
Pri tome se *DPU* razlikuje od CPU-a po većem stupnju paralelizma (potrebno za obradu višestrukih i paralelnih zahtjeva), a od GPU-a po ***MIMD**** arhitekturi, a ne po ***SIMD*** arhitekturi, što je potrebno jer se unutar svakog zahtjeva mogu donositi različite odluke te slijediti drugačiji put kroz sam elektronički sklop. DPU-ovi mogu biti bazirani na ***ASIC***, ***FPGA*** ili ***SoC*** tehnologijama.

Naime, DPU-ovi se sve više koriste u podatkovnim centrima i superračunalima od njihovog uvođenja 2010-ih zbog porasta upotrebe računalstva usmjerenog na podatke, velikih podataka, sigurnosti i umjetne inteligencije/strojnog učenja/dubokog učenja. DPU-ovi su obično dizajnirani da budu neovisne krajnje točke infrastrukture.

Neki od proizvođača DPU-ova su:

- **AMD** s proizvodima: *Capri*, *ELBA* i *DSC*.
- **Broadcom**, sa *Stingray* DPUom.
- **Intel**, s *Infrastructure Processing Unit* (***IPU***),
- **Kalray** s *Kalray K200-LP*.
- **Marwell Technology,** s *OCTEON* i *ARMADA* DPUovima
- **Nvidia/Mellanox Technologies**, s *BlueField*, *ConnectX* i *Innova* DPUovima.



***Moguće su i razne kombinacije: CPU, DPU (pr. NFP), XDP, eBPF i DPDK te drugih programa i servisa!***

***Primjerice, moguće je kombinirati CPU i DPU s DPDK i sa FRRouting** (FRR) **servisom za usmjeravanje. Tako je moguće ekstremno ubrzati svo usmjeravanje i upotrebu protokola za usmjeravanje (pr. OSPF, RIP, BGP, …).***



Pogledajte poglavlje:
**21.4.3. Upotreba protokola za umjeravanje pod Linuxom.**

***Multipleksiranje**** je metoda s kojom se više podataka ili signala kombinira u jedan nîz podataka.
***MIMD**** (engl. *multiple instruction, multiple data*) arhitektura je zadužena za paralelizam u izvršavanju instrukcija.



Pogledajte plakat cijelog mrežnog modela Linuxa: P5.

**Izvori informacija: (823),(824),(825),(826),(827),(828),(829),(830),(831),(832),(833),(1126),(1127), (1303), (1304),(1305),(1306),(1307),(1460),(1577),(1578),(1579),(1580),(1581),(1589).**

### 26.7.3.3.1. Upotreba i konfiguracija eBPF filtera (pravila) vatrozida

U ovoj cjelini proći ćemo osnovnu konfiguraciju i upotrebu novih ***eBPF*** pravila odnosno filtera za vatrozid, a koji koristi ***XDP*** funkcionalnost. Pri tome nećemo objašnjavati konfiguraciju specijaliziranih mrežnih kartica koje nude hardverski ubrzane (akcelerirane) ***XDP***/***eBPF*** funkcionalnosti. Ako vas i to zanima, za ***Netronome*** mrežne kartice pogledajte izvor: **(834)**.
Važno je razumjeti kako ***eBPF*** filter očekuje filtere u posebnom binarnom formatu. Stoga ćemo za pisanje pravila morati koristiti neki od dobro razvijenih programa, koji će naša pravila (koja ćemo nadalje zvati ***BPF*** pravila odnosno filtere) pretvoriti u potrebni binarni oblik. Mi ćemo koristiti ***bcc*** programe i pripadajuće alate koje ćemo instalirati sa:

```
yum -y install bcc bcc-tools bpftool
```

Sada ćemo dobiti nove programe, alate te biblioteke potrebne za dalji rad.



Puna funkcionalnost ***XDP*** i ***eBPF*** filtera je dostupna tek od *RedHat/CentOS* inačice **8.1**.x ili novijih.

Naime ***eBPF*** (prevedimo ga kao prošireni „*Berkeley Packet Filter*") koristi virtualni stroj u kernelu koji omogućuje izvršavanje kôda u prostoru kernela, u posebno izoliranom okruženju (Engl. *Sandbox*) s ograničenim pristupom s ograničenim skupom funkcija. Kôd o kojem ovdje govorimo su zapravo ***eBPF*** filteri odnosno pravila koja ćemo koristiti za filtriranje mrežnog prometa odnosno za funkcionalnosti vatrozida. Ovaj kôd se zatim učitava u kernel i prevodi se u strojni kôd uz istovremeno kompiliranje tog kôda (Engl. *Just-in-time compilation*), kako bi se uopće mogao izvršiti.

Virtualni stroj potom izvršava taj poseban kôd nalik asemblerskom programskom kôdu. Važno je znati da postoje brojne komponente koje se isporučuju uz sustav, a koje koriste ili mogu koristiti ***eBPF*** virtualni stroj.
U našem slučaju ***bcc*** skup alata i biblioteka, kao i *Python*/*Lua* modula koje ćemo koristiti (Engl. *BPF Compiler Collection*) je skup alata za dinamično praćenje stanja rada kernela koji koriste ***eBPF*** virtualni stroj. Ovaj skup ***bcc*** alata se automatski instalirao u direktorij `/usr/share/bcc/tools/` pa ih od tamo moramo i pozivati odnosno pokretati.
Tako recimo s naredbom: `bpflist` možemo vidjeti koristi li neki program *eBPF* filtriranje. Stoga to i provjerimo:

```
bpflist
PID    COMM               TYPE     COUNT
```

Vidimo da nemamo niti jedan takav program odnosno proces.
Istu funkcionalnost možemo dobiti i s naredbom `bpftool prog show`.

Nadalje, a zapravo prema logici rada na nižoj razini, upotrebu ***XDP*** mehanizama možemo vidjeti na razini mrežnog sučelja.
Naime mrežno sučelje na kojem se koristi određeni program koji i koristi *XDP* bit će posebno naznačeno, što možemo vidjeti s naredbom `ip link` na sljedeći način:

```
ip link show

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode
DEFAULT group default qlen 1000
    link/ether 08:00:27:3a:73:34 brd ff:ff:ff:ff:ff:ff
```

Konkretno kada imamo neki program koji bi na mrežnom sučelju `enp0s3` koristio *XDP* ovdje bi vidjeli zastavicu `xdp`.
Sada možemo napraviti svoj filter u programskom jeziku ***C***, izraditi objektni programski kôd (pr. datoteku: `xdp-ebpf.o`) i primijeniti ga za filtriranje mrežnih paketa i aplicirati ga na ***XDP*** razini, na željenom mrežnom sučelju.
Primjena objektnog kôda odnosno filtera bi otprilike bila sljedeća (ako je filter datoteka imena: `xdp-ebpf.o`):

```
ip link set dev enp0s3 xdp obj xdp-ebpf.o
```

I sada ćemo vidjeti sljedeće (skratili smo ispis):

```
ip link show

2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 xdp qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
```

Vidimo da se *XDP* filter (*eBPF* pravila) primjenjuje jer imamo (`xdp`) na ovom mrežnom sučelju (`enp0s3`).
Navedeni filteri se izvorno moraju pisati u ***C*** jeziku, te se on sa programima za kompiliranje ***C*** programskog kôda, poput: `gcc`, `clang` ili sličnih mora pretvoriti u objektni bajt kôd. U našem slučaju bi to napravili sa izvornim ***C*** kôdom; datotekom imena `xdp-ebpf.c` u izlaznu datoteku imena `xdp-ebpf.o` na sljedeći način.

Dakle s naredbom `clang`:

```
clang -O2 -target bpf -c xdp-ebpf.c -o xdp-ebpf.o
```

S parametrom `-target bpf` smo naznačili da se ***C*** programski kôd mora pretvoriti u *BPF* bajt kôd, što ćemo kada se konverzija napravi vidjeti s naredbom `readelf` na sljedeći način (skratili smo ispis):

```
readelf -h xdp-ebpf.o

ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Machine:                           Linux BPF
```

Vidimo da se sada radi o e(*BPF*) kôdu (`Linux BPF`) koji je spreman za apliciranje. Sam ***C*** kôd za filtere nećemo objašnjavati jer bi za to trebali puno vremena (i prostora). Teka sada možemo taj *eBPF* kod aplicirati na mrežnu sučelje na sljedeći način:

```
ip link set dev enp0s3 xdp obj  xdp-ebpf.o
```

I to je to, filter se od tog trena primjenjuje na mrežnom sučelju `enp0s3`.

Za brisanje ***eBPF*** pravila na istom mrežnom sučelju treba napraviti sljedeće:

```
ip link set dev enp0s3 xdp off
```

Važno je razumjeti da je eBPF zapravo skup biblioteka i u konačnici programsko sučelje (*framework*), koje nam daje metode i instrukcije **(839)** s kojima pravimo svoja ***eBPF*** pravila (filtere), koja se primjenjuju na ***XDP*** razini.

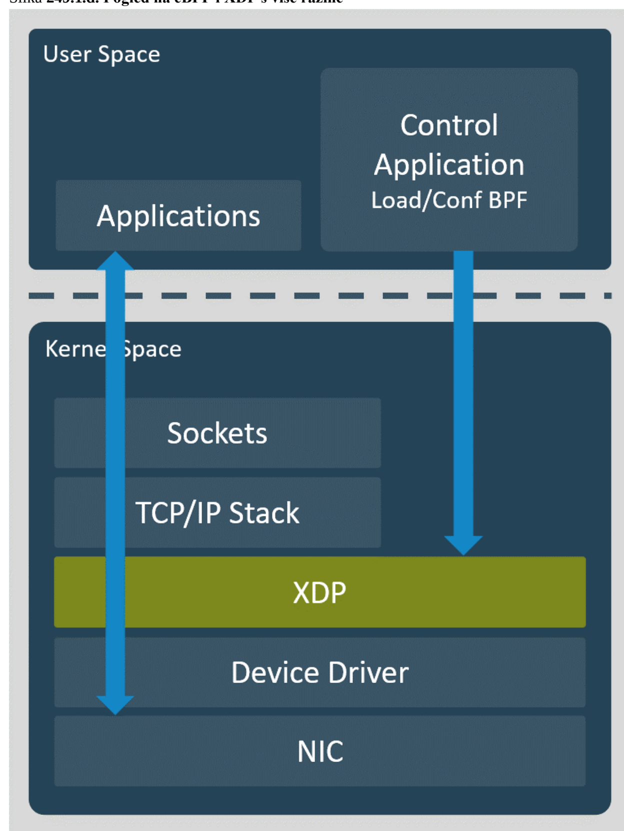
I naredba `tc` (Engl. *Traffic control*) nudi mogućnost korištenja ***eBPF*** filtera. Ova naredba dolazi u softverskom paketu imena: `iproute-tc` odnosno možemo ju instalirati na sljedeći način:

```
yum install tc -y
```

Za detalje primjene pogledajte izvor informacija **(842).**

Pogledajmo sada i cijelu logičku shemu ***XDP***, ***eBPF*** i Linux mrežnog stoga na malo drugačiji način, na slici **249.1.d.**

Slika **249.1.d. Pogled na eBPF i XDP s više razine**




Vidljivo je da se što se aplikacije tiče, ***XDP*** sloj nalazi između TCP/IP mrežnog stôga linuxa i upravljačkog programa mrežne kartice.
Isto tako može se primijetiti da je ***eBPF*** kontrolna aplikacija ta koja *hrani* **XDP** s filterima odnosno pravilima filtriranja.

Dodatno je važno znati da kada napravimo svoje pravilo filtriranja (filter) on prvo prolazi kroz mehanizme provjere, koji provjeravaju da program ne sadrži petlje, globalne varijable ili druge problematične stvari.

Tada, ako je sve u redu i filter je primijenjen, za svaki paket se prolazi kroz filter(e) te se odlučuje što s njim dalje:

- Propustiti ga kroz mrežni stôg (***TCP/IP Stack*** na slici).
- Odbaciti paket (*drop*).
- Vratiti ga nazad na isto mrežno sučelje s kojeg je došao.
- Prebaciti ga na poseban *socket*, prema razini korisničkih aplikacija (*User level*), odnosno prema **AF_XDP (843)** sučelju.

**Izvor fotografije: (840)**

U novije vrijeme, od 2019. godine, pojavljuje se sve više rješenja koje žele pojednostaviti pisanje ***eBPF*** filtera. Jedan od takvih projekata je **bpfilter (841), (844),** koji je jedna od implementacija koja nudi pretvaranje ***iptables*** pravila u ***eBPF*** pravila u letu, uz primjenu istih na ***XDP*** razini bez ikakvog posredovanja korisnika.



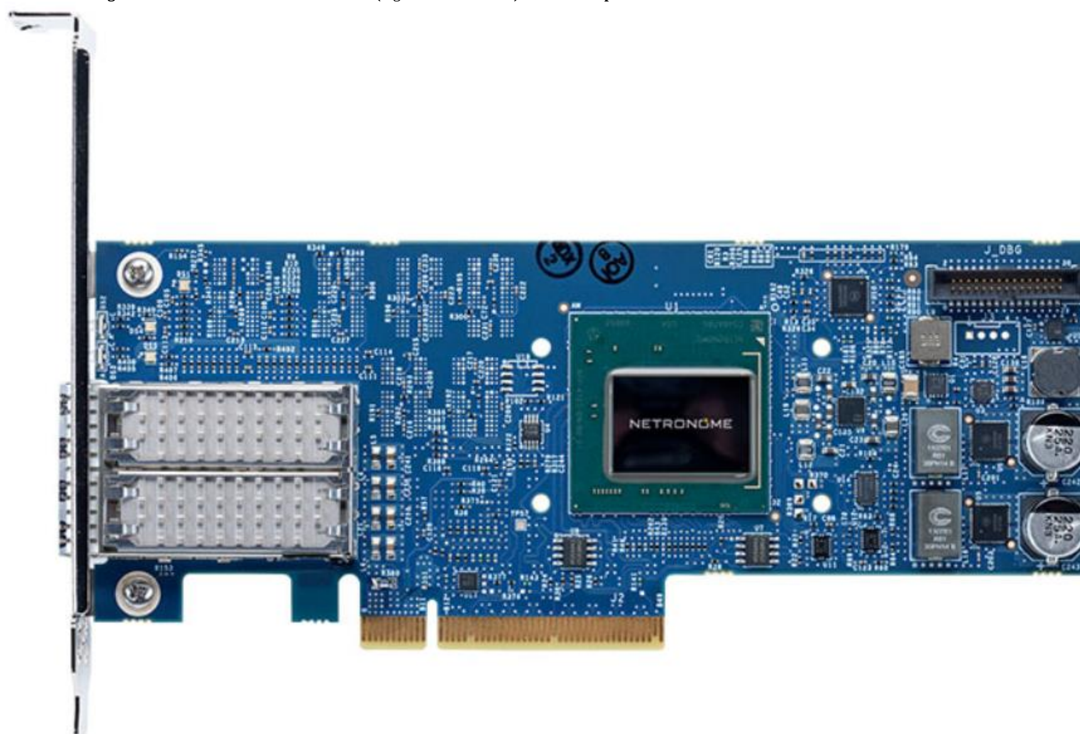
Pogledajte i video koji objašnjava upotrebu BPFiltera: https://www.youtube.com/watch?v=AfgwVya9Cog

Pitanje je sâmo vremena koji od takvih projekata će na kraju zaživjeti i uspjeti na što jednostavniji način migrirati postojeća *iptables* pravila u nova *eBPF*, bez imalo intervencije korisnika. Ipak činjenica je da danas (2020.g.) svi veći korisnici, poput tvrtki: *Facebook*, *Netflix* i drugih već koriste ***eBPF*** i ***XDP*** u produkciji. Pogledajte i projekat Linux zaklade imena ***IO visor*** koji objedinjuje tehnologije ***XDP***, ***eBPF*** i ***BCC***, a čiji članovi su tvrtke poput: ***Huawei, VMware, Facebook, Netronome, SuSe*** i ***Ubuntu***.
Dodatno, i mnogi drugi sustavi koriste ***XDP*** funkcionalnost za brzu obradu mrežnih paketa, poput virtualnog preklopnika imena ***OpenvSwitch*** **(845)** te **ntopNG** **(846)** sustava za dohvaćanje i analizu mrežnog prometa, IPS/IDS programa ***Suricata*****(847)** kao i nekih drugih. Nadalje, primjerice *OpenvSwitch* (**OVS**) podržava ***DPDK*** način rada, što je posebno važno u virtualizaciji.

Nadalje, ako malo detaljnije pogledamo sliku nekoliko stranica prije (**249.1.**) možemo primijetiti da se između ***eBPF*** filtera i sâmog ***XDP*** dijela nalazi posebno specijalizirano ***eBPF*** virtualno računalo (**eBPF VM**) koje zapravo prevodi ***eBPF*** filtere koji usput rečeno koriste posebne ***RISC*** instrukcije, u instrukcije koje kernel razumije i može ekstremno brzo izvršavati. Dakle ***eBPF VM*** može komunicirati direktno s kernelom**K**. Zbog toga je moguće osim funkcionalnosti filtriranja mrežnih paketa, filtrirati i obrađivati i druge vrste poruka. To znači da je moguće napraviti i ***eBPF*** mehanizme i za druge namijene, poput praćenja i analize rada programa, u svrhu analize njihovog rada i performansi. Jedan od takvih programa je program imena `bpftrace` koji je praktično performantnija (učinkovitija) zamjena za programe tipa: `perf`, `strace`, `dstat` i sličnih.

U prijašnjem poglavlju, vidjeli smo primjere upotrebe specijaliziranih kartica, poput ***NFP*** kartice tvrtke ***Netronome*** (slika **249.1.e.**) te primjenu ***DPU*** kartica, a još jedna od zanimljivih ideja je upotreba današnjih grafičkih kartica za obradu mrežnih paketa.

**Slika 249.1.e. Pogled na NFP karticu tvrtke Netronome (**Agilio CX 2x10GbE**) s NFP 4000 procesorom.**




Naime današnje grafičke kartice već standardno imaju na stotine posebnih procesorskih jezgri, koje se u teoriji mogu iskoristiti za obradu mrežnih paketa. Projekt koji želi iskoristiti njihovu snagu obrade i brzo dohvaćanje te procesiranje mrežnih paketa, zove se ***NetGPU*** **(911).**
Njegova osnovna ideja rada je u tome da podržava ***DMA*** prijenos bez kopiranja između mrežnog adaptera/kartice i grafičkih procesora, a može koristiti ***eBPF*** za svo filtriranje mrežnog prometa.



Pogledajte i poglavlje:
**15. Analiza rada linux sustava i programa.**

**Izvori informacija: (834),(835),(836),(837),(838),(839),(840),(841),(842),(843),(844),(845),(846), (847),(911),**
**`man ip link`,`man tc`,`man readelf`.**

# 26.8. Napredne mrežne tehnologije

U ovoj cjelini ukratko ćemo se upoznati s naprednim mrežnim tehnologijama i objasnit ćemo čemu one služe.

### VLAN-ovi odnosno virtualne lokalne mreže

**VLAN** čini logičku grupu ili cjelinu raznih mrežnih komponenti: računala, mrežnih uređaja i slično. *VLAN mreže* se obično grupiraju prema nekim logičkim podjelama mreže: primjerice prema odjelu tvrtke, prema specifičnim zahtjevima korisnika, poslužitelja i slično. Mrežni preklopnici (*switchevi*) i mrežni mostovi (*bridgevi*) propuštaju: *unicast*, *multicast* i *broadcast* promet samo unutar VLAN mreže unutar koje se nalaze. Upotrebom *VLANova* komunikacija je moguća sâmo između svih mrežnih uređaja i računala unutar jedne VLAN mreže. Promet odnosno komunikacija između *VLANova* nije moguća bez uređaja koji rade na osi sloju tri (OSI 3); dakle bez usmjerivača ili *Multilayer* preklopnika koji su konfigurirani za tu namjenu.



Za više detalja oko ***VLANova*** pogledajte poglavlje: **20.6.2. VLANovi odnosno virtualne lokalne mreže.**

### Mrežni mostovi odnosno premosnici (*bridge-vi*)

Mrežni mostovi znani i kao premosnici (Engl. *Bridge*) su uređaji ili tehnologija koja se koristi kako za segmentiranje tako i povezivanje više segmenata mreža na OSI sloju dva (OSI sloj 2) mreže. Linux računala mogu raditi kao mrežni mostovi.



Za više detalja oko mrežnih mostova te njihove konfiguracije i upotrebe pogledajte poglavlje:
**20.6.1. Mrežni most (*bridge*).**

### *Agregacija/Bonding/Team-ing*

Pojmovi *Bonding*, *Agregacija*, *Team* ili *Ether Channel* označavaju razne metode logičkog povezivanja više fizičkih mrežnih kartica u jednu logičku mrežnu karticu. Ta logička mrežna kartica ima veću ukupnu propusnost (engl. *Throughput*) od svake pojedine fizičke mrežne kartice te omogućava redundanciju odnosno ispad jedne (ili više) fizičkih mrežnih kartica bez prekida u radu na razini logičke mrežne kartice, a sâmim time i mrežne komunikacije.



Za više detalja o ovoj tehnologiji pogledajte poglavlje:
**20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).**

---

## 26.8.1. Network *Namespaces*

Izolirani mrežni prostor *Engl. Network Namespace* koristi se za kreiranje potpuno odvojene i izolirane virtualne mreže unutar fizičkog Linux računala. Izolirani mrežni prostor je sâmo dio Linuxovog izolacijskog prostora (*namespacea*) koji nam daje i druge vidove izolacije. Svaki izolirani mrežni prostor se ponaša poput mreže svakog zasebnog (virtualnog) računala, a unutar njega postoje:

- Mrežna sučelja i IP parametri mreže (IP adresa i maska mreža, podrazumijevani usmjerivač, …), kao i mrežni portovi.
- Rute i tablice usmjeravanja (*routing tablice*).
- Vatrozid i svi drugi mrežni elementi koji se nalaze u svakom „normalnom“ računalu.

Dakle možemo reći i kako je ovo vrsta virtualizacije mreže, ali unutar jednog Linux OS-a i njegovog kernela. Ova tehnologija se najčešće koristi za izolaciju mreže unutar raznih Linux kontejnera, poput: **LXC** i **OpenVZ** kontejnera ili **Docker**-a.
Više informacija o navedenim linux kontejnerima pogledajte na sljedećim poveznicama:

- ***LXC***: https://linuxcontainers.org/
- ***OpenVZ***: https://openvz.org/Main_Page
- ***Docker***: http://www.docker.com/



Vezano za općeniti rad Linuxovih izoliranih prostora (*namespace*-ova) pogledajte i poglavlje:
**9.4.5 Izolirani prostori Linuxa (*Linux namespaces*).**

Pogledajmo još nekoliko činjenica o izoliranim mrežnim prostorima Linuxa odnosno o tzv. *Network Namespace-ovima*:

- Svaki mrežni izolirani prostor Linuxa (*Namespace)* ima svoj *loopback* uređaj.
- Virtualni ili fizički mrežni uređaji se mogu dodavati u bilo koji *namespace* te im se može dodijeliti IP adresa.
- Network *Namespace* dijeli isti datotečni sustav s fizičkom instancom Linuxa unutar kojeg se koristi.

Pogledajmo i primjere upotrebe izoliranih mrežnih prostora (*network namespaces*) unutar Linuxa.
Kreirat ćemo jedan mrežni izolirani prostor (*network namespace*) imena `PRVI` pomoću naredbe `ip` na sljedeći način:

```
ip netns add PRVI
```

Svaki novi izolirani mrežni prostor ima samo „*loopback*“ mrežno sučelje pa stoga nije povezan niti s našim Linuxom unutar kojeg je kreiran odnosno „pokrenut“.

Navedeni izolirani mrežni prostor bi obrisali sa sljedećom naredbom (to ovdje nećemo napraviti):

```
ip netns del PRVI
```

Listu izoliranih mrežnih prostora Linuxa možemo vidjeti s naredbom:

```
ip netns list
PRVI
```

Svi izolirani mrežni prostori se zapisuju unutar direktorija `/var/run/netns/` kao posebne datoteke.
Praćenje kreiranja ili brisanja izoliranih mrežnih prostora u stvarnom vremenu možemo napraviti sa sljedećom naredbom:

```
ip netns monitor
```

Sada ćemo ući u ljusku (***bash** shell*) unutar našeg izoliranog mrežnog prostora koji smo kreirali (*namespace* imena `PRVI`):

```
ip netns exec PRVI bash
```

Sada kada smo u ljusci (shellu) našeg *namespacea* pogledajmo koja mrežna sučelja imamo vidljiva (skratili smo ispis):

```
ifconfig -a
lo: flags=8<LOOPBACK>  mtu 65536
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
```

Ovdje vidimo samo „*loopback*“ mrežno sučelje odnosno sučelje imena `lo`.
Ako sada (naredba: `lsns`) pregledamo listu svih izoliranih prostora, vidjeti ćemo na kraju ispisa (koji smo skratili), sljedeće:

```
lsns
        NS TYPE  NPROCS   PID USER   COMMAND
4026532212 net        2  1211 root   bash
4026532298 mnt        2  1211 root   bash
```

Zadnja dva ovdje vidljiva unosa su naši izolirani mrežni prostori (*network namespaces)* te vidimo ***PID*** brojeve `1211` od naše ***bash*** ljuske koju smo pokrenuli unutar tog izoliranog mrežnog prostora. Isto možemo povezati prema ***PID*** broju te vidjeti sve izolirane prostore Linuxa za taj proces (***PID***: `1211`). Dakle u konkretnom slučaju unutar direktorija `/proc/1211/ns/`.
Pri tome je poveznica za mrežni izolirani prostor ***NS*** broj: `4026532212`.
Što se mreže unutar našeg izoliranom mrežnog prostora tiče, ovdje imamo nekoliko opcija:

- Možemo našu fizičku mrežnu karticu vidljivu pod linuxom prebaciti unutar našeg izoliranog mrežnog prostora i dalje nastaviti raditi unutar njega. Pri tome nam više ništa od mrežnih servisa neće raditi na Linuxu odnosno u normalnom mrežnom okruženju, pa ćemo sve morati prebaciti unutar tog izoliranog mrežnog prostora, što baš i nije praktično niti jednostavno. Stoga se ovakva konfiguracija u većini slučajeva ne koristi. Ipak ako netko baš silno ima tu želju;

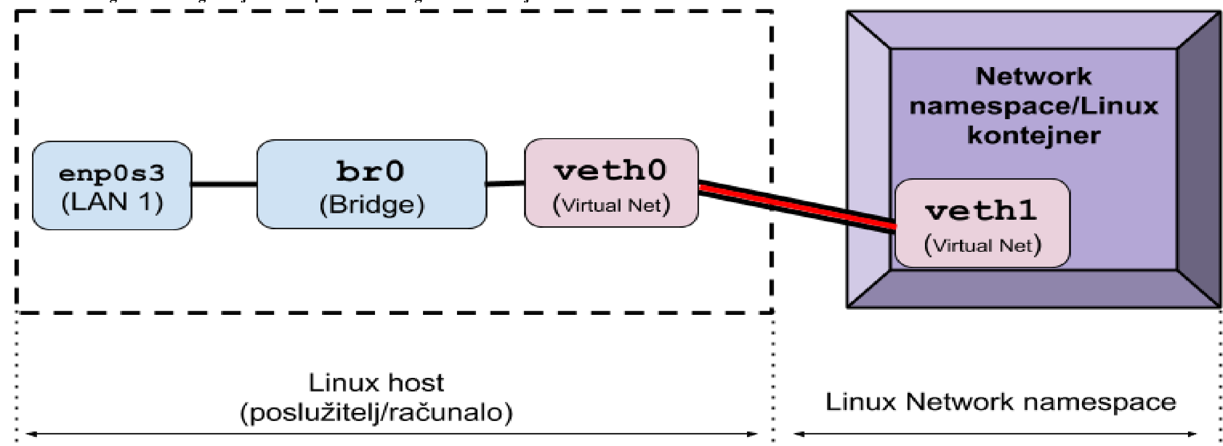
prebacivanje mrežne kartice `enp0s3` u izolirani mrežni prostor imena `PRVI` se može napraviti sa sljedećom naredbom:

```
ip link set enp0s3 netns PRVI
```

- Ono što se u praksi koristi u većini slučajeva kod upotrebe izoliranih mrežnih prostora je da povežemo fizičku mrežnu karticu u standardnom mrežnom prostoru linuxa pomoću ***bridge***(pogl. 20.6.1) sučelja s nekim od virtualnih mrežnih sučelja poput virtualnog *ethernet* sučelja odnosno ***VETH*** vrste sučelja. Naime ***VETH*** sučelje se ponaša kao tunel između izoliranog mrežnog prostora i normalnog mrežnog prostora odnosno normalne mreže Linuxa.

Ovakvu konfiguraciju možemo vidjeti na slici 249.1, a objasnili smo ju u poglavlju: **20.6.4. VETH - posebno mrežno sučelje.**

**Slika 249.1. Pogled na konfiguraciju mreže upotrebom bridge i VETH sučelja**






Pogledajte i poglavlja:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**
**20.6.4. VETH - posebno mrežno sučelje.**

**Izvori informacija:** **`man ip netns`**,**`man ip link`**,**`man 7 network_namespaces`**,**`man 7 namespaces`**.

## 26.8.2. Druga mrežna sučelja (*MACVLAN,IPVLAN,VXLAN,MACVTAP/IPVTAP*)

Osim navedene ***VETH*** vrste posebnih mrežnih sučelja u upotrebi kod izoliranih mrežnih prostora linuxa se u posebnim slučajevima odnosno kod specijaliziranih primjena koriste i sljedeća mrežna sučelja:

***MACVLAN*** - ovo mrežno sučelje se koristi u posebnim slučajevima kada imamo potrebu povezati izolirani mrežni prostor linuxa, direktno bez *bridge* sučelja (Linux ***bridge*** ili ***OVS***) što često nije praktično zbog prednosti odnosno jednostavnosti rada s *bridge* sučeljem. Svejedno u određenim slučajevima primjene možemo koristiti ovo virtualno sučelje koje se poput *bridge* sučelja (na OSI sloju dva) spaja s jedne strane na fizičku mrežnu karticu, a s druge strane se spaja direktno unutar željenog izoliranog mrežnog prostora linuxa. Ovo mrežno sučelje ima notaciju imena poput `macvlan1@enp0s3`, a na vršnom mrežnom sučelju ono praktično postaje pôd sučelje sa svojom MAC adresom. Važno je znati kako postoji pet pôdvrsta *MACVLAN*-ova koje sada nećemo detaljno objašnjavati. Logička shema ovakvog rada je vidljiva na slici **249.2.**
Pogledajte kako konfigurirati *MACVLAN* za mrežni izolirani prostor imena `PRVI` koji je već kreiran:

```
ip link add macvlan1 link enp0s3 type macvlan mode bridge
ip link set macvlan1 netns PRVI
```

***MACVLAN*** sučelje za rad koristi kernel modul imena: `macvlan`.

***IPVLAN*** - primjena ovog mrežnog sučelja je slična kao i kod *MACVLAN* dakle sâmo za specifične potrebe. Naime i ono se vezuje s jedne strane na fizičku mrežnu karticu, a s druge strane unutar izoliranog mrežnog prostora Linuxa odnosno unutar željenog mrežnog izoliranog prostora. Dodatno ovo mrežno sučelje može raditi u *Layer 2 (OSI 2)* načinu rada kao i *MACVLAN (*praktično kao *bridge*) ili u *Layer 3* (OSI 3) pa se tada ponaša kao usmjerivač. U bilo kojem od oba vršna načina rada (*Layer 2* ili *Layer 3*) ovo sučelje prema fizičkoj mrežnoj kartici ima sâmo jednu (virtualnu) MAC adresu, bez obzira koliko *IPVLAN* sučelja prema izoliranih mrežnih prostorima linuxa imali. Logička shema ovakvog rada je vidljiva na slici **249.2.**
Pogledajte kako konfigurirati *IPVLAN* za mrežni izolirani prostor (*network namespace*) imena `PRVI` koji je već kreiran:

```
ip link add ipvlan1 link enp0s3 type ipvlan mode bridge
ip link set ipvlan1 netns PRVI
```

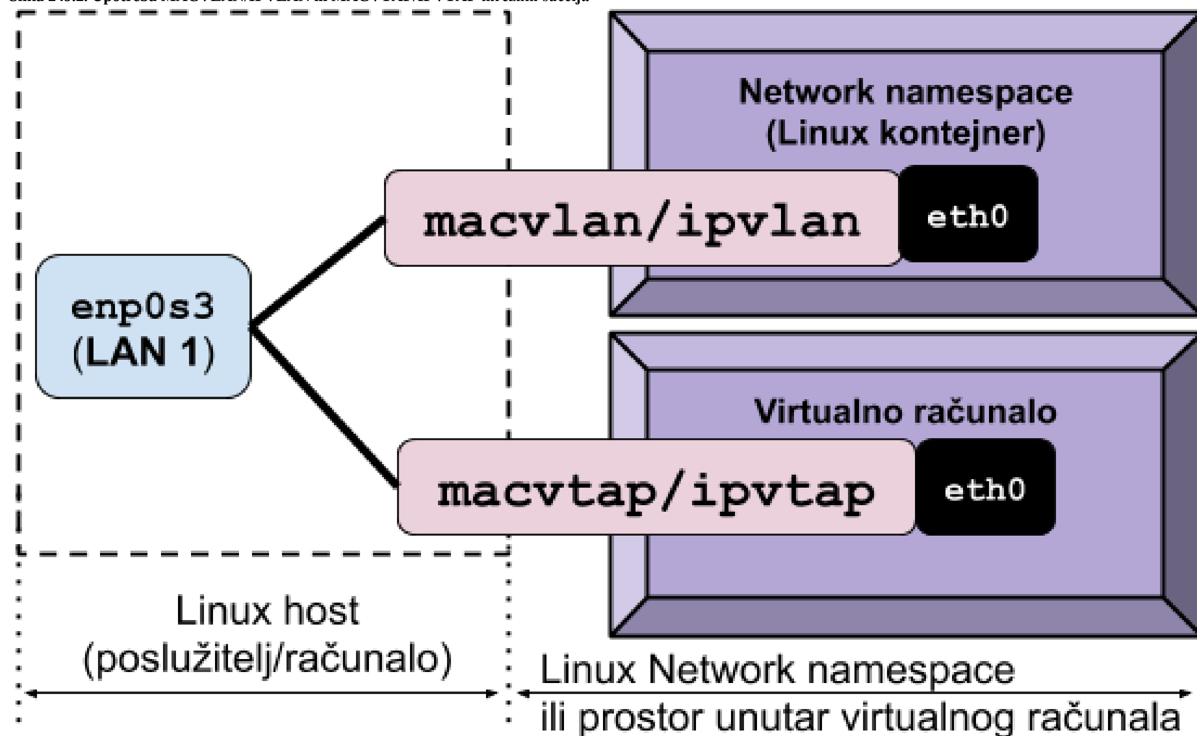
***IPVLAN*** sučelje za rad koristi kernel modul imena: `ipvlan`.

***MACVTAP i IPVTAP*** - primjena ovih mrežnih sučelja je slična kao i kod *gornja dva* sučelja, ali služe za spajanje s virtualnim računalima odnosno s mrežnim sučeljima unutar njih. Kao takvi su zamjena za ***TAP*** mrežno sučelje (***MACVTAP***) ili za ***TUN*** mrežno sučelje (***IPVTAP***). Tijekom njihovog stvaranja, kernel kreira `/dev/tap`**`XY`** sučelje na koje se ***QEMU*** uredno može spojiti i koristiti ga. Osnovna prednost im je u tome što se spajaju direktno na fizičko mrežno sučelje bez potrebe za *bridge* sučeljem. Ova sučelja su novost u najnovijim kernelima pa se očekuje njihova intenzivnija uporaba u budućnosti.
Pogledajmo kako kreirati jedno *MACVTAP* sučelje `macvtap0` povezano na naše fizičko mrežno sučelje `enp0s3`:

```
ip link add link enp0s3 name macvtap0 type macvtap mode bridge
```

***MACVTAP*** sučelje za rad koristi kernel modul imena: `macvtap`, a za rad su mu potrebni kernel moduli: `macvlan` i `tap`.
***IPVTAP*** sučelje za rad koristi kernel modul imena: `ipvtap`, a za rad su mu potrebni kernel moduli: `ipvlan` i `tap`.

**Slika 249.2. Upotreba MACVLAN/IPVLAN ili MACVTAP/IPVTAP mrežnih sučelja**

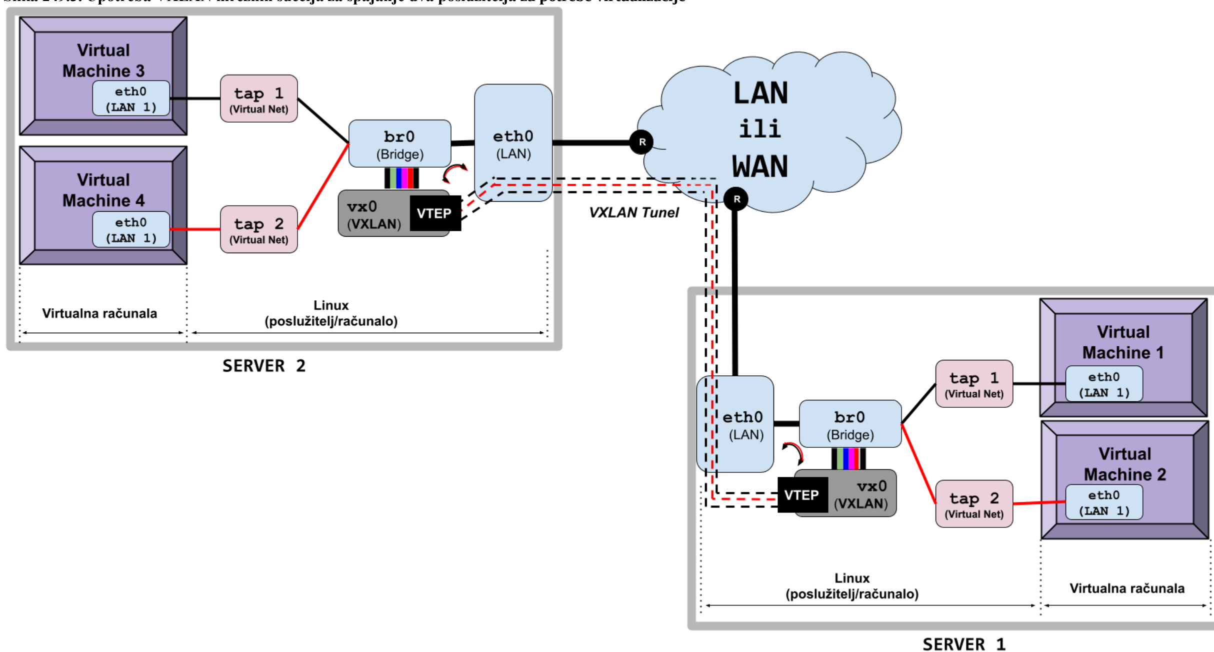


***VxLAN*** - ovo mrežno sučelje nije vezano sâmo uz izolirane mrežne prostore Linuxa već se uglavnom koristi kod komunikacije između ***hipervizora*** u virtualizaciji. Naime ***VXLAN*** (Engl. *Virtual eXtensible Local Area Network*) je protokol za *tuneliranje* osmišljen da riješi problem ograničenog broja ***VLAN*** brojeva (ID-ova) kojih prema standardu (***IEEE 802.1q***) može biti maksimalno 4.096. Dakle u izrazito velikim sustavima za virtualizaciju, u kojima nije bilo dovoljno imati 4.096 VLAN (izoliranih) mreža, morao se smisliti neki drugi način rada. Rad *VXLAN*-a je opisan u standardu RFC 7348.
Naime *VXLAN* sučelje odnosno protokol koji stoji iza njega omogućuje enkapsulaciju virtualiziranih mreža i drugih mrežnih protokola na OSI sloju dva (*Layer 2*) preko standardne IP mreže (OSI sloj 3/*Layer 3*), bez potrebe za upotrebom *VLAN*-ova i posebne konfiguracije preklopnika (*switcha*) kroz koje prolazi mrežni promet odnosno sva komunikacija. To radi tako da se svi mrežni okviri koji dođu do *VXLAN* sučelja obrađuju tako da im se dodaje posebno *VXLAN* zaglavlje te se kao takvi mrežni okviri šalju na mrežu kao *UDP* paketi. Sâmom enkapsulacijom (ugnježđivanjem) i s druge strane dekapsulacijom upravlja komponenta **VXLAN** sustava koja se naziva ***VTEP*** (Engl. *VxLAN Tunnel End Point*). *VXLAN* tehnologija pri tome omogućuje do $\mathbf{2^{24}}$ (16.777.216) virtualnih LAN mreža što je 4.096 puta više od kapaciteta klasičnih *VLAN*-ova. Pošto se ovdje radi o tunelu između dva (ili više) računala, potrebno je konfigurirati oba računala da koriste *VXLAN* tunel. Samim time ovakav tunel ima dvije strane u komunikaciji.
Pogledajte jedan od primjera upotrebe *VXLAN* sučelja pomoću kojeg su povezana dva računala, i u konačnici sva virtualna računala na njima, kako je vidljivo na slici **249.3**. Dakle virtualna računala koja su pokrenuta na prvom poslužitelju (**Server 1** na slici) odnosno koja pokreće njegov ***hipervizor*** za virtualizaciju, se svako preko svog **TAP** (pr. `tap1`) mrežnog sučelja spajaju na Linux *bridge* (`br0`) mrežno sučelje. Potom je zajedničko *bridge* (`br0`) mrežno sučelje spojeno sa fizičkom mrežnom karticom, ali i s ***VXLAN*** mrežnim sučeljem `vx0` koje se naziva i **VTEP** (Engl. *VXLAN tunnel endpoint*). Isti način spajanja je vidljiv i na drugom poslužitelju (**Server 2** na slici). Međusobna komunikacija između virtualnih računala s prvog poslužitelja (**Virtual Machine 1** i **2**) te virtualnih računala s drugog poslužitelja (**Virtual Machine 3** i **4**) se odvija preko *VXLAN* tunela kroz IP mrežu s kojom su spojena oba poslužitelja, bilo da se nalaze u lokalnoj (**LAN**) ili čak udaljenoj mreži (***WAN***).

Osim **VXLAN**ova, koristi se i **NVGRE** (**RFC7637**) te noviji koji ih objedinjuje, naziva **GENEVE** (**RFC 8926**).

***VXLAN*** sučelje za rad koristi kernel modul imena: `vxlan`, a za rad su mu potrebni kernel moduli: `udp_tunnel` i `ip6_udp_tunnel`.

**Slika 249.3. Upotreba VXLAN mrežnih sučelja za spajanje dva poslužitelja za potrebe virtualizacije**






**VXLAN** dodaje **50** bajta svôg dodatnog zaglavlja na svaki mrežni okvir (paket). **VXLAN** tunel sâv mrežni promet koji ulazi u njega, pretvara u **UDP** pakete (promet), da bi ih tijekom primitka na drugoj strani, ponovno konvertirao u izvorni oblik: **TCP** ili **UDP** ili **SCTP** (ovisno o tome u kojem izvornom obliku su bili prilikom ulaska). To naravno unosi određeno kašnjenje i zahtjeva dodatnu procesorsku snagu, čega moramo biti svjesni. VXLAN omogućuje tuneliranje mrežnog prometa s OSI sloja dva (Layer 2) preko mreža koje rade na OSI sloju tri (Layer 3), kako je vidljivo na gornjoj slici.

**OpenStack** ***platforma za virtualizaciju standardno intenzivno koristi VXLAN tunele za međusobno povezivanje poslužitelja. Ako primjerice imate 100 poslužitelja za virtualizaciju, svaki od njih će imati 99 VXLAN tunela prema principu: svaki hipervizor sa svakim hipervizorom*** (*mesh topologija*)***, kako bi sva virtualna računala među sobom mogla komunicirati, i/ili upotrebom posebne multicast grupe za VXLAN komunikaciju [(1213),(1215)] !.***

*__VXLAN__ovi se kao i __VLAN__ovi intenzivno koriste u virtualizaciji, za mrežnu izolaciju virtualnih računala!*

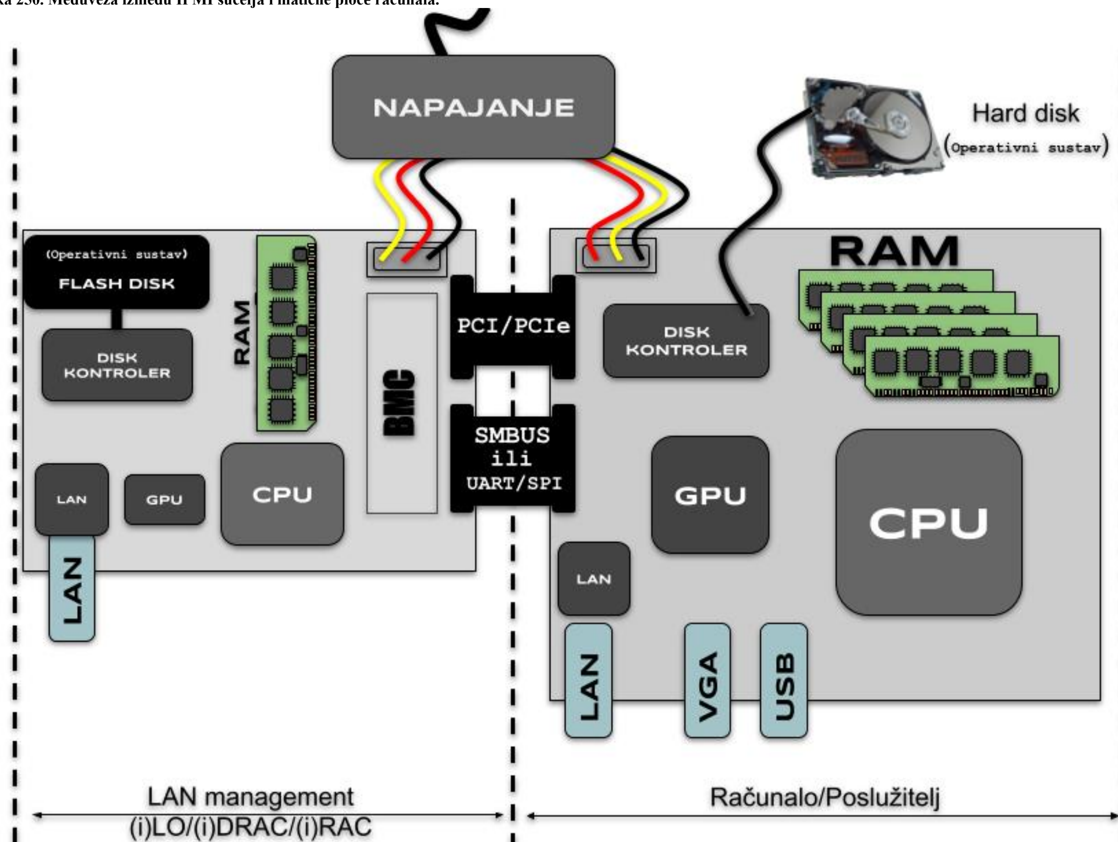
**Izvori informacija:(60),(739),(740),(741),(1052),(1053),(1213),(1214),(1215),** `man ip`, `man ip link`. `man lsns`, `ip netns help`, **RFC 7348.**

### 26.8.3. Intelligent Platform Management Interface (IPMI)

*Intelligent Platform Management Interface* (**IPMI**) koristi se za udaljeni pristup preko mreže do poslužitelja koji ima posebne hardverske komponente na matičnoj ploči (ili je izveden kao posebna kartica). Ovaj sustav čini praktično cijelo minijaturno računalo koje je prije bilo integrirano kao zasebna kartica koja na sebi ima i mrežnu karticu, dok se danas nalazi u samo nekoliko integriranih sklopova (*chipova*) obično na poslužiteljskoj matičnoj ploči kako je vidljivo na slici **250**. Dakle ovom posebnom mrežnom sučelju nakon što ga konfigurirate, možete naravno pristupiti preko mreže, a preko njega imate pristup na cijelo računalo/poslužitelj, čak i kada nemate pristup njegovom operativnom sustavu. Zbog činjenice da zapravo pristupate potpuno neovisnom minijaturnom računalu sa svojim neovisnim operativnim sustavom i mrežnom karticom, a koje je direktno spojeno na matičnu ploču vašeg poslužitelja. Preko njega možete čak i restartati cijelo računalo, resetirati ga i imati potpunu kontrolu nad njim čak od razine *BIOSa*. To znači kako preko ovog mrežnog sučelja možete i instalirati operativni sustav i raditi druge radnje na vašem poslužitelju, kao da se fizički nalazite ispred njega odnosno kao da se nalazite ispred njegovog monitora, miša i tipkovnice. Navedeno sučelje za menadžment odnosno upravljanje, razni proizvođači nazivaju različitim imenima:

- Tvrtka ***HP*** ga zove **iLO** (*Integrated Lights-Out*) ili **ILOM**
- Tvrtka ***Dell*** ga zove **DRAC** (*Dell Remote Access Controller*) odnosno novije inačice **iDRAC** (i=*Integrated*)
- Tvrtka ***Fujitsu*** ga zove **RAC** ili **iRAC** *(Integrated) Remote Management Controller*
- Tvrtka ***Intel*** ga naziva **vPRO**, dok ga neki drugi generalno nazivaju i ***Out-of-band management***.

**Slika 250. Međuveza između IPMI sučelja i matične ploče računala.**




Pogledajmo logičku shemu računala koje ima *LAN menadžment* odnosno **iLO**/**iDRAC**/**iRAC** karticu (slika **250**).
***IPMI*** sučelje se na matičnu ploču uobičajeno spaja preko **PCI** ili **PCIe** sabirnice, a za dohvaćanje raznih statistika (mjerenja) obično se koriste **SMBus** ili **UART/SPI** sabirnice male propusnosti, sve preko specijaliziranog integriranog sklopa (*chipa*) koji se zove *Baseboard Management Controller* (**BMC**). Na **BMC** je spojena i njegova integrirana mrežna kartica na koju se zapravo i spajamo. U konačnici je *BMC sklop,* taj koji šalje određene naredbe matičnoj ploči računala (poslužitelja).
U svakom slučaju, ova posebna mrežna kartica se može konfigurirati iz BIOSa računala ili iz razine operativnog sustava, uz posebne alate koji podržavaju *IPMI* protokol. Ovisno o inačici *IPMI* protokola podržane su (ili nisu) i razne mogućnosti. Za Linux postoji niz alata koji za *RedHat*/*CentOS* Linux dolaze u paketu `ipmitool`. Osim ovog paketa potrebna je:

- Hardverska podrška na matičnoj ploči odnosno *IPMI* (*BMC*) kartica i sustav.
- Imati pripadajuće kernel module za vaš *IPMI* hardver.

Za Linux kernele v.2.6.x potrebni su vam sljedeći kernel moduli (upravljački programi):

- `ipmi_si`
- `ipmi_msghandler`
- `ipmi_devintf`

Ako ipak navedeni kernel moduli (upravljački programi) nisu učitani, možete ih i ručno učitati:

```
modprobe ipmi_msghandler
modprobe ipmi_devintf
modprobe ipmi_si
```

Međutim za Linux kernele v.3.x ili 4.+ (*RedHat/Centos* 7.x/8.x) potrebni su vam sljedeći kernel moduli:

- `ipmi_si`
- `ipmi_ssif`
- `ipmi_msghandler`
- `ipmi_devintf`

Ako navedeni moduli nisu učitani, možete ih i ručno učitati:

```
modprobe ipmi_msghandler
modprobe ipmi_devintf
modprobe ipmi_si
modprobe ipmi_ssif
```

Nakon što su učitani svi potrebni kernel moduli pojaviti će nam se posebni uređaji, poput:

- `/dev/ipmi0` ili
- `/dev/ipmi/0` ili
- `/dev/ipmidev/0`

Zatim možete instalirati i *IPMI* alate, koji pri inicijalizaciji provjeravaju je li sav potreban hardver dostupan:

```
yum install ipmitool
```

Nakon instalacije inicijalizira se i poseban servis imena `ipmievd` koji će se sada i pokretati nakon svakog restarta računala/poslužitelja. Cijela konfiguracija ***IPMI*** sustava se snima u datoteku `/etc/sysconfig/ipmievd`.

Kako bismo bili sigurno da će se ovaj servis pokrenuti i nakon restarta sustava pogledajte (za ***RedHat/CentOS* 6.x**):

```
chkconfig
```

Na ispisu prethodne naredbe pogledajte servis `ipmievd` koji mora imati slijedeće vrijednosti:

```
ipmievd         0:off   1:on    2:off   3:on    4:off   5:off   6:off
```

Za ***RedHat/CentOS*** 7.x/8.x, kako bismo bili sigurni da će se ovaj servis pokrenuti i nakon restarta sustava pogledajte ispis naredbe:

```
systemctl status ipmievd.service
```

Te pogledajte je li ovaj servis u stanju „*disabled*“, poput:

```
● ipmievd.service - Ipmievd Daemon
   Loaded: loaded (/usr/lib/systemd/system/ipmievd.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
```

Za ***RedHat/CentOS*** 7.x/8.x, kako bismo ga pokrenuli, trebamo napraviti sljedeće:

```
systemctl start ipmievd.service
```



Sâmo za provjeru parametara konfiguracije mreže (***iLOM/IPMI***) nije potrebno pokretati ovaj servis.

Sada slijedi osnovna konfiguracija koju ste doduše mogli odraditi i iz *BIOSa* matične ploče jer ponekad je ovo potrebno i mijenjati iz već pokrenutog operativnog sustava.

**Krenimo na konfiguraciju IPMI mrežnog sučelja**

Pogledajmo trenutnu konfiguraciju našeg LAN Management sučelja (***IPMI/ILOM/(i)DRAC***) pomoću naredbe `ipmitool`:

```
ipmitool lan print

Set in Progress         : Set Complete
Auth Type Support       :
Auth Type Enable        : Callback :
                        : User     :
                        : Operator :
                        : Admin    :
IP Address Source       : DHCP
IP Address              : 0.0.0.0
Subnet Mask             : 0.0.0.0
MAC Address             : e4:11:5b:e5:ec:f0
BMC ARP Control         : ARP Responses Disabled, Gratuitous ARP Disabled
Gratituous ARP Intrvl   : 0.0 seconds
Default Gateway IP      : 0.0.0.0
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
Cipher Suite Priv Max   : Not Available
```

Iz ispisa je vidljivo kako su definirane *IP adresa* i maska mreže (*netmask)* te *Gateway*, a drugo nije konfigurirano.

Svaki ***IPMI*** kompatibilni uređaj prijavljuje se na određenom radnom kanalu. U našem slučaju se radi o kanalu broj 2.

U ovom koraku konfigurirati ćemo IP parametre. Prvo ćemo konfigurirati sučelje da koristi fiksno konfiguriranu IP adresu, a ne onu koju mu dodjeljuje *DHCP* poslužitelj jer nam je važno da nam *Lan Management/IPMI/(i)DRAC/(i)LOM* radi i kada je *DHCP* poslužitelj nedostupan. To ćemo napravit na sljedeći način, upotrebom naredbe `ipmitool`:

```
ipmitool lan set 2 ipsrc static
```

Broj 2 označava IPMI kanal na kojem je naš *IPMI* uređaj/kartica na matičnoj ploči.

Sada ćemo konfigurirati i ostale IP parametre:

```
ipmitool lan set 2 ipaddr 192.168.5.125
ipmitool lan set 2 netmask 255.255.255.0
ipmitool lan set 2 defgw ipaddr 192.168.5.1
```

Dakle u prva tri koraka konfigurirali smo:

- *IP* adresu IPMI uređaja (192.168.5.125).
- *Masku mreže* IPMI uređaja (255.255.255.0).
- Podrazumijevani usmjerivač *(Default Gateway)* IPMI uređaja će biti usmjerivač s IP adresom 192.168.5.1.

Još ćemo uključiti mogućnost da *IPMI* uređaj odgovara na ***ARP*** poruke:

```
ipmitool lan set 2 arp respond on
```

Moguće je još kreirati administratorski korisnički račun i lozinku, kao i druge korisničke račune.

Kreirajmo i administratora sustava:

```
ipmitool user set name 2 admin
```

Broj 2 ovdje ne označava kanal nego korisnički broj *(User ID)*. Sada ćemo za novog korisnika s UID br. 2 kreirati lozinku:

```
ipmitool user set password 2
```

Sada ćemo korisniku s ***UID*** (*User ID*) brojem 2, dodijeliti administratorska prava:

```
ipmitool channel setaccess 2 2 link=on ipmi=on callin=on privilege=4
```

Prvi broj 2 u naredbi označava IPMI kanale a drugi broj 2 je ***UID*** broj korisnika.

I sada korisnika `admin` (***UID*** 2) moramo aktivirati.

```
ipmitool user enable 2
```

I konačno pogledajmo našu novu konfiguraciju:

```
ipmitool lan print

Set in Progress         : Set In Progress
Auth Type Support       :
Auth Type Enable        : Callback :
                        : User     :
                        : Operator :
                        : Admin    : PASSWORD
IP Address Source       : Static Address
IP Address              : 192.168.5.125
Subnet Mask             : 255.255.255.0
MAC Address             : e4:11:5b:e5:ec:f0
BMC ARP Control         : ARP Responses Enabled, Gratuitous ARP Disabled
Gratituous ARP Intrvl   : 0.0 seconds
Default Gateway IP      : 192.168.5.1
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
Cipher Suite Priv Max   : Not Available
```

Dakle sve je tu. Spremni smo za upotrebu ovog IPMI mrežnog sučelja, koje bi bilo mudro staviti u zasebnu izoliranu mrežu.

**Druge korisne *IPMI* naredbe**

Pogledajmo kako provjeriti inačicu *firmwarea* na našem *Lan Management/**IPMI/iLOM/iDRAC***:

```
ipmitool mc info
```

Odnosno kako pročitati vrijednosti svih senzora na matičnoj ploči:

```
ipmitool sdr list
```

Ovo je metoda kako provjeriti stanje napajanja poslužitelja:

```
ipmitool power status
```

Naravno, moguće je i restartati ili ugasiti/upaliti poslužitelj (oprez).
Resetiranje poslužitelja (ekvivalent stiskanju *RESET* tipke na kućištu):

```
ipmitool power reset
```

Nasilno gašenje poslužitelja (kao na tipku za gašenje):

```
ipmitool power off
```

Važno je razumjeti, kako poslužitelji koji imaju *Lan Management/IPMI/iLOM/iDRAC* čak i kada su ugašeni, dostupni su preko ovog mrežnog sučelja, pa se samim time poslužitelj može i uključiti ili isključiti (upaliti/ugasiti) pomoću njih.
Paljenje odnosno uključivanje ugašenog poslužitelja (kao na tipku za uključivanje/paljenje) možemo napraviti sa:

```
ipmitool power on
```

Moguće je i privremeno promijeniti *BOOT* uređaj s kojeg će se poslužitelj pokrenuti, kod prvog restarta poslužitelja (nakon drugog restarta se sustav vraća na postavke BIOSa).
Samo za prvo pokretanja sustava s CD/DVD-ROM medija, a kasnije odnosno drugo s diska (normalno), napravimo sljedeće:

```
ipmitool chassis bootdev cdrom
```

**Izvori informacija: (163),(164),(165),(166),** `man ipmitool`.

## 26.8.4. Visoko dostupni sustavi (*High Availability*)

Pod pojmom visoko dostupnih sustava znanih i kao *High Availability* sustavi (**HA**), podrazumijevamo karakteristike sustava čiji je cilj osigurati dogovorenu razinu operativnog učinka, koje se obično odnosi na neprekidni rad sustava, za vremenski period duži od normalnog. Modernizacija te široka primjena i upotreba servisa i usluga na internetu, poglavito usluga u *oblaku*, dovela je do potrebe da ti servisi i usluge budu stalno dostupni, bez ispada ili zastoja u radu. Na primjer, telekomi, bolnice, banke i podatkovni centri zahtijevaju visoku dostupnost svojih sustava za obavljanje rutinskih svakodnevnih aktivnosti. Dostupnost se odnosi na sposobnost ili mogućnost korisnika da dobije uslugu (pr. servis) ili dôbro, da pristupi sustavu ili da ažurira, izmijeni ili snimi postojeći rad. Ako korisnik ne može pristupiti sustavu ili njegovim servisima, s korisničke točke gledišta, oni su nedostupni.

Općenito, termin zastoja odnosno ispada (Engl. *Downtime*) koristi se za označavanje razdoblja kada sustav ili usluga nije dostupna. Postoje tri principa u projektiranju sustava, koji mogu pomoći u postizanju visoke dostupnosti:

- Otklanjanje pojedinačnih točaka kvara. To znači dodavanje ili izgradnju zalihosti (redundancije) u sustav, tako da kvar pojedine komponente ne uzrokuje kvar odnosno ispad cijelog sustava.
- Pouzdano prebacivanje. U redundantnim sustavima, sama točka prebacivanja nastoji postati jedna točka kvara. Pouzdani sustavi moraju osigurati pouzdan mehanizam prebacivanja između komponenti sustava; u pravilu s neispravne na ispravnu.
- Otkrivanje kvarova kako nastaju. Ako se poštuju gornja dva principa, tada korisnik možda nikada neće vidjeti kvar ili ispad pojedine komponente, jer cijeli sustav i dalje mora pouzdano raditi i biti dostupan na korištenje.

Postoji razlika između planiranog i neplaniranog zastoja odnosno ispada sustava.
Planirani zastoji obično su rezultat održavanja koje ometa rad sustava i obično se ne može izbjeći s trenutno instaliranim dizajnom sustava. Događaji planiranog zastoja mogu uključivati zakrpe za softver sustava koje zahtijevaju ponovno pokretanje ili promjene konfiguracije sustava, koje stupaju na snagu tek nakon ponovnog pokretanja. Općenito, zakazani zastoji obično su rezultat nekog logičnog događaja koji je planski pokrenut.

Neplanirani zastoji obično proizlaze iz nekog fizičkog događaja, kao što je kvar hardvera ili softvera ili ekološka anomalija. Primjeri neplaniranih zastoja uključuju nestanke struje, neispravne hardverske komponente poput CPU-a ili RAM memorije, pada sustava povezan s previsokom temperaturom, logički ili fizički prekinute mrežne veze, kršenja sigurnosti ili razne greške u radu aplikacija ili operativnog sustava. Ako se korisnici mogu upozoriti na zakazane (planirane) zastoje ili ispade u radu sustava, to je korisno. Međutim, ako je postojao zahtjev korisnika za istinskom visokom dostupnosti, onda je vrijeme zastoja ili ispada prisutno, bez obzira na to je li ono planirano ili ne. Drugim riječima to vrijeme se broji, bez obzira koji bio uzrok zastoja ili ispada sustava.

Dostupnost se obično izražava kao postotak neprekidnog rada u određenoj godini. Sljedeća skraćena tablica prikazuje vrijeme zastoja/ispada koje će biti dopušteno za određeni postotak dostupnosti, pod pretpostavkom da sustav mora raditi kontinuirano. Ugovori o razini usluge znani kao *Service level agreements* (**SLA**) često se odnose na mjesečne zastoje ili dostupnost sustava, kako bi se izračunala cijena za definiranu razinu usluge.

| Dostupnost sustava % | Godišnje | Mjesečno | Dnevno (24 sata) |
|---|---|---|---|
| **90% ("jedna devetka")** | **36.53 dana** | **73.05 sati** | **2.40 sati** |
| 95% ("jedna i pol devetka") | 18.26 dana | 36.53 sati | 1.20 sati |
| 97% | 10.96 dana | 21.92 sati | 43.20 minuta |
| 98% | 7.31 dana | 14.61 sati | 28.80 minuta |
| **99% ("dvije devetke")** | **3.65 dana** | **7.31 sati** | **14.40 minuta** |
| 99.5% ("dvije i pol devetke") | 1.83 dana | 3.65 sati | 7.20 minuta |
| 99.8% | 17.53 sati | 87.66 minuta | 2.88 minuta |
| **99.9% ("tri devetke")** | **8.77 sati** | **43.83 minuta** | **1.44 minuta** |
| 99.95% ("tri i pol devetke") | 4.38 sati | 21.92 minuta | 43.20 sekundi |
| **99.99% ("četiri devetke")** | **52.60 minuta** | **4.38 minuta** | **8.64 sekundi** |
| 99.995% ("četiri i pol devetke") | 26.30 minuta | 2.19 minuta | 4.32 sekundi |
| **99.999% ("pet devetki")** | **5.26 minuta** | **26.30 sekundi** | **864.00 milisekundi** |
| **99.9999% ("šest devetki")** | **31.56 sekundi** | **2.63 sekundi** | **86.40 milisekundi** |
| **99.99999% ("sedam devetki“)** | **3.16 sekundi** | **262.98 milisekundi** | **8.64 milisekundi** |



Vezano za načine postizanja redundancije, ovisno o komponenti računala, mreže ili protokola, pogledajte poglavlja:
**10.7.2.2. NUMA.**
**12.2.2.1. ECC Memorija.**
**13.11. RAID polja diskova.**
**20.6.5. Redundancija i *Spanning Tree protokol* (STP).**
**20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).**
**21.4. Usmjeravanje (Routing).**
**22. Metode komunikacije.**
**23.4.5. Redundancija na OSI sloju tri (OSI 3) i *VRRP* protokol.**
**25.8.5. DNS protokol.**
**26.8.4.1. Servis conntrack i sinkronizacija stanja mrežnih veza.**

**Izvori informacija: (1090).**

## 26.8.4.1. Servis conntrack i sinkronizacija stanja mrežnih veza

Servis ***conntrackd*** je zadužen za pristup ***netfilter*** sustavu za praćenje veza. Međutim njegova primarna namjena je sinkronizacija stanja mrežnih veza (engl. *sessions*) između nekoliko replika vatrozida ili usmjerivača odnosno uređaja. Dakle ***conntrackd*** se može koristiti za implementaciju visoko dostupnih vatrozida, usmjerivača ili drugih sličnih uređaja koji među sobom moraju sinkronizirati sve sesije ostvarenih veza; primjerice svih ostvarenih TCP i UDP veza na uređaj. *Conntrackd* se također može koristiti kao sakupljač statistika o svim konekcijama na sustav.

On primarno podržava nekoliko načina rada:
- Primarni - pomoćni (engl. *Primary/Master - Backup*).
- Višestruki primarni (engl. *Multi primary/master* ).

Kod većina primjena ***conntrackd*** se kombinira s ***keepalived*** servisom, koji je zadužen za **VRRP** protokol, ali u ovom slučaju i za prebacivanje načina rada ***conntrackd*** servisa između: *MASTER* i *BACKUP*, kako je vidljivo na slici **250.1.** na sljedećoj stranici.



Vezano za **keepalived** servis pogledajte poglavlje:
**23.4.5.1. Keepalived i balansiranje opterećenja (*Load Balancing*).**

Da bi naša konfiguracija uredno radila odlučili smo se instalirati ***keepalived*** i ***conntrackd*** servis na dva poslužitelja, koji će po svojim funkcionalnostima biti **VRRP** poslužitelji s virtualnim IP adresama, te *Load Balanceri* (sve dio ***keepalived*** funkcionalnosti). Za sinkronizaciju stanja veza s primarnog (*MASTER*) na pričuvni (*BACKUP*) uređaj koristi se ***conntrackd*** servis.

Da bi sve to postigli, potrebno je napraviti sljedeće:
1. Potrebno je instalirati ***keepalived*** servis:
```
yum -y install keepalived
```

2. Potrebno je instalirati ***conntrackd*** servis:
```
yum -y install conntrack
```

Nakon instalacije, dobivam ***conntrackd*** servis i njegovu konfiguracijsku datoteku: `/etc/conntrackd/conntrackd.conf` te neke dodatne programe, poput: `conntrack` i `nfct`.

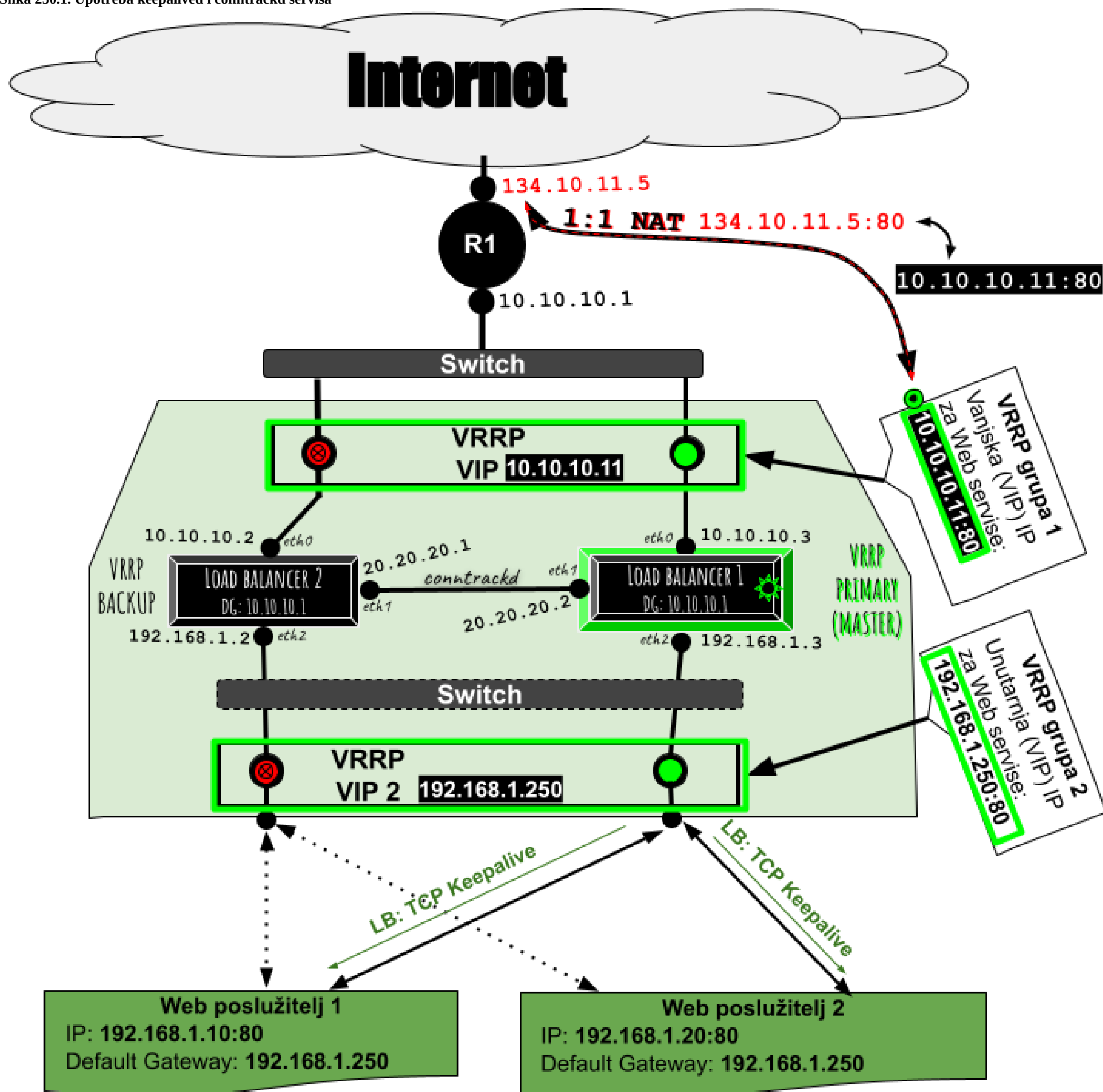
Osim toga, uz *conntrack* paket dobivamo i pomoćnu skriptu koja se može koristiti za servis ***keepalived***, a koju ćemo kopirati:
```
cp /usr/share/doc/conntrack-tools-1.4.4/doc/sync/primary-backup.sh /etc/conntrackd/
```

Naime servis ***keepalived*** će u našem scenariju upotrebe biti zadužen za virtualnu IP adresu te osnovni *load balancing*, dok će ***conntrackd*** servis na primarnom (*PRIMARY/MASTER)* poslužitelju pratiti stanja svih otvorenih konekcija na sustavu te ih slati na pričuvni (*BACKUP*) poslužitelj. Pri tome se *PRIMARY* ***keepalived*** poslužitelju pokreće kao aktivan i on je zadužen za virtualnu IP adresu (***VIP***), a on pokreće i ***conntrackd*** servis kao primarni odnosno aktivni to jest onaj koji snima sve svoje aktivne mrežne konekcije (sesije) i njihova stanja te ih šalje na pričuvni/pomoćni (*BACKUP*) ***conntrackd*** poslužitelj.

To također znači da pričuvni ***keepalived*** poslužitelj nije zadužen za IP adresu, a njegov ***conntrackd*** servis je u stanju slušanja odnosno primanja stanja svih aktivnih mrežnih konekcija s primarnog (aktivnog) poslužitelja. U slučaju ispada primarnog ***keepalived*** servisa to jest poslužitelja, pričuvni postaje aktivan: što se tiče ***keepalived*** i ***conntrackd*** servisa.

Pogledajmo logičku shemu ovakve konfiguracije sustava.

**Slika 250.1. Upotreba keepalived i conntrackd servisa**




Sada ćemo pogledati okvirnu konfiguraciju ***keepalived*** servisa to jest datoteke: `/etc/keepalived/keepalived.conf` usporedno; i to samo inicijalne sekcije: `vrrp_instance`, u konfiguraciji za oba poslužitelja (računala).

Konačno pogledajmo konfiguraciju za prvu virtualnu IP adresu (`10.10.10.11`) i njenu ***VRRP*** grupu broj ***10***:

| Prvo Linux računalo (**R1**) | Drugo Linux računalo (**R2**) | Opis konfiguracije |
|---|---|---|
| `vrrp_instance PRVA {` | `vrrp_instance PRVA {` | •Logički naziv VRRP instance |
| `state MASTER` | `state BACKUP` | •Stanje servisa (MASTER/BACKUP) |
| `interface eth0` | `interface eth0` | •Mrežno sučelje za rad |
| `virtual_router_id 10` | `virtual_router_id 10` | •Broj (identifikator) VRRP **grupe** (**VRID**)* |
| `priority 150` | `priority 100` | •Prioritet* (veći broj je veći *prio*) |
| `advert_int 1` | `advert_int 1` | •Frekvencija slanjaVRRP poruka |
| `authentication {` | `authentication {` | •Autentikacija: |
| `auth_type PASS` | `auth_type PASS` | Vrsta: **PASS** ili **AH** (*IPSEC*) |
| `auth_pass 1234` | `auth_pass 1234` | Lozinka: slijedi nekriptirana lozinka |
| `}` | `}` | |
| `track_interface {` | `track_interface {` | •Prati se stanje **eth0** sučelja |
| `eth0` | `eth0` | |
| `}` | `}` | |
| `virtual_ipaddress {` | `virtual_ipaddress {` | •VRRP virtualna IP za ovu **grupu*** |
| `10.10.10.11/24` | `10.10.10.11/24` | |
| `}` | `}` | |
| `notify_master "/etc/conntrackd/primary-backup.sh primary"` | `notify_master "/etc/conntrackd/primary-backup.sh primary"` | • Što se pokreće ako je MASTER |
| `notify_backup "/etc/conntrackd/primary-backup.sh backup"` | `notify_backup "/etc/conntrackd/primary-backup.sh backup"` | • Što se pokreće ako je BACKUP |
| `notify_fault "/etc/conntrackd/primary-backup.sh fault"` | `notify_fault "/etc/conntrackd/primary-backup.sh fault"` | • Što se pokreće ako je FAULT |
| `}` | `}` | |

Odnosno pogledajmo dio konfiguracije za drugu virtualnu IP adresu (`192.168.1.250`) i njenu ***VRRP*** grupu broj ***20***:

| Prvo Linux računalo (**R1**) | Drugo Linux računalo (**R2**) | Opis konfiguracije |
|---|---|---|
| `vrrp_instance DRUGA {` | `vrrp_instance DRUGA {` | •Logički naziv VRRP instance |
| `state MASTER` | `state BACKUP` | •Inicijalno stanje servisa**: →MASTER ili BACKUP |
| `interface eth2` | `interface eth2` | •Mrežno sučelje za rad |
| `virtual_router_id 20` | `virtual_router_id 20` | •Broj (identifikator) VRRP **grupe** (**VRID**)* |
| `priority 150` | `priority 100` | •Prioritet* (veći broj je veći *prio*) |
| `advert_int 1` | `advert_int 1` | •Frekvencija slanjaVRRP poruka |
| `authentication {` | `authentication {` | •Autentikacija: |
| `auth_type PASS` | `auth_type PASS` | Vrsta: **PASS** ili **AH** (*IPSEC*) |
| `auth_pass 1234` | `auth_pass 1234` | Lozinka: slijedi nekriptirana lozinka |
| `}` | `}` | |
| `track_interface {` | `track_interface {` | •Prati se stanje **eth2** sučelja |
| `eth2` | `eth2` | |
| `}` | `}` | |
| `virtual_ipaddress {` | `virtual_ipaddress {` | •VRRP virtualna IP za ovu **grupu*** |
| `192.168.1.250/24` | `192.168.1.250/24` | |
| `}` | `}` | |
| `notify_master "/etc/conntrackd/primary-backup.sh primary"` | `notify_master "/etc/conntrackd/primary-backup.sh primary"` | • Što se pokreće ako je MASTER |
| `notify_backup "/etc/conntrackd/primary-backup.sh backup"` | `notify_backup "/etc/conntrackd/primary-backup.sh backup"` | • Što se pokreće ako je BACKUP |
| `notify_fault "/etc/conntrackd/primary-backup.sh fault"` | `notify_fault "/etc/conntrackd/primary-backup.sh fault"` | • Što se pokreće ako je FAULT |
| `}` | `}` | |

Dakle najvažniji dio konfiguracije vezan za promjenu stanja rada ***conntrackd*** servisa je skripta: `/etc/conntrackd/primary-backup.sh` koja se ovisno o stanju rada, pokreće s parametrima: `primary`, `backup` ili `fault`.

Pogledajmo i nastavak okvirne konfiguraciju ***keepalived*** servisa to jest datoteke: `/etc/keepalived/keepalived.conf` uz opise (**#**) u istom retku, a vezane za *Load Balancing* to jest raspodjelu mrežnog prometa prema stvarnim web poslužiteljima.

```
virtual_server 10.10.10.11 80 {    # Definira se vanjska IP adresa i port
  delay_loop 5                      # Definicija vremena odgađanja prije prve provjere
  lvs_sched rr                      # Definicija algoritma balansiranja (rr=Round Robin)*
  lvs_method NAT                    # Način rada NAT=translacija adrese:
                                    # -> Vanjska IP: virtual_server
                                    # -> Unutarnje IP: real_server (za sve njih)
  protocol TCP                      # Definira se upotreba TCP protokola
  ha_suspend                        # Ako VirtServ IP nije postavljena, zaustavi aktivnost
                                    # provjere dostupnosti (healtcheck)
  real_server 192.168.1.10 80 {     # Definicija prvog web poslužitelja (IP i PORT)
        TCP_CHECK {             # Definira se upotreba TCP keepalive/healthchecker
            connect_timeout 5     # Maks. vrijeme dozvoljeno za odgovor na provjeru
        }
  }
  real_server 192.168.1.20 80 {     # Definicija drugog web poslužitelja (IP i PORT)
        TCP_CHECK {               # TCP keepalive/healthchecker(provjera dostupnosti)*
            connect_timeout 5      # Maks. vrijeme dozvoljeno za odgovor na provjeru*
        }
  }
}
```

Ovdje definiramo pravila vezana za raspodjelu prometa (*load balancing*), koji će raditi na način da će sav mrežni promet koji dođe na vanjsku IP adresu `10.10.10.11` i port **80**, biti raspodijeljen između stvarnih Web poslužitelja:

- ***Web1***: `192.168.1.10` na portu `80`.
- ***Web2***: `192.168.1.20` na portu `80`.

Osim toga ova dva stvarna web poslužitelja će se provjeravati (`TCP_CHECK`) svakih pet sekundi (s TCP porukama na port 80), te ako nisu dostupni, na njih se (sve dok ne ožive) neće slati promet.

**I konačno dolazimo do *conntrackd* djela konfiguracije.**

Iako datoteka `/etc/conntrackd/conntrackd.conf` ima na stotine opcija i parametara, mi ćemo napraviti najjednostavniju moguću konfiguraciju na oba poslužitelja, uz naše komentare (**#**) uz konfiguraciju:

```
Sync {
        Mode FTFW {                     # Preporučeni način rada
        }
Multicast {
            IPv4_address 225.0.0.50     # Multicast IP adresa koju koristimo za sink.
            Group 3780                  # Grupa (port) koji će servis koristiti
            IPv4_interface 20.20.20.1   # !IP adresa eth1 mrežne kartice poslužitelja!
            Interface eth1              # Mrežna kartica za sinkronizaciju
            SndSocketBuffer 1249280     # Socket međumemorija za slanje
            RcvSocketBuffer 1249280     # Socket međumemorija za slanje
            Checksum on                 # Omogućena provjera konzistencije poruka
        }

General {
        Nice -20                         # Prioritet (nice) s kojim se servis pokreće
        HashSize 32768
        HashLimit 131072
        LogFile on                         # Omogućeno logiranje
        LockFile /var/lock/conntrack.lock # Lock datoteka (pokrenutog servisa)
           UNIX {
                Path /var/run/conntrackd.ctl  # Socket datoteka
                Backlog 20
           }
        NetlinkBufferSize 2097152
        NetlinkBufferSizeMaxGrowth 8388608
        Filter From Userspace {
             Protocol Accept {
                          TCP       # Podrška za TCP protokol
                          SCTP      # Podrška za SCTP protokol
                          DCCP      # Podrška za DCCP protokol
             }

             Address Ignore {
                          IPv4_address 192.168.0.1  # Adrese koje ignoriramo u radu
                          IPv4_address 192.168.1.1  # Adrese koje ignoriramo u radu
             }
        }
}
```

Nakon svega navedenog, potrebno je na oba poslužitelja pokrenuti oba servisa:

```
systemctl start conntrackd
systemctl start keepalived
```

I potom provjerimo njihov status na oba poslužitelja. Prvo za ***conntrackd***:

```
systemctl status conntrackd
● conntrackd.service - connection tracking daemon for debugging and High Availablity
   Loaded: loaded (/usr/lib/systemd/system/conntrackd.service; enabled; vendor
preset: disabled)
   Active: active (running) since Thu 2022-02-10 13:06:02 UTC; 4 weeks 1 days ago
```

Te provjerimo stanje za ***keepalived*** servis:

```
systemctl status keepalived
● keepalived.service - LVS and VRRP High Availability Monitor
   Loaded: loaded (/usr/lib/systemd/system/keepalived.service; enabled; vendor
preset: disabled)
   Active: active (running) since Fri 2022-03-11 13:40:58 UTC; 44min ago
  Process: 17122 ExecStart=/usr/sbin/keepalived -D -p /opt/netzwert/pid/LB-
haxdf_keepalived_slb101.pid (code=exited, status=0/SUCCESS)
```

Prema statusima `loaded` i `active` vidimo da su oba aktivirana. Sada ih možemo trajno aktivirati na oba poslužitelja:

```
systemctl enable conntrackd
systemctl enable keepalived
```

Sve aktivne konekcije možemo vidjeti s naredbom `conntrack` na sljedeći način:

```
conntrack -L
```

Za probu, prvo provjerimo to jest ispišimo sve aktivne mrežne konekcije na primarnom (aktivnom) poslužitelju:

```
conntrack -L
```

A potom se spojite na pričuvni (*backup*) poslužitelj te ponovite ispis:

```
conntrack -L
```

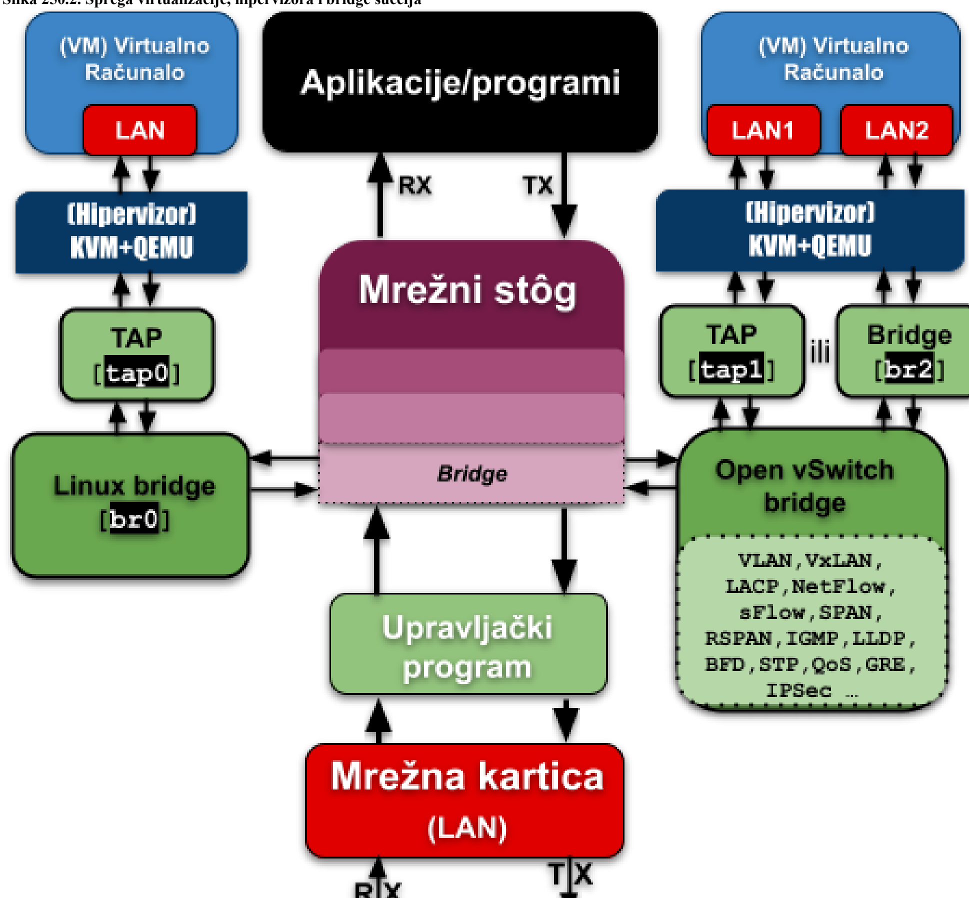
- Ovdje ćete vidjeti da se stanja otvorenih mrežnih konekcija razlikuju u odnosu na primarni. To je normalno, jer pričuvni (*backup*) sustav sve aktivne konekcije čuva u svojoj posebnoj tranzitnoj bazi.
- Tek kada pričuvni (*backup*) poslužitelj odnosno sustav postane primarni (aktivni), osim ***keepalived*** servisa, u to stanje se prebacuje i ***conntrackd*** servis te samim time ažurira stanje svih konekcija na način da sve konekcije koje su do nedavno bile aktivne na starom primarnom (aktivnom) uređaju, sada prelaze na njega i postaju njegove (aktivne) konekcije.

**Izvori informacija: (1191),(1192),** `man conntrackd`,`man conntrackd.conf`,`man conntrack`,`man nfct`.

## 26.8.5. Open vSwitch servis

Naime kao što je vidljivo na slici 250.2., *hipervizor* koji je zadužen za virtualna računala, mrežno sučelje virtualnom računalu može dodijeliti na dva načina. Prvi je preko posebnog ***TAP*** sučelja koje je opet s ostatkom mrežnog podsustava spojeno s ***Linux bridge*** sučeljem. Drugi način je ili veza na **TAP** sučelje spojeno na ***Open vSwitch***, što je standardna i preporučena metoda. Slijedeća metoda, koja se najčešće ne koristi je direktna veza preko ***Open vSwitch*** sustava i to preko njegovog posebnog ***bridge*** sučelja.

**Slika 250.2. Sprega virtualizacije, hipervizora i bridge sučelja**




U svakom slučaju *hipervizorima* je potrebna mogućnost premošćivanja prometa između virtualnih računala i vanjskog svijeta, a za tu namjenu imamo ili *Linix bridge* ili *Open vSwitch* (bridge).

***Open vSwitch*** je usmjeren na implementacije virtualizacije na više poslužitelja, za okruženja za koje prethodna implementacija (*Linux bridge*) često nije prikladna.
Ta okruženja često karakteriziraju vrlo dinamične krajnje točke, održavanje logičkih apstrakcija i integracija s hardverom koji podržava rasterećenje sustava upotrebom specijaliziranih elektronički sklopova, poput primjerice: ***Chipseta***, ***ASIC***, ***NFP*** ili ***DPU*** procesora i/ili upotrebu posebnih softverskih komponenti koje mogu značajno ubrzati prihvaćanje ili obradu mrežnih paketa, kao što je primjerice ***DPDK***.



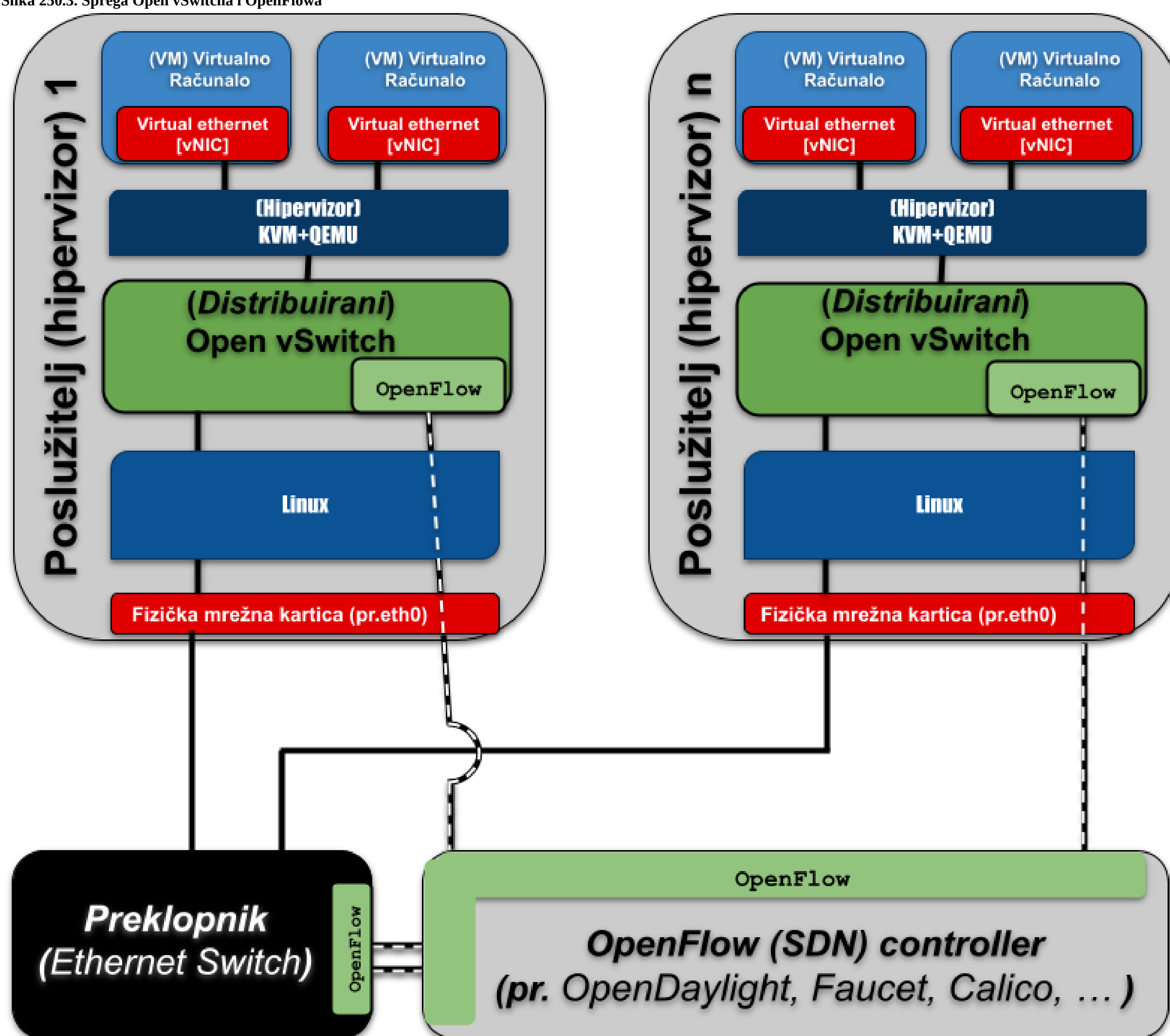
Vezano za navedene specijalizirane elektroničke sklopove te **DPDK**, pogledajte poglavlje:
**26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP + eBPF*).**

**Što je Open vSwitch i što sve podržava?**
***Open vSwitch*** je komponenta produkcijski dokazane kvalitete, s višegodišnjom primjenom u mnogim sustavima za virtualizaciju, poput: ***XEN server***, ***Linux KVM*** (***Proxmox VE***, ***OpenStack***), ***VirtualBox***, ***oVirt***, ….
Potpuna implementacija *Open vSwitch* sustava (koji se naziva i **OVS**), odrađena je s Linux kernelom 3.3. tijekom 2012. godine. *Open vSwitch* je višeslojni virtualni preklopnik licenciran pod licencom Apache 2.0 otvorenog koda.
Većina programskog kôda napisana je u je u programskom jeziku **C** pa je stoga lako prenosiv na druge platforme. Dizajniran je da omogući masovnu automatizaciju mreže kroz programsko proširenje (API), a istovremeno podržava standardna sučelja i protokole upravljanja poput ***OpenFlow***.
Osim toga on podržava *agregaciju* više fizičkih mrežnih sučelja u jedno logičko (pomoću **LACP** protokola) te podržava ***VLAN*** mreže to jest ***802.1Q*** protokol. Nadalje, on podržava i ***LLDP*** (*network discovery*), ***BFD*** (detekciju neispravnih linkova), ***STP*** (*Spanning tree protocol*) i ***RSTP*** te protokole za tuneliranje poput: ***GRE***, ***VxLAN***, ***STT***, ***Geneve*** te ***IPSec*** protokol.
On podržava i ***QoS*** te ***traffic policing*** na razini mrežnog sučelja prema virtualnom računalu te multicast (***snooping*** i ***IGMP*** protokol). I na kraju podržava i protokole poput: ***NetFlow***, ***sFlow*** i ***IPFIX*** za izradu statistika mrežnog prometa te ***SPAN*** i ***RSPAN*** za takozvani ***port mirroring*** odnosno kopiranje svih mrežnih paketa s jednog mrežnog sučelja na drugo.
Dizajniran je tako da ima podršku za distribuirani rad na više fizičkih poslužitelja, slično *VMwareovom vNetwork* distribuiranom *vSwitchu* ili *Ciscovom Nexusu 1000V*.

Naime *OpenvSwitch* radi i kao takozvani ***OpenFlow*** preklopnik prilagođen virtualiziranim okruženjima kao što je Linux KVM. Dakle on se koristi kao softver koji se ponaša kao višeslojni preklopnik (engl. switch), a služi za međusobno povezivanje virtualnih računala na istom fizičkom poslužitelju (*hipervizoru*) ili između različitih fizičkih računala (*hipervizora*).

Upravo ova njegova sposobnost komunikacije upotrebom ***OpenFlow*** protokola, preko kojeg s njim može komunicirati udaljeni kontroler i zadavati mu naredbe za (re)konfiguraciju internog *Open vSwitch* preklopnika, čini ga nevjerojatno dobrim za distribuirani rad. Naime, kako je i vidljivo na donjoj slici, s *Open vSwitch* komponentama na pojedinim fizičkim poslužiteljima odnosno *hipervizorima*, mogu upravljati udaljeni ***OpenFlow*** kontroleri znani kao ***SDN*** (engl. *software defined network*) kontroleri. Uloga *SDN* kontrolera je konfiguracija pojedinih *Open vSwitch* (virtualnih) preklopnika, kao da se radi o klasičnim naprednim višeslojnim preklopnicima odnosno tzv. *multilayer switchevima*.

**Slika 250.3. Sprega Open vSwitcha i OpenFlowa**




Tako je moguće s centralnog mjesta paralelno upravljati s mrežnim sustavom svih *hipervizora*, na svim poslužiteljima, naravno preko *Open vSwitch* sustava. Isto tako ***SDN*** kontroler, može (ako to fizički preklopnik podržava), upravljati i s fizičkim preklopnicima (*switchevima*) i svim njihovim funkcijama, što se vrlo često koristi u podatkovnim centrima. Za takav rad su potrebni mrežni preklopnici koji podržavaju upravljanje pomoću ***OpenFlow*** protokola.

***Open vSwitch*** se sastoji od nekoliko osnovnih elemenata:

- Linux kernel modula odgovornih za njegov rad (primjerice centralnog modula `openvswitch` i drugih.).
- ***ovs-vswitchd*** – Linux servis u kojem je implementirana osnovna funkcionalnost softverskog preklopnika.
- ***ovsdb-server*** – poslužitelj interne baze podataka u koje se sprema konfiguracija OVS-a.
- ***ovs-dpctl*** – alat za konfiguriranje i rad s kernel modulima.
- ***ovs-vsctl*** – alat za konfiguriranje Open vSwitch sustava (preko servisa ***ovs-vswitchd***).
- ***ovs-appctl*** – program koji se koristi za slanje naredbi OVS servisima.
- ***ovs-ofctl*** – program za upravljanje i kontrolu ***OpenFlow*** sustavom.
- ***ovs-pki*** – program za stvaranje i upravljanje infrastrukturom javnih ključeva.

Što se optimizacija u radu *Open vSwitcha* i *hipervizora* tiče, pogledajte optimizacije koje smo spomenuli u poglavljima o virtualizaciji i o ***IOMMU***, a koja im drastično mogu ubrzati rad.



Vezano za virtualizaciju (i optimizacije) pogledajte poglavlja:
**10.7.1.4. Ulazno-izlazni memorijski kontroler (IOMMU).**
**27.1.2. Rad s virtualizacijom (KVM+QEMU).**
**27.1.3. Optimizacije.**

**Open vSwitch, KVM i QEMU konfiguracija**
Krećemo s instalacijom QEMU/KVM programa (*hipervizora*):
```
yum install -y qemu-img qemu-kvm-common qemu-kvm-core
```

Potom ćemo instalirati ***Open vSwitch*** servise (za ***RedHat/CentOS 8.x***), ali prvo dodajmo novi repozitorij:
```
yum install -y https://repos.fedorapeople.org/repos/openstack/openstack-yoga/rdo-release-yoga-1.el8.noarch.rpm
```
← Navedeni repozitorij je prilagođen ***OpenStack*** sustavu za virtualizaciju, a ovisi za koju inačicu ***OpenStack*** sustava želimo instalaciju. Sve inačice su vidljive ovdje: https://repos.fedorapeople.org/repos/openstack/.
Mi smo odabrali trenutno najnoviju (takozvanu ***Yoga*** inačicu). Naravno, ako to želite, i sami možete kompilirati *Open vSwitch* iz njegovog izvornog programskog kôda, dostupnog na: https://www.openvswitch.org/download/
Mi ipak krećemo s instalacijom s gornjeg repozitorija prilagođenog za *OpenStack*:
```
yum -y install openvswitch
```

Sada je potrebno kreirati trajnu konfiguraciju mrežnih sučelja koja ćemo koristiti (fizičko `eno1`), uređivanjem takozvane ***ifcfg*** datoteke: `/etc/sysconfig/network-scripts/ifcfg-eno1`, koja treba sadržavati:
```
DEVICE=eno1
ONBOOT=yes
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
```
Obratite pažnju na: `DEVICETYPE=ovs` i na: `TYPE=OVSPort`. To znači da je ovo sučelje tzv. *OVS port* unutar *OVS* komponente. Sada ćemo kreirati novu datoteku: `/etc/sysconfig/network-scripts/ifcfg-br-ex` koja će definirati OVS Bridge sučelje `br-ex`, Ova konfiguracijska datoteka treba sadržavati sljedeće:
```
DEVICE=br-ex
BOOTPROTO=none
ONBOOT=yes
TYPE=OVSBridge
DEVICETYPE=ovs
USERCTL=yes
PEERDNS=yes
IPV6INIT=no
IPADDR=192.168.1.10
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=192.168.1.1
DNS2=8.8.8.8
```
Dakle ovo sučelje je *OVS bridge, zbog:* `TYPE=OVSBridge` i `DEVICETYPE=ovs`.
Omogućimo pokretanje OVS servisa automatski te ga i pokrenimo:
```
systemctl enable openvswitch
systemctl start openvswitch
```

## *Malo teorije o vrstama mrežnih sučelja u Open vSwitchu*
S obzirom da je *Open vSwitch* integriran s **Red Hat** kompatibilnim distribucijama Linuxa, za konfiguraciju posebnih *Open vSwitch* mrežnih sučelja unutar **ifcfg** datoteka (pr. `/etc/sysconfig/network-scripts/ifcfg-eno1`) dostupno je nekoliko opcija. S ovim opcijama ćemo se upravo upoznati.



Vezano za konfiguraciju mrežnih sučelja i takozvanim ***ifcfg*** datotekama, pogledajte poglavlje:
**25.6.5.1.2. Trajna (permanentna) ručna konfiguracija mreže.**

Slijede opcije i parametri unutar takozvanih ***ifcfg*** datoteka koji se mogu koristiti, koristimo li *Open vSwitch*.
Prva opcija koja mora biti postavljena je:
```
DEVICETYPE=ovs
```

Definicija (`DEVICETYPE=ovs`) naznačava sustavu, da se radi o *Open vSwitch* sustavu. Nakon toga sljedeće opcije mogu biti:

- `TYPE=OVSBridge` - za *OVS* bridge uređaje, dok se ime mrežnog sučelja definira kao pr. (`DEVICE=br-ex`).
- `TYPE=OVSPort` - za *OVS* port (kao port na preklopniku), koji se mora vezati sa sučeljem (*bridge*, fizičko sučelje, …).
- `TYPE=OVSIntPort` - za *OVS* interni port koji se mora vezati s nekim sučeljem (*bridge*, fizičko sučelje, …).
- `TYPE=OVSBond` - za *OVS **Bond*** sučelje (***LACP***) koje se mora vezati s nekim fizičkim sučeljima (pr. `eth0`, `eth1`, …).
- `TYPE=OVSPatchPort` - za *OVS **patch*** sučelje koje se mora vezati s nekim drugim ***patch*** sučeljem.
- `TYPE=OVSTunnel` - za *OVS* tunel sučelje (pr. za ***GRE ili VxLAN*** tunele).
- Ako nemamo `TYPE=OVSBridge` tada se može koristiti i `OVS_BRIDGE=IME sučelja`, gdje definiramo *bridge* sučelje (pr. `OVS_BRIDGE=br-ex`).

Moguća je i upotreba dodatnih OVS opcija, navođenjem: `OVS_OPTIONS=` nakon koje slijede opcije, poput primjerice: `tag=100` za pripadnost VLANu 100. Nadalje, ako koristimo ***DPDK*** varijantu *Open vSwitcha*, moguća je upotreba i dodatnih sučelja: `OVSDPDKBond` (***LACP*** bond), `OVSDPDKPort` (za fizičko mrežno sučelje) i `OVSDPDKVhostUserPort` (za vezu s *bridge* sučeljem). Pogledajmo nekoliko primjera osnovnih konfiguracija:

| **[ifcfg-br-ex]** **OVS Bridge** | **[ifcfg-eno1]** **OVS port** **(prema br-ex)** | **[ifcfg-bond0]** **OVS bond s mrežnim sučeljima (eno1 i eno2) te s *bridgem* na br-ex.** |
|---|---|---|
| DEVICE=br-ex<br>DEVICETYPE=ovs<br>TYPE=OVSBridge<br>... | DEVICE=eno1<br>DEVICETYPE=ovs<br>TYPE=OVSPort<br>OVS_BRIDGE=br-ex<br>... | DEVICE=bond0<br>DEVICETYPE=ovs<br>TYPE=OVSBond<br>OVS_BRIDGE=br-ex<br>BOND_IFACES="eno1 eno2"<br>OVS_OPTIONS="bond_mode=balance-tcp<br>lacp=active" |

Vratimo se na naš servis te provjerimo radi li *OVS*:

```
ovs-vsctl show
8d0316be-df79-433a-b76b-e2e1706fac46
    ovs_version: "2.15.6"
```

Vidimo da radi te da je inačice **2.15.6.**
Sada je potrebno restartati mreži servis (ili cijelo računalo); primjerice sa (za ***RedHat/Centos 7.x***):

```
systemctl restart network
```

Potom kreirajmo *OVS bridge* sučelje s naredbom `ovs-vsctl` unutar *OVS* sustava (preko ***`ovs-vswitchd`*** servisa):

```
ovs-vsctl add-br br-ex
ovs-vsctl add-port br-ex eno1
```

Pogledajmo što smo dobili:

```
ovs-vsctl show
8d0316be-df79-433a-b76b-e2e1706fac46
    Bridge br-ex
        Port eno1
            Interface eno1
        Port br-ex
            Interface br-ex
                type: internal
    ovs_version: "2.15.6"
```

Dakle unutar OVS-a imamo *bridge* sučelje (`br-ex`) na koje je spojeno fizičko mrežno sučelje (mrežna kartica) `eno1`.
Važno je razumjeti da se *OVS bridge* sučelja vide i iz Linuxa, primjerice s naredbama `ip` i `ifconfig`.
*Open vSwitch* podržava i ***VLAN*** mreže (802.1Q). Pogledajmo konfiguraciju ***TAP*** sučelja koja pripadaju različitim *VLANovima.*

```
ovs-vsctl add-br br0
ovs-vsctl add-port br0 eth0
ovs-vsctl add-port br0 tap0 tag=100
ovs-vsctl add-port br0 tap1 tag=200
```

Vidimo kreiranje OVS *bridge* sučelja (**`br0`**) i dodavanja mrežne kartice (**`eth0`**) u njega, te dodavanje *TAP* sučelja u isti *bridge*. Međutim *TAP* sučelje (**`tap0`**) je ubačeno u *VLAN* 100, a *TAP* sučelje (**`tap1`**) u *VLAN* 200. Kasnije *TAP* sučelja dodajemo virtualnim mrežnim karticama unutar pojedinih virtualnih računala. Ključna riječ **`tag=`** je za pripadnost *VLAN* mrežama.



Kao što ste vidjeli *Open vSwitch* osim što podržava ***bridge*** i ***bond*** sučelja, podržava i posebna interna sučelja, ali i sučelja koja se mogu koristiti samo i isključivo, ako je *Open vSwitch* kompiliran i konfiguriran da koristi **DPDK**.

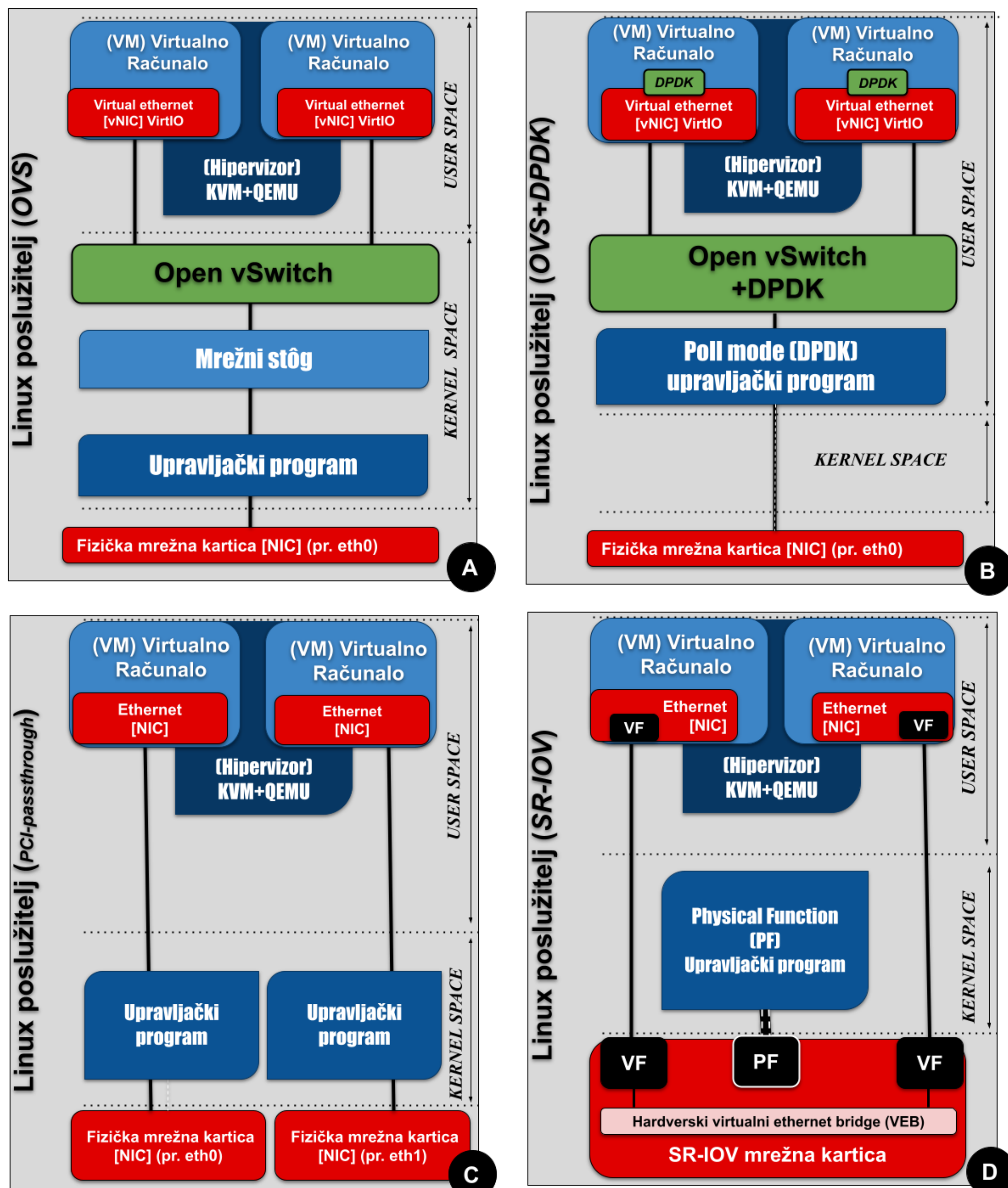
**Izvori informacija: (1326),(1327),(1328),(1329),(1330),(1331),(1332),(1333),(1334), `man ovs-vsctl`.**

## 26.8.6. Usporedba mrežnih tehnologija hipervizora za virtualizaciju

Pošto smo se do sada upoznali s nekoliko načina na koji se ***hipervizor*** s jedne strane spaja na virtualno mrežno sučelje unutar virtualnog računala (***vNIC***), a s druge na fizičku mrežnu karticu (*NIC*), sada ćemo ih detaljnije usporediti.

Pogledajte usporedne logičke sheme raznih tehnologija povezivanja virtualnih računala s fizičkom mrežom, na slici dolje.

**Slika 250.4. Usporedba virtualnih preklopnika (Virtual Switches) u virtualizaciji.**



Pogledajmo sliku 250.4., te četiri najčešća modela povezivanja s virtualnim preklopnikom (tzv. ***Virtual switch***), ili bez njega:

- Klasični model (A) u kojem, ako unutar virtualnih računala koristimo *paravirtualizirano* (***VirtIO***) mrežno sučelje, ono zaobilazi *hipervizor* (to jest ***QEMU*** emulator) i mrežno komunicira kroz ***Open vSwitch*** (**OVS**). Ovo je i mrežno najsporija komunikacija od navedenih, zbog više logičkih slojeva mreže i mehanizama kroz koje mora proći svaki mrežni paket. Ova metoda je slična još jednostavnijoj bez ***Open vSwitcha***, kod koje se virtualno računalo direktno spaja na ***TAP*** a potom na Linux ***Bridge*** sučelje. Obije metode: ***OVS*** i Linux ***TAP+Bridge*** imaju slične performanse: propusnost, latenciju i zauzeće CPUa. Dakle ovdje su performanse zadovoljavajuće za većinu primjena.

- Model (B) u kojem, ako unutar virtualnih računala ovaj puta koristimo ***DPDK*** *paravirtualizirano* (***VirtIO***) mrežno sučelje, i ono zaobilazi *hipervizor* te prolazi kroz ***Open vSwitch*** (**OVS**). Međutim ovdje se zbog ***DPDK*** zaobilazi i cijeli mrežni stôg Linuxa. Zatim se pomoću posebnog upravljačkog programa (samo za odabrane) mrežne kartice koji podržava tzv. ***Pooling*** način rada, ekstremno brzo mogu dohvaćati mrežni paketi.



Vezano za **DPDK**, pogledajte poglavlje:
**26.7.3.3. Najnovija metoda dohvaćanja i obrade mrežnih paketa (*XDP* + *eBPF*).**

- Model (C) je potpuno drugačiji jer ne koristi niti *virtualizirana* (*emulirana*) niti *paravirtualizirana* mrežna sučelja. Naime on se zove PCI propusni (engl. *PCI passthrough)* način rada u kojem propuštamo fizičku mrežnu karticu (ili više njih), kroz PCI sabirnicu, sve do virtualnog računala. Ovo je ekstremno brz način komunikacije, ali ne koristi ***Open vSwitch*** (**OVS**) kao virtualni preklopnik, a također zaobilazi *hipervizor*. To sve zajedno drastično ubrzava propusnost, te smanjuje latenciju i zauzeće procesora, jer daje izravan pristup hardveru (mrežnoj kartici). Važan zahtjev da bi to uopće radilo je posebna fizička mrežna kartica koja podržava ***DPDK*** način rada.

- Model (D) je sličan modelu (C) jer isto zaobilazi i *hipervizor* i virtualni preklopnik to jest ***Open vSwitch*** (**OVS**). On zahtjeva još kompleksniju mrežnu karticu koja podržava takozvani ***Single-Root I/O Virtualization*** (**SR-IOV**). ***SR-IOV*** radi slično kao PCI propusni način rada jer propušta PCI uređaje do virtualnog računala. Međutim ovakve mrežne kartice omogućuju više takozvanih ***Virtual Network Functions*** (**VNF**) uređaja da dijele isto fizičko mrežno sučelje odnosno jedan PCIe uređaj. Hardverski emulirani PCIe uređaj naziva se ***Virtual Function*** (***VF***), a fizički PCIe uređaj naziva se ***Physical Function*** (**PF**). Fizičke funkcije (**PF**) imaju pune PCIe značajke. Virtualne funkcije (**VF**) imaju jednostavne PCIe funkcije koje obrađuju samo ulazno/izlazne operacije koristeći PCI tehnologiju prolaza (*PCI-passthrough*). Pri tome ***VNF***-ovi imaju izravan pristup ***VF***-ovima.

Pogledajmo i usporednu tablicu važnijih značajki svakog od navedenih modela:

| ***Značajke*** | **Model A - OVS** | Model B – **OVS+DPDK** | Model C – ***PCI-passthrough*** | Model D – ***SR-IOV*** |
|---|---|---|---|---|
| ***Sučelje*** | Preko **OVS**, kroz mrežni stôg Linuxa | Preko **OVS,** upotrebom **DPDK,** zaobilazeći mrežni stôg Linuxa | Bez **OVS** zaobilazeći mrežni stôg Linuxa, direktno na **PCI** sučelje. | Bez **OVS** zaobilazeći mrežni stôg Linuxa, direktno na **PCI** sučelje. |
| ***Propusnost VM-VM (lokalno)*** | Veća | Velika | Vrlo velika (*ovisno o izvedbi mrežne kartice*) | Vrlo velika (*ovisno o izvedbi mrežne kartice*) |
| ***Propusnost Prema van*** | Veća | Velika | Ekstremno velika *(Hardverska brzina)* | Ekstremno velika *(Hardverska brzina)* |
| ***Jednostavnost migracije virtualnih računala na drugi poslužitelj*** | Vrlo jednostavno | Manje jednostavna, ali moguća | Kompliciranije zbog vezanja na PCI sabirnicu/uređaj | Kompliciranije zbog vezanja na PCI sabirnicu/uređaj |
| ***Sigurnost*** | Velika *(Na razini **vSwitcha**)* | Velika *(Na razini **vSwitcha**)* | Mala zbog nedostatka virtualnog preklopnika (**vSwitcha**) | Mala zbog nedostatka virtualnog preklopnika (**vSwitcha**) |

**Izvori informacija: (1379),(1380),(1381),(1382),(1383),(1384),(1385),(1386).**

## 26.8.7. Virtual Private Network (VPN)

*Virtual Private Network* (*VPN*) odnosno virtualna privatna mreža je pojam ili tehnologija koja omogućava sigurno povezivanje na udaljenu mrežu, preko interneta. Odnosno VPN je moguće učiniti sigurnim za korištenje povrh nesigurnog komunikacijskog medija (kao što je javni internet) odabirom protokola za tuneliranje koji implementira potrebne sigurnosne značajke za jamčenje povjerljivosti i integriteta podataka. Naime, VPN uspostavlja šifriranu vezu odnosno komunikacijski kanal između vašeg računala i VPN poslužitelja, koji je uobičajeno dostupan na internetu.
Cilj VPN-a je omogućiti osobnim računalima, poslužiteljima kao i drugoj mrežnoj opremi komunikaciju s odredišnom mrežom preko (nesigurne) mreže, kao da su dio iste odredišne mreže, na načine koji čine prelazak kroz posredničku mrežu (uobičajeno nesigurni internet) transparentnim za komunikaciju. Transparentnim znači da kroz takav komunikacijski kanal mogu prolaziti čak i mnogi mrežni protokoli nižih razina, što naravno ovisi o vrsti VPN komunikacijskog kanala i protokola koji se koristi.

Kako radi VPN ?

- **Šifriranje** – VPN šifrira podatke koji putuju između vašeg uređaja i VPN poslužitelja. To osigurava da su podaci zaštićeni od prisluškivanja.
- **Tuneliranje** – VPN stvara "*tunel*" kroz koji podaci prolaze. Ovaj tunel je šifriranjem zaštićen od neovlaštenog pristupa.
- **Maskiranje** (*Masquerading*) – kada se povežete na VPN poslužitelj, vaša stvarna IP adresa je skrivena, a vaša mrežna aktivnost se prikazuje kao da dolazi s IP adrese VPN poslužitelja.

Gdje se sve koristi VPN ?

- Osobna upotreba – za zaštitu privatnosti i pristup geo-blokiranom sadržaju.
- Poslovna primjena – za sigurno povezivanje zaposlenika s tvrtkom odnosno njenom mrežom, pogotovo ako rade na daljinu; primjerice od kuće.
- Putovanja – za pristup uslugama i sadržaju iz matične zemlje dok ste u inozemstvu.
- Školstvo i istraživanje – za pristup obrazovnim resursima i istraživačkim bazama podataka koje su ograničene na određene regije.

Koje kategorije VPN veza postoje ?

- **Site-to-Site VPN** – za povezuje cijele mreže, obično se koristi u poslovnim okruženjima. Dakle povezuje primjerice dvije udaljene lokacije iste tvrtke u jednu zajedničku mrežu.
- **Remote Access VPN/Client-to-Site VPN** – omogućava pojedinačnim korisnicima da se povežu s privatnom mrežom s udaljenih lokacija. Primjerice za udaljeno spajanje na mrežu tvrtke.

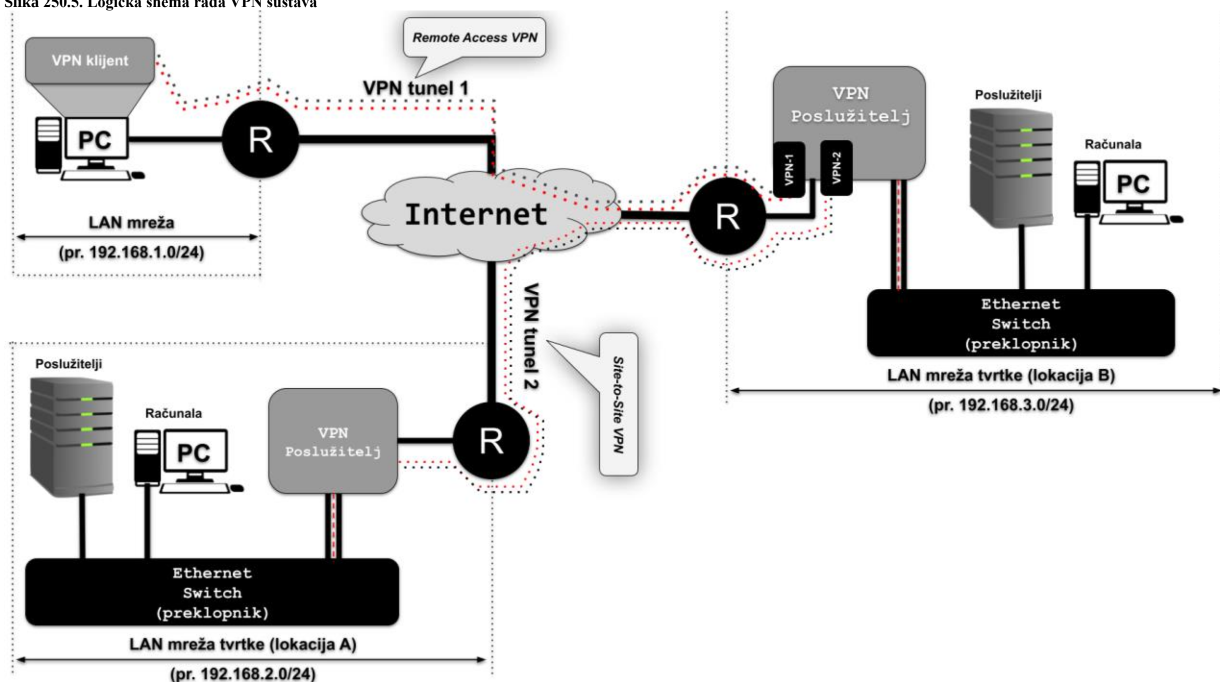
Pogledajmo i tablicu s najčešćim protokolima koji se koriste za VPN:

| Protokol | Opis | Prednosti | Mane |
|---|---|---|---|
| **PPTP** | *Point-to-Point Tunneling Protocol*; stariji protokol s osnovnim šifriranjem i brzim povezivanjem, ali manje sigurnim. | *- Vrlo brzo uspostavljanje veze.<br>- Jednostavan za konfiguraciju.<br>- Široko podržan.* | *- Zastarjela i ranjiva enkripcija.<br>- Niska razina sigurnosti.<br>- Prikladan samo za osnovne sigurnosne potrebe.* |
| **L2TP** | *Layer 2 Tunneling Protocol;* stariji protokol. | *- Jednostavan za implementaciju.<br>- Može se koristiti za tuneliranje različitih protokola.* | *- Nema ugrađenu enkripciju, stoga nije siguran.<br>- Prikladan samo za specifične primjene gdje enkripcija nije potrebna.<br>- Podložan napadima ako se koristi bez enkripcije.* |
| **L2TP + IPSEC** | *Layer 2 Tunneling Protocol* (L2TP) koristi se zajedno s IPSec za enkripciju podataka. Ova kombinacija pruža sigurniji prijenos podataka u odnosu na čisti L2TP. | *- Relativno jednostavan za implementaciju.<br>- Široko podržan na raznim platformama.<br>- Sigurniji od PPTP protokola.* | *- Sporiji od OpenVPN-a i IKEv2/IPSec-a.<br>- Može biti blokiran od strane NAT firewalla.<br>- Može imati problema s performansama zbog dvostrukog enkapsuliranja.* |
| **OpenVPN** | Open-source protokol poznat po svojoj sigurnosti i fleksibilnosti; koristi SSL/TLS za enkripciju. | *- Visoka sigurnost.<br>- Fleksibilan i prilagodljiv.<br>- Podrška za mnoge enkripcijske algoritme.<br>- Open-source.* | *- Može biti složen za konfiguraciju.<br>- Potrebna dodatna softverska podrška na nekim platformama.<br>- Može biti sporiji na nekim mrežama* |

| Protokol | Opis | Prednosti | Mane |
|---|---|---|---|
| **SSTP** | *Secure Socket Tunneling Protocol*; koristi SSL/TLS za zaštitu podataka i često se koristi na Windows platformama. | *- Dobro prolazi kroz vatrozide.<br>- Integriran u Windows operativne sustave.<br>- Visoka sigurnost zahvaljujući SSL/TLS enkripciji.* | *- Ograničen na Windows platforme.<br>- Zatvoren izvor, što smanjuje transparentnost.<br>- Može biti sporiji od OpenVPN-a na nekim mrežama.* |
| **IKEv2 + IPSEC** | *Internet Key Exchange* version 2 u kombinaciji s IPsec-om; pruža dobru sigurnost i stabilnost, posebno na mobilnim uređajima. | *- Brzo uspostavljanje veze.<br>- Stabilan na mobilnim uređajima (brzo se povezuje nakon prekida).<br>- Podrška za visoku razinu enkripcije.* | *- Može biti blokiran od strane vatrozida.<br>- Podrška za IKEv2 može biti ograničena na nekim uređajima.<br>- Relativno složena konfiguracija.* |
| **WireGuard** | Novi open-source VPN protokol, dizajniran za visoku sigurnost i brzinu. Koristi modernu kriptografiju i jednostavniji programski kôd od OpenVPN-a ili IKEv2. | *- Vrlo brzo uspostavljanje veze.<br>- Visoka sigurnost s modernim kriptografskim algoritmima.<br>- Manje složen za implementaciju.<br>- Open-source i transparentan.* | *- Još uvijek relativno nova tehnologija.<br>- Podrška nije toliko široko rasprostranjena kao kod drugih protokola.<br>- Ne nudi podršku za dinamičke IP adrese.* |

Pogledajmo i logičku shemu rada VPN sustava.

**Slika 250.5. Logička shema rada VPN sustava**




Na slici su vidljiva dva logička primjera upotrebe VPN-a:

- ***Remote Access VPN*** – u ovom primjeru je vidljivo kako se računalo (pr. kućno računalo [PC]) pomoću svog VPN klijenta može spojiti na udaljeni VPN poslužitelj, preko interneta. Tada se stvara poseban kriptirani (šifrirani) komunikacijski kanal odnosno *tunel*. Kroz taj *tunel* naše računalo na mreži 192.168.1.0/24, s primjerice IP adresom 192.168.1.10, može pristupati mreži tvrtke (lokacija **B**), čija mreža je u opsegu IP adresa 192.168.3.0/24.

- ***Site-to-Site*** – u ovom primjeru je vidljivo kako se VPN poslužitelj unutar tvrtke (udaljena lokacija **A**), preko interneta, spaja na udaljeni VPN poslužitelj tvrtke na drugoj lokaciji (lokacija **B**), te se za to također stvara kriptirani *tunel*. Taj *tunel* ovdje omogućuje svim uređajima na lokalnoj mreži (192.168.2.0/24) da pristupaju svim mrežnim resursima tvrtke s druge lokacije odnosno koji su na mreži 192.168.3.0/24, a vidljivi su i dostupni kao da se radi o lokalnoj mreži.

U oba slučaja VPN poslužitelj na promet koji dolazi kroz svaki pojedini VPN tunel odobrava ili zabranjuje određena prava pristupa resursima unutar svoje mreže. To znači da može ograničiti pristup samo na određene resurse na mreži; primjerice pristup do samo određenih poslužitelja ili mrežnih uređaja ili računala i slično.

Osim toga određene stvari **VPN poslužitelj** nalaže tijekom uspostave veze ili dodatno može naložiti VPN klijentu, a među njima su:

- **Dodijela IP adrese** – VPN poslužitelj dodjeljuje VPN klijentu privatnu IP adresu unutar VPN mreže. Ova IP adresa se koristi za komunikaciju unutar VPN mreže. Poslužitelj može naložiti određenu statičku IP adresu ili dinamički dodijeliti adresu iz predefiniranog raspona.
- **DNS konfiguraciju** – poslužitelj može naložiti klijentu da koristi specifične DNS poslužitelje (npr. za sprječavanje DNS propuštanja ili za korištenje poslužitelja za filtriranje sadržaja). Ovi DNS poslužitelji se automatski konfiguriraju na klijentu nakon uspostavljanja VPN veze.
- **Pravila za usmjeravanje (*Routing*)** – poslužitelj može odrediti koje mrežne rute će klijent koristiti. Ovo može uključivati sve rute (za potpunu tuneliranje, kada sav promet ide kroz VPN), ili samo određene rute (tzv. ***split tunneling***, gdje samo određeni promet ide kroz VPN). Poslužitelj također može prisiliti klijenta da sav promet usmjerava kroz VPN ili može ograničiti pristup samo na određene resurse unutar mreže.
- **Prilagodba mrežnih postavki** - poslužitelj može konfigurirati dodatne mrežne postavke, kao što su **MTU** veličina (*Maximum Transmission Unit*), što može pomoći u optimizaciji performansi ili pomoću u normalizaciji mrežnog prometa.
- **Algoritmi za kriptiranje (enkripciju)** – poslužitelj određuje koji će enkripcijski algoritmi i protokoli biti korišteni (npr. AES-256, SHA-2 za *hashing*, itd.). Klijent mora koristiti iste enkripcijske postavke kako bi uspostavio sigurnu vezu.
- **Dozvole pristupa i autentikacija** – poslužitelj upravlja autentifikacijom korisnika (npr. putem korisničkog imena i lozinke, certifikata, ili dvofaktorske autentifikacije i sl.). Poslužitelj dodatno može ograničiti pristup određenim resursima na osnovu korisničkog identiteta ili pravila.
- **Ograničenja propusnosti i sesije** – poslužitelj može naložiti ograničenja propusnosti (***bandwidth throttling***) ili trajanja sesije, što može utjecati na količinu podataka koju klijent može prenijeti ili koliko dugo može ostati povezan.

Osim toga određene stvari **VPN klijent** i sam dodatno može konfigurirati, a među njima su:

- **Odabir VPN poslužitelja** – klijent može odabrati na koji će se VPN poslužitelj povezati (ako VPN pružatelj nudi više servera). Ovaj izbor može ovisiti o lokaciji poslužitelja, brzini, opterećenju, ili specifičnim potrebama (npr. geo-deblokiranje).
- **Postavke za DNS** – iako poslužitelj može naložiti korištenje specifičnih DNS poslužitelja, neki VPN klijenti omogućuju korisnicima da ručno promijene DNS postavke ili koriste vlastite DNS poslužitelje.
- **Split tunneling** - ako poslužitelj dopušta, klijent može konfigurirati tzv. ***split tunneling***, gdje korisnik odlučuje koji će promet prolaziti kroz VPN, a koji će ići izravno na internet. To omogućava zadržavanje visoke brzine za određene aplikacije dok se osigurava privatnost za druge.
- **Postavke enkripcije i protokola** - klijent može odabrati koji VPN protokol želi koristiti (npr. OpenVPN, IKEv2, *WireGuard*), pod uvjetom da poslužitelj podržava više protokola. Korisnici mogu ponekad prilagoditi postavke enkripcije, ali samo unutar granica koje podržava poslužitelj.
- **Kill switch** - klijent može uključiti ili isključiti tzv. ***kill switch*** funkciju, koja automatski prekida internet vezu ako VPN veza padne. Ovo osigurava da podaci ne odlaze na otvorenu mrežu (Internet) [često u kombinacija sa *Split tunnelingom*].
- **Postavke vatrozida** - neki klijenti omogućuju ručnu konfiguraciju ili dodavanje pravila za vatrozid (*firewall*), čime se dodatno osigurava privatnost.
- **Prilagodba mrežnih postavki** - klijent može konfigurirati dodatne mrežne postavke, kao što su **MTU** veličina (*Maximum Transmission Unit*), što može pomoći u optimizaciji performansi ili pomoću u normalizaciji mrežnog prometa; primjerice ako odredišna mreža ima manji **MTU**. Ako poslužitelj šalje MTU, njegova vrijednost MTU-a ima veću važnost od one postavljene od strane klijenta.
- **Skripte i automatizacija** - napredni korisnici mogu dodavati vlastite skripte ili automatizaciju (npr. skripte koje se pokreću prije ili poslije uspostavljanja veze) kako bi prilagodili ponašanje VPN klijenta prema specifičnim potrebama.

Često korišteni VPN klijenti i poslužitelji pod Linuxom su:

- **OpenVPN** - VPN softver koji koristi SSL/TLS protokol za šifriranje veze.
- **WireGuard** - moderan, brz i jednostavan VPN protokol koji je poznat po jednostavnosti i visokoj sigurnosti.
    - **Pritunl** je GUI za ***OpenVPN*** i ***WireGuard***, koji olakšava postavljanje i upravljanje VPN-om.
- **StrongSwan** – popularan za IPSec implementaciju koja podržava IKEv2 protokol.
- **xl2tpd + strongSwan** - *Layer 2 Tunneling Protocol* u kombinaciji s IPSec-om za sigurnu VPN vezu.
- **OpenConnect** - open-source VPN klijent koji podržava *Cisco AnyConnect* i kompatibilne SSL VPN-ove.

**Izvori informacija: (1724),(K-10),(K-12).**

### 26.8.8. Intrusion detection system (IDS) i Intrusion prevention system (IPS)

Pojam *Intrusion detection system* (**IDS**) odnosi se na sustav (komponentu) odnosno uređaj ili softversku aplikaciju koja nadzire mrežu ili računalne sustave zbog zlonamjernih aktivnosti ili kršenja zadanih pravila. Prilikom upotrebe ovakvih sustava svaka aktivnost ili kršenje zadanih pravila (pr. upada u sustav) obično se prijavljuje administratoru ili se prikuplja centralno pomoću sustava za upravljanje sigurnosnim informacijama i događajima zvanim **SIEM** (***S****ecurity* ***I****nformation and* ***E****vent* ***M****anagement*). Pri tome **SIEM** sustav kombinira izlaze iz više izvora te koristi tehnike filtriranja alarma za razlikovanje zlonamjernih aktivnosti od lažnih alarma. Prema načinu rada razlikujemo dvije vršne kategorije *IDS* sustava:

- Najčešća klasifikacija *IDS*-a su mrežni sustavi za otkrivanje upada odnosno **NIDS** (***N****etwork* ***I****ntrusion* ***D****etection* ***S****ystems*). Primjer ovakvog sustava je sustav koji analizira dolazni mrežni promet (pr. s interneta).
- Druga kategorija *IDS*-a su sustavi za otkrivanje upada na računalu odnosno **HIDS** (***H****ost* ***I****ntrusion* ***D****etection* ***S****ystems*). Primjer ovakvog sustava je primjerice softver koji nadzire važne datoteke operativnog sustava.

Međutim, također je moguće klasificirati *IDS*, prema načinu detekcije anomalija odnosno obrazaca napada:

- Najpoznatija varijanta jest detekcija temeljena na potpisu odnosno tzv. ***signature-based detection*** – radi prepoznavanjem loših obrazaca ponašanja na osnovu definiranih potpisa odnosno obrazaca. Ti potpisi dolaze iz različitih izvora poput sigurnosnih organizacija (pr. ***CERT***, antivirusne tvrtke) ili platformi za razmjenu informacija (pr. ***MITRE ATT&CK***, ***VirusTotal***, ***Emerging Threats*** [*ET Open*], **OISF,** …) ili od strane tvrtki ili organizacija koje razvijaju ili prodaju IDS/IPS sustave.
- Druga varijanta detekcije jest detekcija temeljena na anomalijama odnosno otkrivanje odstupanja od modela "dobrog" prometa, koji se često oslanja na strojno učenje.
- Sljedeća uobičajena varijanta jest otkrivanje temeljeno na reputaciji odnosno prepoznavanje potencijalne prijetnje prema ocjenama reputacije. Umjesto da se oslanjaju samo na statičke "*signature*" ili heuristiku, ovi sustavi ocjenjuju mrežne entitete (IP adrese, domene, URL-ove itd.) na temelju njihove reputacije, tj. povijesti aktivnosti i poznatih asocijacija s malicioznim ili benignim ponašanjem. Reputacijski podaci dolaze iz različitih izvora, uključujući globalne baze podataka, *feedove* prijetnji, izvještaje o zlonamjernom softveru, povijesne podatke o aktivnostima IP adresa, i druge podatke prikupljene iz mrežnih aktivnosti.

Vezano za detekciju metodom potpisa odnosno obrazaca: unutar definicije svakog obrasca napada uobičajeno se definira:

- ***Identifikator*** (***Signature ID***) – jedinstveni broj ili oznaka koja identificira specifični obrazac unutar IDS/IPS sustava. Ovaj identifikator omogućuje administratorima da brzo prepozna i pretraži određeni napad u sustavu log poruka.
- ***Naziv napada –*** kratki opis ili naziv koji opisuje vrstu napada.
- ***Opis napada –*** detaljniji opis onoga što obrazac prepoznaje, uključujući moguće posljedice napada ako prođe neotkriven. Na primjer, "*Obrazac prepoznaje pokušaj ubrizgavanja SQL koda u web aplikaciju preko HTTP zahtjeva*".
- ***Obrazac mrežnog prometa (Traffic Pattern)*** – precizno definirani niz podataka ili naredbi unutar mrežnog prometa koji upućuju na napad. Ovaj dio može uključivati:
  - ***IP adrese*** – specifične IP adrese s kojih se napad očekuje.
  - ***Portove*** – određeni portovi koji se koriste u napadu.
  - ***Protokole*** – specifični mrežni protokoli kao što su HTTP, TCP, UDP, … .
  - ***Nizove znakova*** – specifični nizovi unutar paketa, npr. zlonamjerni kod u HTTP zaglavlju ili tijelu poruke.
  - ***Binarne potpise*** – nizovi bajtova koji su specifični za određeni *malware* i drugo
- ***Akcijske upute (Action Taken") –*** *definira što IDS/IPS treba učiniti kada se prepozna napad. To može uključivati:*
  - ***Alarmiranje*** – obavještavanje administratora o sumnjivom prometu.
  - ***Blokiranje*** – prekidanje sesije ili blokiranje određenog prometa.
  - ***Logiranje*** – zapisivanje svih detalja o napadu za kasniju analizu.
- ***Informacije o inačici obrasca*** – kada je stvoren ili ažuriran, i tko ga je kreirao. To pomaže u održavanju i reviziji sustava.
- ***Pravila ili uvjeti pod kojima se obrazac primjenjuje*** – kao što su određeni vremenski okviri, tip prometa, ili kontekst u kojem se napad može dogoditi.
- ***Dodatne informacije ili uvjeti koji pomažu smanjiti lažno pozitivne rezultate***. Na primjer, može uključivati provjeru da li se određeni niz znakova pojavljuje u specifičnom kontekstu koji nije zlonamjeran.



Naziv detekcije metodom potpisa odnosno obrazaca preuzet je od **antivirusnih** programa, koji koriste isti mehanizam za detekciju virusa ili zlonamjernog softvera.

Iako su i *Vatrozidi* (*Firewalls*) i *IDS* sustavi zaduženi za mrežnu sigurnost, *IDS* se razlikuje od vatrozida po tome što konvencionalni mrežni vatrozid koristi statički skup pravila za dopuštanje ili odbijanje mrežnih veza (primjerice, prema IP adresi ili portu i slično). To u slučaju vatrozida sprječava upade, ali samo pod pretpostavkom da se radi o točno definiranom odgovarajućem skupu pravila. U osnovi, vatrozidi ograničavaju pristup između mreža kako bi spriječili upade, ali uobičajeno ne signaliziraju napade unutar mreže. Dok *IDS* sustav opisuje sumnju na upad nakon što se on dogodio, a potom i signalizira alarm prema primjerice ***SIEM*** sustavu. *IDS* također prati napade koji potječu iz branjenog (našeg) sustava.

To se tradicionalno postiže ispitivanjem mrežnih komunikacija, identificiranjem heuristike i obrazaca ponašanja (često poznatih kao *potpisi*) uobičajenih računalnih napada i poduzimanjem radnji za upozoravanje operatera. Na kraju, postoje i sustavi koji su u stanju prekinuti zlonamjerne veze, a nazivaju se sustavi za sprječavanje upada to jest ***I****ntrusion* ***P****revention* ***S****ystem* (**IPS**), a on obavlja i kontrolu pristupa poput vatrozida aplikacijskog sloja (*Application Layer Firewall*).

## Intrusion prevention system (IPS) sustavi.

Neki *IDS* sustavi imaju mogućnost reagiranja na otkrivene napade ili upade odnosno maliciozne radnje. Takvi sustavi s mogućnošću odgovora obično se nazivaju sustavima za sprječavanje upada odnosno **IPS** (***I****ntrusion* ***P****revention* ***S****ystem*). Naime *Intrusion Prevention System* (**IPS**) sustavi su sigurnosna rješenja dizajnirana za detekciju i automatsku prevenciju sumnjivih aktivnosti u mrežnom prometu ili na uređajima unutar informatičkih sustava. Za razliku od *IDS*-a, koji samo upozorava na potencijalne prijetnje, *IPS* poduzima akcije kako bi odmah spriječio napade.

Kako rade IPS sustavi:

- ***Aktivno praćenje mrežnog prometa*** - *IPS* sustav prati sav mrežni promet u stvarnom vremenu.
- ***Analiza prometa*** - IPS može koristiti različite metode za analizu prometa, kao i *IDS*, a to su:
  - ***Signature-Based Detection*** – prepoznaje poznate napade usporedbom prometa s unaprijed definiranim obrascima napada (tzv. "*signaturama*").
  - ***Anomaly-Based Detection*** – uspoređuje promet s normalnim obrascima ponašanja kako bi identificirao odstupanja koja mogu ukazivati na napad.
  - ***Behavior-Based Detection*** – prati ponašanje korisnika i sustava te detektira neuobičajene aktivnosti koje mogu biti pokazatelj prijetnje.
- Poduzimanje akcija odnosno ako *IPS* identificira zlonamjernu aktivnost, može automatski poduzeti sljedeće mjere:
  - ***Blokiranje prometa*** – onemogućavanje daljnjeg prijenosa sumnjivih paketa ili prekidanje sesija.
  - ***Odbacivanje paketa*** – odbacivanje specifičnih zlonamjernih paketa prije nego što dođu do cilja.
  - ***Modifikacija prometa*** – uklanjanje ili promjena dijelova paketa kako bi se neutralizirala prijetnja.
  - ***Resetiranje veze*** – prekidanje postojeće sesije ili veze između napadača i ciljanog sustava.
  - ***Upozoravanje*** – slanje obavijesti administratorima kako bi mogli poduzeti daljnje korake.
- ***Logiranje i praćenje*** – *IPS* sustavi obično bilježe sve incidentne događaje, uključujući detalje o napadu, poduzetim akcijama i drugim relevantnim informacijama za buduću analizu i poboljšanje sigurnosne politike.

Prednosti IPS sustava su:

- ***Automatska prevencija*** – *IPS* sustavi pružaju proaktivnu zaštitu, sprječavajući napade prije nego što uzrokuju štetu.
- ***Smanjenje rizika*** – sposobnost blokiranja sumnjivih aktivnosti smanjuje rizik od kompromitacije mreže ili uređaja.
- ***Poboljšana sigurnost*** – kombinacija različitih metoda detekcije omogućuje otkrivanje i sprječavanje širokog spektra prijetnji.
- ***Brzi odgovor*** – *IPS* sustavi djeluju u realnom vremenu, što je ključno za sprječavanje brzih i destruktivnih napada.

***IPS*** *sustav se najčešće postavlja između interneta i lokalne mreže ili između mrežnih segmenata mreže odnosno na ključnim točkama u mreži, gdje može pregledavati i analizirati svaki paket podataka koji prolazi kroz mrežu.*

Neki od češće korištenih *open source IDS* i *IPS* sustava za Linux su:

- **Snort**
  - Vrsta: IDS/IPS
  - Opis – *Snort* je jedan od najpopularnijih *open source* sustava za detekciju i prevenciju upada. Nudi mogućnosti detekcije u stvarnom vremenu, analizu protokola, prepoznavanje napada i još mnogo toga.
- **Suricata**
  - Vrsta: IDS/IPS
  - Opis – Suricata je napredni *open-source* IDS/IPS sustav koji nudi višestruke funkcionalnosti, uključujući analizu mrežnog prometa, prepoznavanje obrazaca napada i integraciju s alatima za obradu podataka poput *Elastic Search*-a.
- **OSSEC**
  - Vrsta: HIDS (***Host-based IDS***)
  - Opis – OSSEC je moćan *open source* HIDS sustav koji se koristi za praćenje integriteta datoteka, analize log datoteka i detekciju upada na razini računala (*hosta*).
- **Wazuh**
  - Vrsta: HIDS/IPS, SIEM
  - Opis – *Wazuh* je proširena inačica OSSEC-a s dodatnim funkcijama za centralizirano upravljanje, naprednu analizu i praćenje sigurnosti u oblaku, s mnogim proširenjima i dodacima (*open source i close sourse*), što ga čini SIEM rješenjem. Dakle *Wazuh* integrira SIEM funkcionalnosti, omogućujući prikupljanje, analizu i korelaciju sigurnosnih podataka iz različitih izvora u stvarnom vremenu. Ova funkcionalnost omogućuje organizacijama praćenje sigurnosnih događaja, otkrivanje anomalija i brzo reagiranje na sigurnosne incidente.

Što se komercijalnih rješenja tiče, imaju ih mnoge tvrtke poput: ***Cisco Firepower****,* ***Palo Alto Networks Next-Generation Firewall (NGFW), Check Point Next-Generation Intrusion Prevention System (NGIPS), FireEye Network Security (NX), … .***



Osnovne razlike između **IDS**-a i **IPS**-a su:

- **IDS** – sam o otkriva napade i upozorava administratore, ali ne poduzima nikakve akcije.
- **IPS –** otkriva napade i automatski poduzima korake da spriječi potencijalne prijetnje.

**Izvori informacija: (1725),(K-10).**

# Virtualizacija i Linux kontejneri

# 27. Virtualizacija i linux kontejneri

U ovoj cjelini govorit ćemo o virtualizaciji i Linux kontejnerima. Virtualizacijom dobivamo mogućnost korištenja više virtualnih računala na jednom (fizičkom) računalu. Pri tome svako od tih virtualnih računala može imati drugačiji operativni sustav. U konačnici sva virtualna računala koriste resurse samo jednog fizičkog računala pa su sâmim time hardverski resursi bolje iskorišteni. To se poglavito vidi na poslužiteljima koji pokreću veći broj virtualnih računala. Naime, ako gledamo iskorištenje resursa klasičnih poslužitelja, može se primijetiti kako su oni veći dio vremena poprilično neiskorišteni odnosno ne koriste sve resurse računala (CPU, RAM, disk, …) u potpunosti ili donekle optimalno. Stoga je iskorištenje resursa istog hardvera upotrebom više virtualnih računala puno bolje i isplativije. U čemu je razlika između klasičnih sustava i virtualizacije?
Naime kod klasičnih sustava kod kojih operativni sustav instaliramo direktno na hardver tijekom instalacije operativnog sustava, potrebno je instalirati i:

- Sve upravljačke programe (engl. *drivers*) za sâv hardver na računalu: matičnu ploču i njen ***BIOS*** te njen *čipset*, disk kontroler, mrežnu i zvučnu karticu kao i sâv ostali hardver odnosno za sve komponente računala.
- Sve aplikacije odnosno korisničke i druge programe, koje je potom potrebno i konfigurirati.

Slika 251 prikazuje nam ovakav (standardni) scenarij u kojem operativni sustav instaliramo direktno na hardver odnosno na disk računala, kao što činimo s našim stolnim (*desktop*), prijenosnim (*laptop*) ili mobilnim uređajima (mobiteli i tableti).
Dakle tradicionalno na hardver odnosno na disk računala prvo instaliramo operativni sustav sa svim potrebnim upravljačkim programima za sâv pripadajući hardver na njemu. Naime bez ispravnih upravljačkih programa naš operativni sustav uopće nije u mogućnosti koristiti hardver koji imamo u računalu. To znači kako bez ispravno instaliranog i funkcionalnog upravljačkog programa namijenjenog točno određenoj inačici operativnog sustava naš hardver ne može raditi. To znači kako recimo bez upravljačkog programa za disk kontroler uopće nećemo moći pristupiti našem disku (tvrdi disk/SSD ili CD/DVD-ROM), grafičkoj kartici (za prikaz slike na monitoru računala) te mrežnoj kartici (bez koje nema mreže i interneta) kao i svim drugim komponentama, kojih ima na desetke u svakom računalu, a kako je pojednostavljeno vidljivo na slici 251.

**Slika 251. Instalacija operativnog sustava instaliranog direktno na hardver s vidljivim upravljačkim programima (engl. *drivers*).**




Tek potom možemo instalirati sve naše aplikacije odnosno programe. Zatim je potrebno sve te programe i ispravno konfigurirati za naše potrebe. Čak i ako se radi samo o uredskom paketu programa poput primjerice *LibreOffice* ili *Microsoft Office*, potrebno je određeno vrijeme za konfiguraciju klijenta elektroničke pošte, programa za pisanje i oblikovanje tekstualnih dokumenata (MS *Word* ili *LibreOffice Writer*) i slično. Ako tome dodamo i druge specijalizirane programe poput posebnih uređivača teksta ili programske alate za programiranje kao i za neku drugu specifičnu struku, sâma konfiguracija obično može potrajati satima. U slučaju kada je namjena računala poslužiteljska, tada na ovo računalo (sada poslužitelj) instaliramo poslužiteljske servise čija konfiguracija i optimizacija je još kompleksnija, a nerijetko traje i danima.

**Što se događa kada dođe do nekog kvara računala?**
Današnja računala, ali i mnogi poslužitelji više nisu projektirani da traju desetke i više godina, pa moramo računati na to da će se sigurno neka od komponenti računala pokvariti. U slučaju kada nam se pokvarila sâmo matična ploča s procesorom (CPU) i radnom memorijom (RAM), ako smo imali sreće i prethodno izradili kopiju svih podataka, tada ćemo kupnjom novog računala i korištenjem starog diska vjerojatno uspjeti za dan-dva sve vratiti u donekle funkcionalno stanje. Ako nismo imali sreće, bit će potrebno iz početka instalirati operativni sustav kao i sve nove upravljačke programe, jer sada imamo potpuno drugačiji hardver, pa tek onda sve programe uz ponovno konfiguriranje istih i vraćanje starih podataka. Međutim, ako nismo redovito izrađivali sigurnosne kopije i/ili ako nam se pokvario disk s podacima (što je i najčešći uzrok gubitka podataka) tada imamo velikih, a možda i nerješivih problema. Što je gubitak podataka o kojem smo govorili. Gubitak podataka se može i hoće dogoditi na bilo kojem uređaju ili mediju na koji se spremaju podaci. Bilo koji gubitak podataka, čak i krivo svrstavanje/snimanje podataka se smatra nekom vrstom gubitka podataka.
Ono što nas najviše brine je trajni gubitak podataka važnih za naše ili vaše poslovanje. Vrste gubitka podataka su:

- **Ljudska greška** - slučajno ili nesvjesno (zbog neznanja): brisanje, prepisivanje ili modifikacija datoteka ili direktorija (mapa).
- **Korupcija datoteka** (Engl. *File Corruption*) - programska greška, infekcija virusom ili drugo.
- **Hardver** - kvar diska, disk kontrolera, procesora, memorije (RAM) ili slično.
- **Vezano za lokaciju** - greške u naponskoj mreži, gromovi, požar, poplave, potresi, ….

Prema podacima za Europu, glavnih šest uzroka gubitaka podataka su redom:

- **Hardver**, koji ukupno čini ~42%. Većinom su to greške od prenapona i greške na diskovima.
- **Ljudska greška**, koju čini 31 %. Većinom se događaju zbog slučajnog brisanja podataka.
- **Korupcija datoteka**, na koju otpada 13%.
- **Infekcija virusima**, koja je bila uzrok za 7% grešaka.
- **Krađa,** a koja se posebno odnosi na prijenosna računala je činila oko 5%.
- **Greške na lokaciji** su bile uzrok u oko 3% slučaja, većinom zbog grmljavine, problema s napajanjem i poplava.

**Izvori informacija vezanih za gubitke podataka**: (*The Cost Of Lost Data, David M. Smith*), (*National Archives & Records Administration in Washington*), (*Research by Computer Economics*), (*Strategic Research Institute*).

---

# 27.1. Virtualizacija

Virtualizacija je rješenje za većinu navedenih problema, ali kako ona stvarno radi i čemu sve služi?.
Upotrebom virtualizacije u odnosnu na tradicionalne operativne sustave instalirane direktno na hardver (računalo/poslužitelj) dobivamo i sljedeće prednosti:

- Smanjujemo troškove održavanja.
- Ubrzavamo proces izrade sigurnosne kopije cijelog operativnog sustava te svih aplikacija i podataka unutar njega. Naime cijelo virtualno računalo je u konačnici jedna datoteka koja predstavlja disk tog virtualnog računala. Ona se obično komprimira zajedno sa svim podacima o virtualiziranom hardveru od kojega se sastoji to virtualno računalo:
    - Disk kontrolera te vrste i veličine diska, matične ploče i *čipseta* te procesora (CPU-a).
    - Količina dostupne RAM memorije.
    - Mrežne kartice i svih drugih komponenti virtualnog računala.

Korištenjem virtualizacije mi, na fizičko računalo (hardver) više ne instaliramo naš željeni operativni sustav, već obično specijalizirani i prilagođeni operativni sustav, ako govorimo o takozvanoj punoj virtualizaciji (o kojoj sada i pričamo).
Na tom operativnom sustavu obično već imamo sve upravljačke programe ili ih eventualno naknadno instaliramo, što uglavnom nije potrebno zbog visokog stupnja integracije velikog broja upravljačkih programa u ovakve sustave. Dakle brzo smo riješili sâmu instalaciju operativnog sustava za virtualizaciju, koji sadrži i posebnu komponentu koja se naziva ***hipervizor***.
Fizičko računalo na koje se instalira specijalizirani operativni sustav s ***hipervizorom*** se još naziva i domaćin (engl. *Host*).
Navedeni ***hipervizor*** je zadužen za pokretanje virtualnih računala koje ponekad nazivamo virtualne mašine ili virtualni strojevi, a koji se nazivaju i gosti [engl. *Guest*], u smislu virtualizacije. Upravo o njima ćemo sada i govoriti.

**Slika 252. Upotreba virtualizacije.**




Kako smo do sada shvatili, za virtualizaciju je potrebna određena platforma odnosno takozvana *Hipervizor* platforma, bilo unutar samostalnog operativnog sustava ili unutar već postojećeg operativnog sustava koji ima mogućnost kreiranja virtualiziranog okruženja za naša virtualna računala. Važno je razumjeti da postoje dvije kategorije platformi za virtualizaciju:

- ***Tip* 2** koji se pokreće na već instaliranom operativnom sustavu. Najčešće korišteni su *VirtualBox* i *VMware player/workstation*, ali i neki drugi. Oni u pravilu nisu namijenjeni za *produkcijsku* upotrebu jer (namjerno) nemaju sve alate i mogućnosti samostojećih sustava za virtualizaciju. S druge strane zbog čega bi netko i razvijao istovremeno rješenja poput *VMware player/workstation*, ali i *VMware ESXi* kada među njima ne bi bilo razlike.
- ***Tip* 1** su samostojeći operativni sustavi s pripadajućim ***Hipervizorom***. Dakle cijeli paket: OS + *Hipervizor* je optimiziran za virtualizaciju te ne sadrži nepotrebne programe ili alate. Predstavnici ove grupe su: *VMware ESXi*, *Xen Server*, *Microsoft Hyper-V* (iako je on prema logici rada nešto između ove dvije vrste), *OpenStack* i naravno *Proxmox VE*, ali mnogi drugi.

Upoznali smo se s logičkom shemom ***Tipa 1*** platformi za virtualizaciju. Ovdje imamo pripadnike rješenja koja su inicijalno zamišljena i razvijana za velika okruženja sa stotinama ili tisućama poslužitelja. Osnovne dvije grane predstavnika ovakvih sustava, prema logičkoj implementaciji su:

- ***VMware ESXi***
- ***Microsoft Hyper-V*** kao i ***Xen Server***, ali i **OpenStack** te neki drugi poput ***Proxmox VE***. Iako *Proxmox VE* nije dizajniran za velika okruženja (ne za više od 32+ fizička poslužitelja i nekoliko tisuća virtualnih računala i Linux kontejnera).

Njihova ideja je instalacija osnovnog *hipervizora* direktno na poslužitelj (zbog toga pripadaju kategoriji ***Tip-1***), a uz koje dobivamo sâmo osnovne funkcionalnosti (koje imamo i u *VMware player/workstation* ili *Virtual Boxu*), ali i više od toga.
Sve važne funkcionalnosti koje stvarno želimo od ovakvog sustava obično dobivamo instalacijom Tzv. *Virtual Appliance-a* odnosno virtualnog računala koje se pokreće unutar tog *Hipervizora*. Upoznajmo se s nekima od njih.

***VMware ESXi***
Kod njega nakon instalacije osnovnog *hipervizora* direktno na fizičko računalo, dodatno se instalira i jedno virtualno računalo koje se zove ***vCenter Server*** tj. ***VCSA***, a koje u najosnovnijoj konfiguraciji ima integriran tzv. *Embedded Platform Controler* koji pokreće ***PostgreSQL*** bazu podataka u koju se snimaju sve konfiguracijske postavke:

- Od konfiguracije svakog virtualnog računala.
- Preko mrežne konfiguracije svakog poslužitelja i virtualnih računala.
- Do konfiguracije *klastera (engl cluster)* kao i drugih virtualnih računala (tzv. *Virtual Appliancea*) tj. za dodatne funkcionalnosti sustava.

Klijenti koji se spajaju na menadžment sučelje se zapravo spajaju na gore navedeno virtualno računalo koje nakon dodavanja svakog pojedinog *ESXi* (*hipervizor*) poslužitelja u *vCenter/vSphere* postaje svjesno i svih drugih virtualnih računala te naravno svih ostalih komponenti cijelog sustava. Redundancija (zalihost) se postiže posebnom implementacijom više ovakvih virtualnih računala koja se po mogućnosti trebaju pokretati na različitim fizičkim računalima; zbog pouzdanosti i visoke dostupnosti.

***Microsoft Hyper-V***
Kod Microsoft **Hyper-V** rješenja stvar je slična kao kod *VMware ESXi* sustava s time da se ovdje nudi nekoliko *Virtual Appliance-a* odnosno virtualnih računala od kojih svako pokriva jedan dio funkcionalnosti, pa tako imamo:

- Virtualno računalo: **System Center Virtual Machine Manager**.
- Virtualno računalo: **System Center Configuration Manager**.
- Virtualno računalo: **System Center Operation Manger**.

I ovdje se sve konfiguracije pohranjuju na (ovaj puta) ***Microsoft SQL Server*** koji isto mora biti negdje instaliran, a obično unutar nekog (četvrtog) virtualnog računala.

## Rješenja bazirana na Linuxu

Drugi pripadnici ***Tipa 1*** virtualizacije su ***OpenStack*** i ***Proxmox VE*** (ali i mnogi drugi). Mi ćemo ipak opisati sâmo dva navedena.

**OpenStack**
Ovo rješenje je slično kao prethodna dva s time da ovdje nema potrebe za instaliranjem virtualnih računala koja nude određene funkcionalnosti, već se to postiže instaliranjem tzv. uloga odnosno "*Rola*" na svaki fizički poslužitelj. Tako nam je za minimalno funkcioniranje sustava potrebna minimalno jedna instalacija takozvanog ***Kontrolera*** sa svojim pripadajućim SQL poslužiteljem na koji se pohranjuju sve konfiguracije sustava.
Dodatno potrebna nam je i ***Network*** *rola* odnosno poslužitelj koji ima pristup svim unutarnjim i vanjskim mrežama i preko kojeg prolazi mrežni promet: unutra ↔ van. Komponenta odnosno *uloga* koja je zadužena za ovaj rad se zove ***Neutron***.
Svi ostali poslužitelji na kojima se pokreću virtualna računala moraju imati *rolu* odnosno ulogu takozvanog ***Compute*** poslužitelja, koja se zove ***Nova***.  Za normalan rad nam trebaju i sljedeće uloge odnosno *role*:

- **Horizon** je zadužen za Web sučelje.
- **Keystone** je zadužen za korisničke račune, *policy* i sl.(tzv. *Identity service*).
- U slučaju kada nam trebaju predlošci (*template-i*) tada nama treba i ***Heat*** komponenta.
- Kao *Block storage* se koristi komponenta koja se zove ***Cinder***.
- **Glance** je menadžer za slike (*image)* virtualnih računala, a tu je još i cijeli nîz drugih komponenti tj. *uloga*.

Vezano za redundanciju odnosno zalihost nudi se slično rješenje kao kod *VMware ESXi* i *Microsoft Hyper-V* dakle prvo slijedi instalacija i konfiguracija komponenti koje su preduvjet za redundanciju te konfiguraciju udvostručenih uloga (*rolea)* i to na više različitih fizičkih poslužitelja. To kreće od SQL poslužitelja koji sada mora biti redundantan i po mogućnosti u "*Multi-Master*" konfiguraciji, a ispred kojega mora biti instaliran *Load Balancer* koji isto mora biti redundantan. Ispred njega mora postojati i neka komponenta na IP sloju (*Layer* 3), koja također mora biti redundantna. I sve druge uloge moraju biti posebno ("poduplano") instalirane na različitim fizičkim poslužiteljima te konfigurirane za ovakav način rada. Sve navedeno drastično komplicira dizajn i znanja potrebna za otkrivanje grešaka, kada se jednom pojave. Međutim ovakav dizajn nudi drastično povećanje kapaciteta na desetke, stotine ili više fizičkih poslužitelja u velikom *OpenStack* sustavu odnosno takozvanom *klasteru*.



***OpenStack*** *u pravilu kao Hipervizor koristi onaj ugrađen u Linux kernel odnosno* ***KVM+QEMU****, ali je prema potrebi u mogućnosti spajati se (i koristiti) i vanjske hipervizore poput: VMware ESXi ili MS Hyper-V* i druge.

**Proxmox VE**
***Proxmox VE*** je namijenjen za relativno manja okruženja i ograničen je na maksimalno 32 fizička poslužitelja.
Zbog potpuno drugačije namjene i njegov dizajn je potpuno drugačiji od gore navedenih, iako i ovdje imamo slične komponente.
I dalje imamo lokaciju gdje se pohranjuju apsolutno sve konfiguracije: od konfiguracija svakog virtualnog računala ili mrežnih komponenti, preko sustava za pohranu podataka, vatrozida i slično.
Međutim pošto je ovo prema dizajnu znatno manji sustav od gore navedenih, stoga je bio moguć određeni kompromis.
Naime ovdje nema potrebe za glomaznim SQL poslužiteljima i njihovom redundancijom odnosno klasterom (engl. *cluster*) jer je zamišljeno da možemo imati *sâmo* do 32 fizička poslužitelja, a ne stotine njih.
To u praksi znači kako nećemo imati desetke tisuća ili stotine tisuća virtualnih računala u našem podatkovnom centru, već vrlo vjerojatno sâmo nekoliko stotina ili tisuća virtualnih računala i/ili Linux kontejnera, pa nam stvarno ne treba veliki SQL poslužitelj ili *klaster* SQL poslužitelja (u slučaju redundancije). Razvojni tim zadužen za **Proxmox VE** se odlučio koristiti takozvani ***Corosync*** kao višestruko dokazan *klasterski* sustav ispred kojeg se nalazi datotečni sustav koji koristi SQlite.
On se sinkronizira između svih poslužitelja koji se nalaze u *Proxmox VE* klasteru.
Naime ovdje se radi o takozvanom Proxmox cluster file system rješenju, za koje možemo reći sljedeće:

- Stabilan je i u produkcijskoj je upotrebi cijeli niz godina: u produkcijskoj upotrebi je od 2008. godine.
- Pouzdan je i otporno na ispade bilo kojeg poslužitelja ili više njih, a ovisno o njihovom broju.
- Distribuiran je pa svi poslužitelji imaju pristup svim konfiguracijskim datotekama u svakom trenutku.
- Sve promjene na bilo kojoj konfiguracijskoj datoteci na bilo kojem poslužitelju u klasteru su vidljive apsolutno svim ostalim ***Proxmox VE*** poslužiteljima u *klasteru* i to trenutno, odnosno čim se promjena dogodila.

Naime svaka promjena bilo koje konfiguracije se odmah sinkronizira sa svim poslužiteljima u *klasteru* pa je tako već u početku sve redundantno. To znači kako se ispadom bilo kojeg fizičkog poslužitelja u *klasteru* neće ništa dogoditi što se tiče konfiguracije, jer se ona ionako replicirana na sve poslužitelje u *klasteru*. Dodatno i uloga "*Kontrolera*" odnosno poslužitelja koji vodi brigu o svim virtualnim računalima i njihovim resursima, isto je redundantna jer oni ionako dijele sve konfiguracije preko *klasterskog* dijeljenog datotečnog sustava pa stoga svi poslužitelji u klausteru imaju ovu "*ulogu*". U konačnici je svejedno i na koji ste se fizički poslužitelj spojili preko Web sučelja jer oni svi imaju instalirani mali web poslužitelj, te svi oni "vide" sve ostale poslužitelje, sve konfiguracijske datoteke, sva virtualna računala i sve ostalo što je potrebno za rad. Možemo reći da, ako smo kreirali *Proxmox VE klaster* s tri (ili više) poslužitelja u njemu, tada imamo trostruku (ili višestruku) redundanciju što se tiče menadžmenta i konfiguracije. Osim toga, svaki pojedini fizički poslužitelj u *Proxmox VE klasteru* je:

- *Kontroler* poslužitelj i *Web* poslužitelj.
- Takozvani *Identity* poslužitelj zadužen za korisničke račune i grupe te prava pristupa.
- *SQL* poslužitelj odnosno njegov ekvivalent, ali i *Network* poslužitelj.
- I takozvani *compute* poslužitelj odnosno *Hypervizor* poslužitelj uz još štošta drugo.

Uz napomenu kako se ništa dodatno ne mora instalirati na ***Proxmox* VE** poslužitelje jer je sve već dostupno i automatski instalirano te spremno za rad. Potrebno je sâmo znanje za konfiguraciju svega što vam treba za dalji rad. Ostaje otvoreno sâmo pitanje s redundantnim sustavom za pohranu podataka (NAS/SAN) kao što je slučaj i kod drugih sustava, odnosno o tome se sami moramo pobrinuti. *Proxmox VE* po pitanju sustava za pohranu podataka trenutno nudi upotrebu lokalnih: *direktorija*, *ZFS*, *LVM* ili *LVM-thin* te mrežnih dijeljenih sustava: *NFS*, *CIFS (Samba)*, *RBD* (CEPH), *CephFS*, *iSCSI* te *ZFS over iSCSI*.
**Što se tiče virtualizacije, *Proxmox VE* koristi standardnu Linux virtualizaciju: QEMU + KVM dok za Linux kontejnere (trenutno) koristi LXC tehnologiju.**

---

**Klasteri za virtualizaciju**
U poglavlju: **13.12. NAS/SAN sustavi i druge tehnologije** govorili smo o klasterima za pohranu podataka te načinima kako oni rade. U načelu svaki klaster (engl. *cluster*) sastoji se od minimalno dva ili obično više sustava (računala, poslužitelja i sl.) koji se koriste zbog redundancije (zalihosti) i sigurnosti/pouzdanosti, a obično i zbog veće brzine rada zbog paralelne obrade i/ili pristupa. Slično je i s klasterima za virtualizaciju. Svrha klastera za virtualizaciju je osigurati neprekidni pristup kritičnim servisima ili aplikacijama. To radi tako da ako jedan od fizičkih poslužitelja (*hipervizora*) koji pokreće određeno virtualno računalo zakaže, virtualizacijski klaster trenutačno raspoređuje radno opterećenje na druge fizičke poslužitelje, osiguravajući minimalno vrijeme zastoja i sprječavajući gubitak podataka. To čini klastere vrlo otpornim i tolerantnim na greške.
Ova značajka virtualizacijskog klastera se zove ***High Availabiliy cluster*** odnosno sustav visoke dostupnosti, a mehanizam preraspodjele virtualnog računala na drugo fizičko računalo (*hipervizor*) se obično naziva ***Cluster resource scheduler***. Druga važna značajka virtualizacijskih klastera je njihova skalabilnost. To znači da je na njima vrlo jednostavno instalirati i pokretati (ili brisati) nova virtualna računala bez ometanja rada postojećih virtualnih računala, kao i dodavati nove fizičke poslužitelje u klaster. S obzirom na činjenicu da su ukupni fizički resursi klastera raspodijeljeni na više fizičkih poslužitelja (*hipervizora*), lakše je dodjeljivati resurse virtualnim računalima. Napredniji virtualizacijski klasteri omogućuju i sljedeće **značajke**:

- Mogućnost replikacije sustava za pohranu: što omogućuje replikaciju virtualnih računala s primjerice lokalnih ili mrežnih odnosno dijeljenih diskova dodijeljenih jednom fizičkom poslužitelju na drugi.
- Sustav za izradu sigurnosnih kopija (engl. *backup*) te mehanizam za povrat (engl. *restore*) virtualnih računala iz sigurnosnih kopija. Kao i podršku za klonove, predloške (engl. *template*) i snimke stanja (engl. *snapshot*) u radu virtualnih računala, a u naprednim sustavima i njihovo kopiranje na željeni drugi fizički poslužitelj u klasteru.
- Migraciju virtualnih računala s jednog na (bilo koji) drugi fizički poslužitelj; obično tzv. ***Live*** migracijom (na živo) ili takozvanom ***Offline***: gašenjem na prvom fizičkom poslužitelju, kopiranje na drugi pa pokretanje na drugom.
- Napredne mrežne modele rada sustava: primjerice tzv. ***Bridge*** način rada ili ***Routing*** način rada. Te upotrebu naprednih mrežnih protokola za mrežnu redundanciju (pr. ***LACP*** i sl.), kao i napredne vatrozide i *Software Defined Network* (**SDN**) sustave.

**Izvori informacija: (1517),(1518),(1519),(1520).(1521),(1522),(1523).(1524),(1525),(1526). Pogledajte i poster P5.**

---

## Virtualizacija i mreža

Bez obzira koju platformu i *hipervizor* za virtualizaciju koristili, mrežni model veze između fizičkog računala (*hipervizora*) i virtualnih računala ugrubo se može podijeliti u dva načina rada

### Bridge način rada

Način rada upotrebom mrežnog mosta (engl. *bridge*), koji rade poput fizičkih mrežnih preklopnika, ali implementiranih u softver. U takvom radu svi virtualni gosti (virtualna računala) mogu dijeliti jedan mrežni most ili je moguće stvoriti više mrežnih mostova za odvojene mrežne domene. Svako fizičko računalo (poslužitelj) može imati do 4.094 sučelja mrežnih mostova. U ovom načinu rada virtualna računala imaju pristup mreži od OSI sloja dva (OSI 2) do svih viših slojeva. To znači da se virtualna računala ponašaju kao da su izravno spojena na fizičku mrežu, a njihove virtualne mrežne kartice se ponašaju kao da su u pitanju fizičke te pri tome svaka između ostaloga ima i svoju **MAC** adresu, koja ovisi o *hipervizoru*. Mreža, također s druge strane vidi da svako virtualno računalo ima svoju vlastitu ***MAC*** adresu odnosno adresu svoje [virtualne] mrežne kartice.
Osnovne značajke mrežnih mostova su (vidljivo na slici 252.1. s desne strane [B]):

- Mrežni mostovi rade na sloju dva OSI modela. Dakle oni mrežne okvire prihvaćaju i obrađuju gledajući njihove ***MAC*** adrese jer ne znaju (niti ih zanima) ništa o TCP/IP i višim komunikacijskim protokolima.
    - To znači da mrežni mostovi propuštaju i protokole poput: ***ARP***, ***ICMP***, ***IGMP***, ***ECN***, ***NGP***, …
- Mrežni mostovi na osnovi **MAC** adrese u svakom mrežnom okviru saznaju na koji su priključak mrežnog mosta povezana virtualna računala. Kada mrežni most prosljeđuje mrežni okvir, on ga šalje samo na mrežni priključak koji je najbliži odredištu odnosno odredišnoj **MAC** adresi.



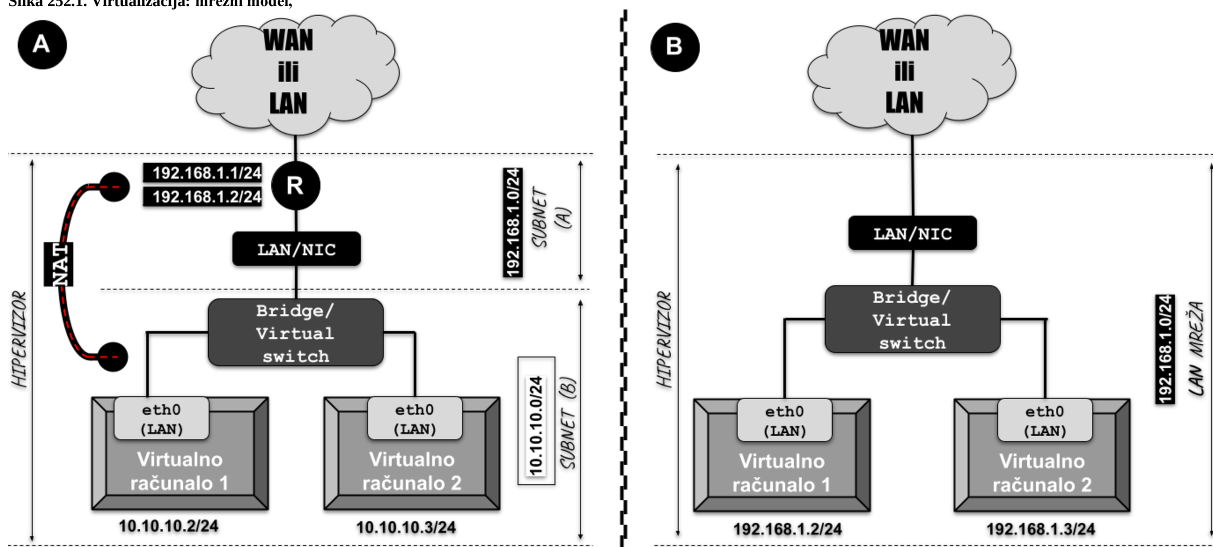
Vezano za **MAC** adrese, pogledajte poglavlje:
**19.3.1. Što su MAC adrese.**

### Routing i NAT način rada

Većina pružatelja usluga ne podržava ***Bridge*** način rada. Oni iz sigurnosnih razloga onemogućuju umrežavanje čim otkriju više MAC adresa na jednom sučelju. Stoga imamo ***Routing*** odnosno ***NAT*** način rada, koji omogućuje gostima (virtualnim računalima) koji imaju samo privatne IP adrese, pristup mreži korištenjem IP adresa hipervizora (domaćina) za odlazni promet. To se odrađuje upotrebom tehnike translacije odnosno prevođenja adresa znane i kao *Network address translation* (**NAT**).
Ona se odnosi na metodu preslikavanja jednog IP adresnog prostora odnosno opsega IP adresa, u drugi.
Kod ovakvog načina rada u pravilu su moguća dva scenarija (vidljivo na slici 252.1. s lijeve strane [A]):

- Preslikavanje IP adresa s unutarnje mreže u kojoj imamo privatni IP adresni prostor, na drugu mrežu, s drugim, obično javnim ili čak drugim privatnim adresnim prostorom, tako da svako virtualno računalo dobije svoju jedinstvenu fiksnu vanjsku (obično javnu) IP adresu. Ovakav načina rada se naziva **NAT 1:1**.
- Preslikavanje svih IP adresa s unutarnje mreže u kojoj imamo privatni IP adresni prostor, na drugu mrežu, s drugim, obično javnim adresnim prostorom, tako da sva virtualna računala za izlaz koriste javnu (vanjsku) IP adresu.
  U ovakvom *NAT* načinu rada, naš usmjerivač (ili komponenta Linuxa [R]) za svaki naš mrežni paket na lokalnoj mreži (LAN) koji izlazi van, mijenja izvorišnu IP adresu sa svojom vanjskom (javnom) IP adresom, skrivajući stvarne privatne (unutarnje) IP adrese virtualnih računala. Ovakav način rada je poznati i kao: ***One-to-many NAT***, ***Port address translation*** (**PAT**), ***IP masquerading*** ili ***NAT overload*** odnosno ***Many-to-one NAT***.

Slika 252.1. Virtualizacija: mrežni model,




**Izvori informacija: (1442), (K-8)**

### 27.1.1. Linux kao hipervizor

Sada kada smo se upoznali s raznim sustavima za virtualizaciju, vratimo se našem čistom Linuxu i njegovom *Hipervizoru* za virtualizaciju.

***Hipervizor*** je komponenta koja pokreće virtualno računalo tako da za njega (za virtualno računalo) emulira sâv potreban hardver:

- Od matične ploče i njenog BIOSa te njenog *chipseta*.
  Podsjetite se poglavlja: **10.1.1. Matična ploča, CPU, *chipset* i sabirnica(e)**.
- Te naravnog centralnog procesora (CPU) i svih njegovih značajki. Podsjetite se poglavlja: **10.7. CPU**.
- Disk kontrolera i diskova. Podsjetite se poglavlja: **14. Diskovni (I/O) podsustav**.
- Mrežnih kartica. Podsjetite se poglavlja: **19.2.2. Mrežne kartice**, kao i drugog hardvera (pr. grafička kartica).

Naime za svako virtualno računalo *hipervizor* mora emulirati sâv navedeni hardver računala, kao da se radi o fizičkom računalu, kako bi na neki način *"prevario"* operativni sustav koji ćemo potom instalirati na to virtualno računalo.
Dakle *hipervizor* emulira sve komponente koje imaju i fizička računala te ih stavlja unutar virtualnog računala na koje mi potom instaliramo operativni sustav. Na tom operativnom sustavu virtualnog računala potom instaliramo upravljačke programe za taj virtualni odnosno emulirani hardver: matična ploča i *chipset*, CPU, disk kontroler i drugo, kao da se radi o normalnom fizičkom računalu.

Važno je razumjeti da se u inicijalnom dijelu instalacije operativnog sustava unutar virtualnog računala, kada se instaliraju upravljački programi, instaliraju se upravljački programi za emulirane (virtualne) komponente našeg virtualnog računala koje nam je dodijelio *hipervizor*. Ako smo recimo kao disk kontroler za naše virtualno računalo unutar *Hipervizora* odabrali *IDE* (ATA) disk kontroler, tada ćemo dobiti primjerice virtualiziran (*emuliran*) *PIIX3* disk kontroler sa svojim pripadajućim BIOS-om i svim svojim funkcionalnostima. Dakle u trenutku kada krenemo s instalacijom našeg virtualnog računala, primjerice s **ISO** slike našeg operativnog sustava, kada se budu morali instalirati upravljački programi za disk kontroler, bit će odabran *PIIX3* disk kontroler kao da se radi o stvarnom (hardverskom) disk kontroleru ***PIIX3***.
Ista je stvar sa svakom dugom komponentom našeg budućeg virtualnog računala.



Za efikasnu virtualizaciju potrebna je i podrška centralnog procesora (CPU) koja drastično ubrzava sve procese potrebne za virtualizaciju. ***Intel*** ovu funkcionalnost unutar centralnog procesora naziva **VT-x**, a ***AMD*** ju naziva **AMD-V**.

Operativni sustav i sve aplikacije unutar virtualnog računala nisu svjesne da se nalaze unutar virtualnog računala te se ponašaju kao da su instalirane na "normalno" odnosno fizičko računalo.

## ***KVM* + *QEMU***

Ulogu *Hipervizora* pod Linuxom odrađuje takozvani ***KVM*** (*Kernel-based Virtual Machine*). On je zapravo kernel modul za virtualizaciju u Linux kernelu, koji omogućuje da kernel funkcionira kao *hipervizor*. On je integriran u Linux kernel od inačice kernela 2.6.20, koja je objavljena 5. veljače 2007. *KVM* se učitava kao kernel modul imena `kvm`. ***KVM*** zahtijeva procesor s hardverskom podrškom za virtualizaciju, poput Intel ***VT-x*** ili ***AMD-V***. *KVM* nam u kombinaciji s navedenom podrškom unutar procesora, a koja usput rečeno postoji negdje od 2005. godine, nudi virtualizaciju za mnoge operativne sustave (tzv. goste prema terminologiji za virtualizaciju).

Dakle podržana je virtualizacija: *Linuxa, BSD Unixa, Solaris Unixa, Windowsa, Haiku OS, ReactOS, Plan 9, AROS Research Operating System, macOS*, kao i *Android, GNU/Hurd, Minix, Darwin OS*-a i drugih.
Pri tome je važno razumjeti kako sâm **KVM** ne emulira hardver već je samo *Hipervizor* za virtualna računala.

Naime *KVM* se preko linux kernela oslanja na fizički hardver, a emulaciju virtualnih komponenti: matična ploča (BIOS) i *chipset*, CPU, disk kontroler, mrežna kartica i drugi, odrađuje druga komponenta koja se zove ***QEMU*** (*Quick EMUlator*). Pogledajte sliku 253.



***QEMU*** ne simulira već **emulira** hardver, što znači kako je ponašanje emuliranog (*virtualnog*) hardvera praktično identično stvarnom (fizičkom) hardveru jer ga kao takvog i doživljavaju virtualna računala za koja je namijenjen.

Slika 253. Veza između hardvera, kernela: KVM i QEMU i virtualnog računala.




Pogledajmo i što sve **QEMU** emulira. Pri tome nećemo navesti baš sve kategorije i uređaje, već sâmo one osnovne:

- Matičnu ploču sa svojim *BIOSom* (*SeaBIOS*) koji podržava: *ATA DMA i Bus mastering, UHCI/OHCI/EHCI/xHCI, podršku za VGA (grafičke kartice), odabir CD-ROM medija za pokretanje sustava, BIOS BBS pozive* (*Boot Specification*)*,LBA adresiranje, VESA BIOS ekstenzije (VBE), PCI firmware specifikaciju, TPM i drugo.*
- *Chipseti: PIIX3 (PCI IDE ISA Xcelerator) ili i440FX (noviji).*
- Mnoge arhitekture računala: *x86, x86-64, MIPS64, Sun Sparc (sun4m i sun4u), ARM, PowerPC, Risc-V,…*
- Razne modele x86 procesora: ***Intel****: Core duo, Sandy i Ivy Bridge, Haswell, Broadwell, …* ***AMD****: Opteron, Epyc.*
  Kao i posebno emulirane (univerzalne) virtualne procesore poput: *kvm32* i *kvm64* ili *qemu32* i *qemu64*.
- Disk kontrolere: ATA ili SATA AHCI te SCSI (*LSI MegaRAID SAS 1078, LSI53C895A, NCR53C9x,Tekram DC-390*) i druge******.
- Grafičke kartice: *Cirrus* CLGD 5446, Standardnu VGA (Bochs-VESA-BIOS-Extensions) te ***Red Hat***: QXL VGA ili VirtIO GPU te posebnu SPICE karticu kao i druge kartice**.
- Mrežne kartice: RTL8139, Intel Gigabit (PRO/1000 tj. `e1000`), NE2000, SMC91c111 i druge******
- Paralelne, serijske, PS/2 i USB portove.

**Imamo i posebne takozvane *paravirtualizirane*** VirtIO uređaje:**

- *VirtIO* SCSI i blok uređaji (`vioscsi` i `viostor`).
- *VirtIO* mrežna kartica (`virtio-net`) i *VirtIO grafička kartice* (`virtio-vga`).
- *VirtIO* ***IOMMU*** *(*`virtio-iommu`*) i VirtIO konzola (*`virtio_console`*).*
- *VirtIO* *balloon*; ako je Hipervizor: Linux ***KVM+QEMU*** (`virtio_balloon`), odnosno ako je to ***VMware*** (`vmw_balloon`) ili ako je to ***Hyper-V*** (`hv_balloon`).
- *VirtIO generator slučajnih brojeva (*`virtio_rng`*) i VirtIO kripto uređaj (*`virtio_crypto`*).*

Iz navedenog smo vidjeli kako **QEMU** može emulirati sve komponente računala, što je prijeko potrebno kako bi mogli kao virtualno računalo instalirati nemodificirani operativni sustav po želji, poput primjerice: *Windows* 10, *Windows Server* 2019, *Ubuntu Linux*, *FreeBSD Unix* ili neki drugi operativni sustav.
Ovdje je zanimljivo i to da je moguće umjesto klasičnih ***emuliranih*** uređaja poput disk kontrolera, mrežne i grafičke kartice, koristiti i takozvane ***paravirtualizirane***** uređaje. Što su oni i čemu služe?

***Linux KVM kao Hipervizor podržava NUMA arhitekturu i nudi dodatne optimizacije za ovu arhitekturu računala!.***

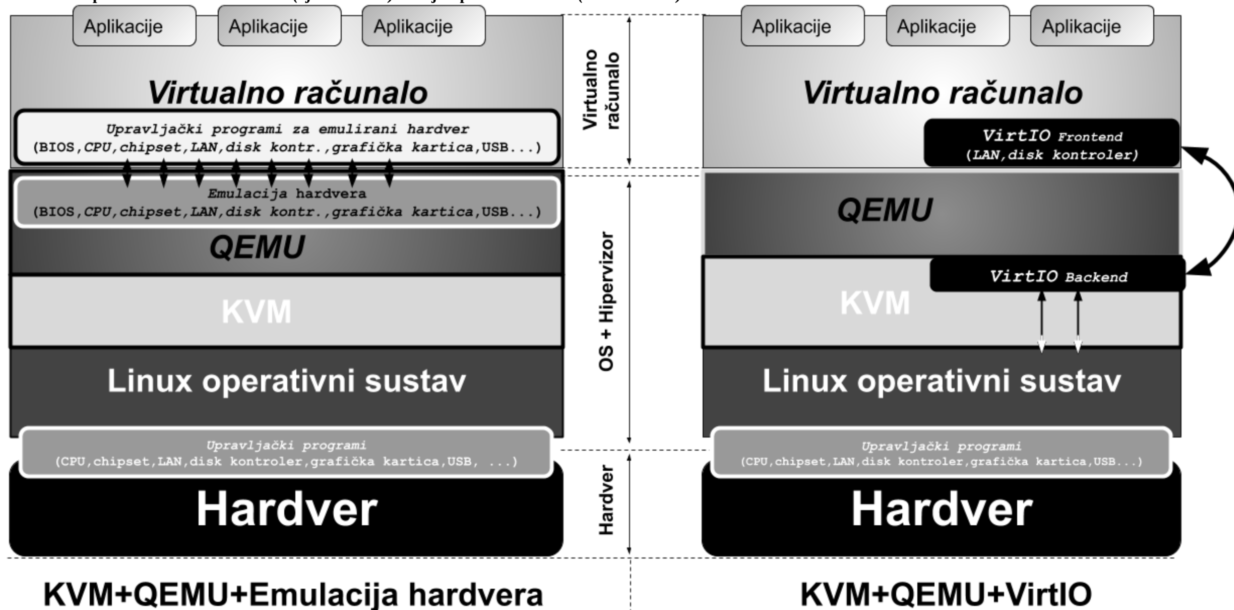
**Izvori informacija: (868),(869),(870),(871),(872),(K-8).**

## Paravirtualizirani uređaji

Paravirtualizirani uređaji odnosno uvjetno rečeno hardver; poput disk kontrolera, mrežne ili grafičke kartice, za razliku od emuliranog hardvera, radi tako da *hipervizor*, a u ovom slučaju *QEMU* uopće ne radi emulaciju tog hardvera.
Naime kako smo do sada već mogli zaključiti: kod *emulacije* hardvera emulira se rad svog sklopovlja navedenog hardvera uz pripadajući BIOS za taj hardver. Jasno je kako su za to potrebni resursi računala, iako je veći dio ovih operacija ubrzan zbog podrške za virtualizaciju unutar današnjih procesora.
Međutim za *emuliranje* rada nekih uređaja poput disk kontrolera ili pogotovo mrežnih kartica, koje moraju baratati s milijunima mrežnih paketa u sekundi, klasična emulacija dolazi do svojih ograničenja. Naime poglavito na većim brzinama mreže od 1Gbps, a pogotovo na 10Gbps ili više, trošilo bi se previše resursa na emulaciju pa je uvedena takozvana *paravirtualizacija* uređaja, što se konkretno odnosi (za sada) na disk kontrolere, mrežne i grafičke kartice. Ova funkcionalnost je razvijena unutar takozvanog ***VirtIO*** paketa upravljačkih programa i sustava.

*VirtIO paravirtualizirani* upravljački programi poboljšavaju performanse rada unutar virtualnog računala, smanjujući kašnjenje (latenciju) ulazno izlaznih operacija (I/O) te povećavajući propusnost gotovo na razinu rada fizičkog računala. Naime *VirtIO* nudi ***API*** (sučelje za programsku komunikaciju) za ulazno/izlazne operacije (I/O) preko kojega se komunikacija odrađuje na sljedeći način, kako je vidljivo na slici 254 (desna strana).

**Slika 254. Usporedba: klasični način rada *(lijevi dio slike)* i onaj s upotrebom *VirtIO (desni dio slike)*.**




Naime kako je vidljivo na slici 254, upotrebom ***VirtIO*** za mrežne kartice i disk kontrolere unutar virtualnog računala, instalira se upravljački program za ***VirtIO*** mrežnu karticu i za disk kontroler. Taj upravljački program je praktično sâmo jedna komponenta cijelog upravljačkog programa, a stoga je označen kao *Frontend* odnosno gornji dio, dok drugu stranu komunikacije s *VirtIO* sustavom čini jezgra *VirtIO* sustava (*Backend*) koja se nalazi na razini KVM komponente.

KVM potom direktno komunicira s kernelom i sâmim time s hardverom. Zbog ovakvog načina komunikacije preskače se potreba za emulacijom hardvera te se zaobilazi QEMU i komunicira direktno s KVM-om. S time se dobilo drastično ubrzanje rada. Međutim i dalje sve druge komponente virtualnog računala (*BIOS, CPU, chipset, USB*,…) moraju biti emulirane pošto one nisu zamišljene za rad preko *VirtIO* sustava jer za to niti nema potrebe.

## Podrška za KVM u Linuxu

Podrška za *KVM* u Linuxu prvo mora biti omogućena u sâmom kernelu te moramo imati kompiliran `kvm` kernel modul.
Provjerimo imamo li *KVM* podršku u našem kernelu:

```
grep CONFIG_VIRTIO /boot/config-`uname -r`
```

```
CONFIG_VIRTIO_BLK=m
CONFIG_VIRTIO_NET=m
CONFIG_VIRTIO=y
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_PMEM=m
CONFIG_VIRTIO_BALLOON=m
```

Najvažnije kategorije su: `VIRTIO=y`, `VIRTIO_BLK`, `VIRTIO_NET`, `VIRTIO_PCI` kao i neki drugi.
Dakle minimalno moramo imati postavljene na `=y`, i to: `CONFIG_VIRTIO=y` i `CONFIG_VIRTIO_PCI=y` dok drugi mogu biti dostupni kao kernel moduli odnosno imati oznaku `=m`.
Tek sada možemo učitati *KVM* kernel modul, kako bismo mogli kreirati virtualno računalo odnosno koristiti virtualizaciju:

```
modprobe kvm
```

Ovakvim učitavanjem `kvm` kernel modula učitat će se i dodatni kernel moduli specifični za *Intel* ili *AMD* procesore preko kojih se (ako procesor ima podršku za to) hardverski ubrzavaju sve radnje vezane za virtualizaciju. Za *Intel* procesore to možemo provjeriti tražeći CPU zastavicu (*flag*) **`vmx`** odnosno za AMD **`svm`**. To možemo provjeriti sa sljedećom naredbom:

```
grep -m1 -o1 -i -e vmx -e svm /proc/cpuinfo
```

Hardverska podrška za virtualizaciju (u hipervizoru) se uključuje s opcijom: `-accel kvm` (kasnije ćete vidjeti gdje i kako).

**Izvori informacija: (873), (874),(875),(876),(K-8).**

### 27.1.2. Rad s virtualizacijom (KVM+QEMU)

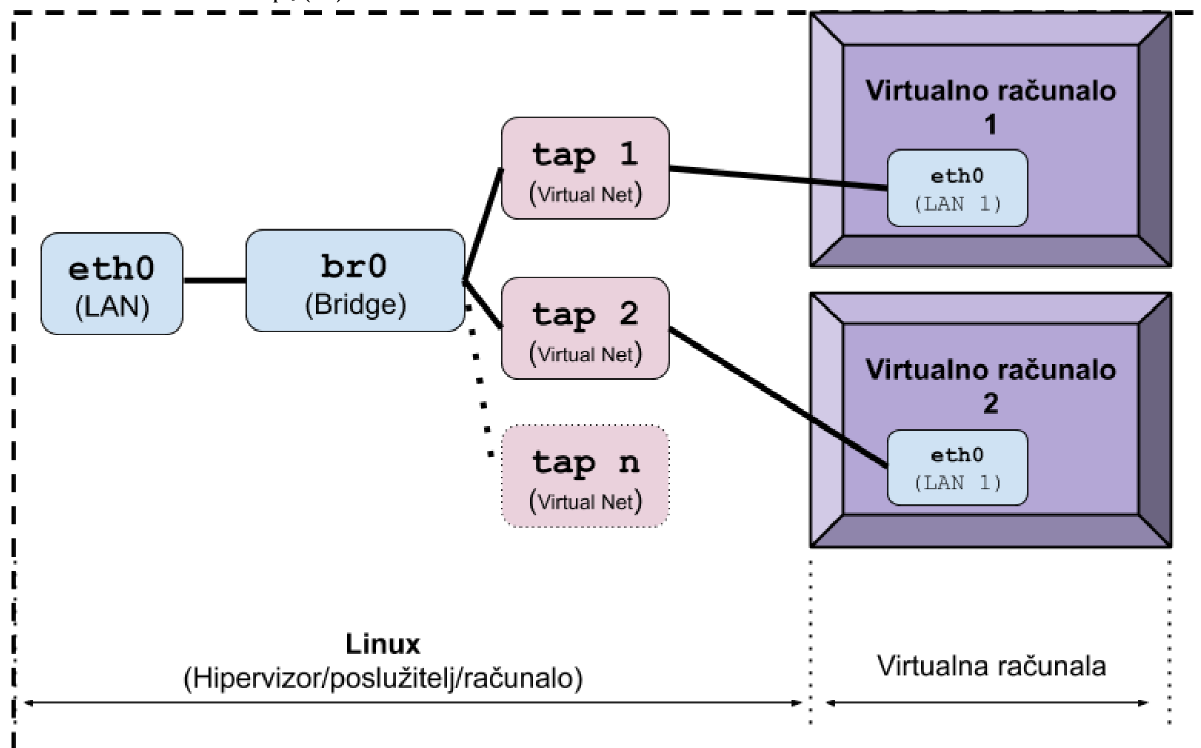
Važno je znati kako gotovo sâv emulirani hardver, u što pripadaju matična ploča i njen BIOS, *chipset*, CPU, disk kontroler, zvučne i druge kartice, osim mrežne, kroz ***QEMU***, preko ***KVM*** sustava komuniciraju dalje s linux kernelom i u konačnici hardverom računala. Jedina iznimka je mrežna kartica, koja bilo da je emulirana (pomoću ***QEMU***) ili koristimo li takozvani ***VirtIO***, ona s Linuxom kao *hipervizorom* i u konačnici hardverom mreže komunicira na malo drugačiji način.

#### 27.1.2.1. Konfiguracija mreže

Mreža virtualnog računala se na *hipervizor* može spojiti sâmo pomoću posebnog mrežnog sučelja zvanog ***TAP***. Naime da bi se ispod razine *hipervizora* to ***TAP*** mrežno sučelje spojilo s fizičkom mrežnom karticom kako bi se mrežni paketi mogli slati na mrežu, njega (*TAP*) je prvo potrebno nekako povezati s fizičkom mrežnom karticom. Međuveza između ***TAP*** mrežnog sučelja povezanog s mrežnom karticom unutar virtualnog računala i fizičkom mrežnom karticom je posebno premosno mrežno sučelje (Engl. *Bridge*), na slici vidljivo kao **br0**. Pogledajmo logičku shemu ovakve međuveze na slici 255.

***Prednost upotrebe bridge sučelja je u tome što se ono može vezati i za logička sučelja (pr. Bond sučelje zbog redundancije).***

**Slika 255: Međuveza: Linux eth0-br0-tap → (VM) eth0**




Poveznicu između ***TAP*** sučelja i virtualne mrežne kartice unutar virtualnog računala odrađuje ***QEMU*** emulator o kojem ćemo govoriti kasnije. Važno je razumjeti kako se svako novo pojedino ***TAP*** mrežno sučelje u pravilu povezuje s jednom mrežnom karticom unutar virtualnog računala. Prema tome, ako recimo naše virtualno računalo ima dvije mrežne kartice, morat ćemo na razini *hipervizora* odnosno pripadajućeg Linuxa kreirati dva ***TAP*** mrežna sučelja.
Stoga se prisjetimo sljedećih poglavlja.



Vezano za ***TAP*** mrežno sučelje pogledajte poglavlje:
**20.6.3. TUN i TAP - posebna mrežna sučelja.**
Vezano za mrežni most (*Bridge*) pogledajte poglavlje:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**

**U novije vrijeme postoji i potencijalna zamjena za TAP sučelje a zove se MACVTAP:**
Pogledajte poglavlje:
**26.8.2. Druga mrežna sučelja (*MACVLAN,IPVLAN,VXLAN,MACVTAP/IPVTAP*).**



*U primjerima koji slijede, sve radimo na testnom računalu korištenjem Linux bridge sučelja, s direktnim pristupom na njega, jer ćemo u jednom trenutku izgubiti sve mrežne veze prema tom računalu, a koje ćemo kasnije vratiti.*

***Umjesto Linux bridge sučelja, mogli smo koristiti i [Open vSwitch](Open vSwitch) i njegovo bridge sučelje (poglavlje:26.8.5.)***

***Prvo zapišimo naše IP parametre mrežne kartice `eth0`(ili koju već imate): IP adresu i masku mreže, te Default gateway.***

**1.** U prvom koraku, za sva buduća virtualna računala moramo kreirati *bridge* mrežno sučelje.
Dakle kreirajmo *bridge* mrežno sučelje imena: `br0` upotrebom naredbe `ip` na sljedeći način:

```
ip link add name br0 type bridge
```

**2.** Sada aktivirajmo ovo novo `br0` mrežno sučelje upotrebom naredbe `ip` na sljedeći način:

```
ip link set dev br0 up
```

**3.** Potom s *bridge* mrežnim sučeljem (`br0`) povežimo naše fizičko mrežno sučelje (`eth0`):

```
ip link set dev eth0 master br0
```

Zatim pogledajmo kako sada izgleda naše *bridge* mrežno sučelje `br0`:

```
ip link show master br0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP
```

Ovdje vidimo vezu: `br0` ←→ `eth0`. Sada nam još treba ***TAP*** sučelje i njegova veza na *bridge* sučelje (`br0`).

**4.** U ovom koraku kreirajmo ***TAP*** mrežno sučelje imena: `tap0` te ga aktivirajmo:

```
ip tuntap add tap0 mode tap
ip link set dev tap0 up
```

**5.** Dodajmo i `tap0` mrežno sučelje u naš *bridge* `br0`, sa slijedećom naredbom, kako bi imali vezu: `eth0`←→`br0`←→`tap0`

```
ip link set dev tap0 master br0
```

Pogledajmo kako sada izgleda naše *bridge* mrežno sučelje `br0` (skratili smo ispis):

```
ip link show master br0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP
4: tap0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP
```

Vidimo da imamo vezu ***TAP*** sučelja (`tap0`) te naše fizičke mrežne kartice (`eth0`) preko *bridge* sučelja (`br0`).
Isto možemo vidjeti i s naredbom `bridge` koja dolazi u istom softverskom paketu kao i naredba `ip`:

```
bridge link show br0
```

**6.** Pošto sada na `eth0` sučelju ne smijemo imati IP parametre, obrišimo ih (s mrežnog sučelja `eth0`) pa pokrenimo:

```
ip address flush dev eth0
```

**7.** I sada ako želimo konfigurirati IP adresu: 192.168.1.10 uz masku mreže (*netmask*): 255.255.255.0 za naše mrežno sučelje `br0` tada ćemo morati napraviti sljedeće:

```
ip address add 192.168.1.10/24 dev br0
```

Ne zaboravimo dodati podrazumijevani usmjerivač (*Default gateway).* U našem slučaju je to IP adresa: 192.168.1.1:

```
ip route add default via 192.168.1.1
```

Svi navedeni primjeri konfiguriraju sva navedena sučelja i IP parametre privremeno, pa će nakon restarta računala sve biti vraćeno na staro stanje.



*Ne zaboravimo da se IP parametri definiraju na **bridge** sučelju, a ne više na fizičkoj mrežnoj kartici koja je dio bridge sučelja*

.

Kasnija veza između svakog pojedinog ***TAP*** mrežnog sučelja i virtualizirane mrežne kartice unutar virtualnog računala se definira u trenutku pokretanja virtualnog računala, s opcijom poput: `-netdev type=tap,id=`**`net0`**`,ifname=`**`tap0`**`,`... pri čemu se za ovaj identifikator (`id=`**`net0`**) veže virtualizirana mrežna kartica koja će sadržavati opciju `netdev=`**`net0`**.

## *Drugi mrežni parametri i opcije*

Kada u sljedećim koracima budemo pokretali virtualno računalo, važno je razumjeti kako se njemu dodjeljuje virtualizirana ili *paravirtualizirana* mrežna kartica, ovisno kako je konfigurirano. Dakle moguće je odabrati bilo koju od sljedećih mrežnih kartica: RTL8139 (`-device rtl8139`), Intel Gigabit Pro 1000 (`-device e1000`), NE2000, *SMC91c111* te ***VirtIO*** (paravirtualizirana) ili ***vmxnet3*** (*paravirtualizirana* mrežna kartica za *VMware* virtualna računala) koja će se potom koristiti unutar virtualnog računala.

Nadalje važno je razumjeti kako se virtualna mrežna kartica unutar virtualnog računala može vezati prema *hipervizoru* na dva načina:

- Prvi i standardni način o kojem cijelo vrijeme i govorimo je način povezivanja s takozvanom ***bridge*** metodom, spajanjem na ***TAP*** mrežno sučelje koje radi na OSI sloju dva (OSI 2) pa su tako našem virtualnom računalu dostupni svi mrežni slojevi od OSI sloja dva na više. Ova metoda komunikacije je i najbrža i preporučena.

- Drugi i manje standardan način (koji nećemo detaljno objašnjavati) je spajanjem u takozvanom ***NAT*** načinu spajanja u kojem se spajamo na *hipervizor* od kojega moramo dobiti IP parametre iz posebnog privatnog opsega adresa koje onda potom *hipervizor* pretvara (***NAT***=*Network Address Translation*) odnosno translatira u neki drugi opseg IP adresa s kojim izlazimo izvan ove izolirane mreže. Ovakav način rada zahtjeva postojanje ***DHCP*** poslužitelja u posebnoj mreži na kojoj *hipervizor* opslužuje virtualna računala (kako bi ona mogla dobiti IP parametre) uz dodatan rad **NAT**-a. U svakom slučaju ovaj način rada je prilično sporiji od ***bridge*** načina rada, a osim toga on radi na OSI sloju tri (OSI 3).

Nakon što smo odabrali virtualiziranu (*emuliranu*) mrežnu karticu, ovisno o njoj imamo dostupne i neke opcije. Ako se odlučimo za ***VirtIO*** mrežnu karticu čija upotreba se i preporučuje zbog njene brzine i njenih mogućnosti, dobivamo sljedeće:

- Po potrebi možemo koristiti i takozvanu ***Multiqueue*** funkcionalnost. Dakle *VirtIO* mrežna kartica nam nudi mogućnost da koristimo više nîzova odnosno kanala u mrežnoj komunikaciji, tako da svaki komunikacijski kanal može opsluživati po jedna CPU jezgra (**RSS**). Dakle, ako naše virtualno računalo ima više CPU jezgri, svaka od njih može odrađivati zadatke vezane za primanje i slanje paketa prema mreži (odnosno prema hipervizoru na dalje). Ova funkcionalnost ovisi o podršci u hipervizoru[(885)] i nižim slojevima (pr. ***TAP**** sučelju). Ako primjerice imamo dvije CPU jezgre za virtualno računalo (*vCPU*) i želimo uključiti ovu funkcionalnost, tada će opcije i parametri s kojima pokrećemo naše virtualno računalo biti nešto poput: `-device virtio-net-pci,mq=on,vectors=6,…`

Pri čemu smo broj šest (**6**) dobili na sljedeći način: 2(rx + tx queue nîz) x **2CPU** + 2za konfiguraciju i kontrolu (config + control).
Tada, ako unutar virtualnog računala pokrenemo sljedeću naredbu vidjet ćemo ova dva (2) ***Multiqueue*** nîza:

```
ethtool -l eth0
Channel parameters for eth0:
Pre-set maximums:
Combined:       2
Current hardware settings:
Combined:       2
```

Vidjet ćemo i kako se ***VirtIO*** mrežne kartice raspodjeljuju između CPU jezgri (pogledajte `/proc/interrupts`).



Za detalje oko ove funkcionalnosti pogledajte poglavlje:
**25.2.2. Multiqueue funkcionalnost.**

- Dodatno je moguće i ograničiti brzinu prometa kroz virtualizirano odnosno paravirtualizirano (***VirtIO***) mrežno sučelje.

**Izvori informacija: (877),(883),(884),(1414),(1415),(1416),(1417),(1418),(K-8),** **`man qemu-img`**, **`man partprobe`**, **`man fdisk`**, **`man e2fsck`**, **`man resize2fs`**, **`man growpart`**, **`man fstrim`**, **`man 5 fstab`**, **`man 5 mount`**.

#### 27.1.2.2. Praćenje i analiza rada *TAP* sučelja

U određenim slučajevima možemo imati potrebu analizirati mrežni promet koji dolazi na ***TAP*** sučelje te u konačnici i na virtualiziranu (*emuliranu*) mrežnu karticu unutar virtualnog računala; što zbog praćenja prometa ili zbog analize rada u slučajevima kada imamo određene poteškoće poput gubitka paketa od ili prema virtualnom računalu. Dakle s programima poput Wireshark, `tcpdump` ili sličnim možemo pratiti mrežne pakete na našem ***TAP*** sučelju.

Primjerice za `tap0` bi to bilo s naredbom:

```
tcpdump -i tap0
```

Ako sumnjamo da imamo određene gubitke u mrežnim paketima prema mreži virtualnih računala, prvo moramo otkloniti sve eventualne probleme na nižim slojevima, krenuvši od:

- Sabirnice i mrežne kartice na njoj; pogledajte poglavlje: **10.2. (Re)konfiguracija sabirnice i uređaja na njoj.**
- Signala prekida (***IRQ***) i direktnog pristupa memoriji (***DMA***); pogledajte poglavlja: **10.5.1.1.**, **10.5.2.**
- CPU afiniteta; pogledajte poglavlja: **10.7.2.1.1.** i **10.7.2.2.1.** te **10.7.2.2.2.**
- Optimizacije upravljačkih programa mrežnih kartica; pogledajte poglavlje: **11.1.1.3.**
- Postavki brzine, *duplex* i drugih parametara rada mrežne kartice; pogledajte poglavlje: **19.6.1.** i povezana poglavlja.
- Kontrolu protoka na OSI sloju dva te sva povezana mrežna sučelja na OSI sloju dva; **poglavlje 20.6.** Naime sva mrežna sučelja na ovom sloju koja koristite, a u vezi su s ***TAP*** sučeljem mogu biti uzrok problema.
- Problemi s ***MTU*** postavkama; opisano u poglavlju: **23.4.2** ili s velikim mrežnim okvirima (*Jumbo*), kako je opisano u poglavlju: **23.4.3.** Ove postavke moraju biti usklađene na svim prolaznim mrežnim sučeljima i uređajima.
- Problemi s protokolima za redundanciju na OSI sloju tri; kako je opisano u poglavlju: **23.4.4.1.**
- Mnogih ***TCP*** parametara, vidljivih u cijelom poglavlju o TCP-u (poglavlje: **24.2. TCP**), a poglavito:
    - Opcija za skaliranje TCP prozora; poglavlje: **24.2.8.2.**
    - Odabira TCP algoritma za nadzor zagušenja; poglavlje: **24.2.9.**
    - *Timera* i stanja veze; poglavlje: **24.2.16.1.** i drugih.
- Postavki međumemorija na raznim slojevima mreže; vidljivo u poglavlju: **25.1.1.**
- Optimizacije veličine TX međumemorije (***txquelen***), kako je opisano u poglavlju: **25.1.2.**
- Odabira mehanizma za raspodjelu mrežnih paketa; kako je opisano u poglavlju: **25.1.4.**
- Razne druge optimizacije i mehanizmi*****, poput:
    - ***RSS*** (*Receive Side Scaling*) mehanizam i *Multiqueue* funkcionalnost, opisan u poglavlju: **25.2.1.**
    - ***RPS***, ***RFS*** i ***XPS*** mehanizmi, opisani u poglavlju: **25.2.3.**
    - ***LRO i GRO***, opisani u poglavlju: **25.5.1.** te ***LSO, TSO i GSO*** mehanizmi, opisani u poglavlju: **25.5.2.**
    - ***Checksum offload*** mehanizam, opisan u poglavlju: **25.5.3.**
    - ***Scatter-gather*** mehanizam, opisan u poglavlju: **25.5.4.**

***** ***Na ekstremno opterećenim virtualnim računalima, u određenim konfiguracijama, preporuka je imati uključen samo GRO mehanizam!***

Tek sada dolazimo do samog ***TAP*** mrežnog sučelja, pa u slučaju da je to sučelje imena `tap0`, statistike mrežnih paketa koji su došli ili poslani s njega, kao i grešaka, ali i odbačenih te drugih paketa, možemo vidjeti s `ip` naredbom na sljedeći način:

```
ip -s -s link ls dev tap0

4: tap0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP qlen 1000
RX: bytes          packets     errors  dropped   overrun mcast
    18754183253615 49951618012 0       0         0       717133
    RX errors: length   crc      frame   fifo     missed
               0        0        0       0        0
TX: bytes          packets     errors  dropped carrier collsns
    6272630160751 23038565067 0       0       0       0
    TX errors: aborted  fifo    window heartbeat
               0        0        0       0
```

U našem slučaju nemamo nikakvih grešaka (`errors`,`dropped` ili `overrun`) za primanje (RX) ili slanje (TX), kako bi i trebalo biti. Međutim u slučaju kada nešto nije u redu, mogu se pojaviti greške koje ćemo vidjeti u ovim statistikama.
`RX:` statistika ovdje se odnosi na pakete koji odlaze s virtualnog računala prema hipervizoru odnosno dalje na mrežu.

U slučajevima kada pod `TX:` statistikom imamo puno paketa koji su odbačeni (`dropped`), a provjerili smo sve gore navedeno, još je jedino moguće da sporadično (ako se to događa povremeno) virtualno računalo ne stigne procesirati sve mrežne pakete iz memorije koja se koristi za slanje: od ***TAP*** sučelja prema virtualnoj mrežnoj kartici (unutar virtualnog računala).
Uzrok može biti (povremeno) prezagušeno virtualno računalo pa je u tom slučaju jedino što ovdje možemo optimizirati, eventualno veličina TX međumemorije ***TAP*** sučelja, koja je označena kao `qlen`, koja se odnosi na postavku naziva `txqueuelen` čije detalje možete pročitati u poglavlju: **25.1.1**.

Veličina ove međumemorije definira maksimalan broj mrežnih paketa koji se mogu spremiti u međumemoriju mrežnog sučelja (konkretno `tap0`) za slanje dalje. Za probu ga povećajmo dvostruko više nego je standardno postavljen, dakle povećajmo ga na ***2000*** (nekada je potrebno znatnije povećanje) i to ovako:

```
ip link set tap0 txqueuelen 2000
```

U slučaju kada se ove greške odbacivanja paketa događaju sporadično, s navedenom promjenom možemo riješiti problem. Trajno ovakvu promjenu možemo napraviti kreiranjem nove *udev rules* datoteke koja bi sadržavala nešto poput:

```
SUBSYSTEM=="net", ACTION=="add", KERNEL=="tap*", ATTR{tx_queue_len}="2000"
```

Međutim, ako nam se to konstantno događa, tada je moguće da moramo raditi drugačiju optimizaciju, a ako koristimo ***VirtIO*** virtualnu mrežnu karticu, konkretno možemo uključiti i ***Multiqueue*** funkcionalnost kako je opisano na prethodnoj stranici.



Ne zaboravite kako ***VirtIO*** virtualizirane mrežne kartice nemaju svoju međumemoriju (***ring buffer***) pa je jedina međumemorija prema virtualnoj mreži virtualnog računala, upravo ova na ***TAP*** sučelju. Dodatno na njima (***VirtIO***) nećete moći niti vidjeti statistike na razini mrežne kartice koje bi inače dobili s naredbom poput: `ethtool -S eth0`.

**Izvor informacija: (910), `man ip`, pogledajte i poster P5.**

## 27.1.2.3. Disk kontroleri

Važno je znati kako ***QEMU*** i ***KVM*** vezano za disk kontrolere i na kraju diskove koje virtualna računala koriste, imaju mogućnost ograničavanja njihovog rada:

- Ograničavajući brzinu protoka podataka za snimanje i/ili čitanje: **MB/s.**
- Ograničavajući broj operacija prema disku za snimanje i/ili čitanje: **ops/s** (***IOPS***).
- I druga ograničenja, što ovisi o inačici ***QEMU***/***KVM*** komponente sustava.

## TRIM i DISCARD

Važno je znati da ***QEMU*** i ***KVM*** imaju mogućnost emuliranja rada **SSD** diskova uz podršku za tzv. ***TRIM*** metodu, ali i takozvanu ***Discard*** opciju. Ove opcije znače da se operativnom sustavu unutar virtualnog računala, a koje ili mora podržavati **TRIM** (za SSD diskove) ili ***Discard*** opciju ***(****za volume menadžere* poput **LVM2** ili **ZFS*)***, u slučaju kada datotečni sustav označava blokove kao neiskorištene nakon brisanja datoteka, virtualni diskovni kontroler može te informacije proslijediti *hipervizoru* te njegovom diskovnom sustavu. Njegov diskovni sustav će u skladu s time smanjiti diskovnu sliku (tzv. *Disk Image*) brisanjem nepotrebnih blokova podataka, na najučinkovitiji način: ovisno o tome koriste li se TRIM ii DISCARD, ali svakako učinkovito.
Nadalje, bilo na *hipervizoru* ili virtualnom računalu: moguće je *Discard/* opciju (opcija `discard`) dodati i u `/etc/fstab` unose, za ***swap***, ***ext4***, ***XFS***, ***Btrfs*** ili druge datotečne sustave na particijama ***SSD*** ili ***NVMe*** diskova, ili na particijama *volume menadžera* koji to podržavaju. Iako to primjerice za ***ext4***, ***XFS*** i ***Btrfs*** nije preporuka, dok **ZFS** ionako koristi svoje mehanizme za to. Stoga se uglavnom radije preporuča upotreba **FSTRIM** mehanizma odnosno servisa (o njemu ubrzo).



Da bi virtualno računalo uopće moglo koristiti ove opcije, moramo *hipervizoru* naložiti da virtualno računalo koristi *paravirtualizirani* disk kontroler *VirtIO SCSI* [**`virtio-scsi-pci`**] te da ima aktiviranu opciju ***Discard*** (`discard=on`).

Sve navedeno podrazumijeva dodavanje sljedećih opcija na Hipervizoru u dio konfiguracije za disk kontroler virtualnog računala:

```
-device virtio-scsi-pci, …
-drive file=...,discard=on,detect-zeroes=unmap,...
```

Dodatna opcija koju smo ovdje dodali je `detect-zeroes` koja može biti postavljena na `on`/`off` (uklj./isklj.) ili `unmap`.
Ona može detektirati (ako je u `on`) pokušaje zapisivanja *nula* (tj. brisanja) prema diskovnom sustavu. Ako je postavljena na `unmap` to znači da će dodatno poslati i *Discard/TRIM* opciju prema diskovnom sustavu. Za starije operativne sustave virtualnih računala koja ne podržavaju *Discard/TRIM* opciju, preporuka je postaviti ju na `unmap` odnosno postavite ju bez obzira na OS virtualnog računala. ← Konkretne opcije ovise o inačici *KVM/QEMU* servisa.



Za detalje pogledajte o **TRIM/DISCARD/UNMAP** pogledajte poglavlje: **13.3.1.3. TRIM funkcionalnost.**

***Vratimo se na hipervizor***
Pogledajmo na *hipervizoru* (fizičkom računalu), njegove diskove te podržavaju li *TRIM/Discard* opciju:

```
lsblk --discard
NAME          DISC-ALN DISC-GRAN DISC-MAX DISC-ZERO
sda                  0      512B       2G         0
└─sda1               0      512B       2G         0
sdb                  0        0B       0B         0
└─sdb1               0        0B       0B         0
```

Na ovom sustavu imamo dva diska (`/dev/sda` i `/dev/sdb`). Pri tome je disk `sda` ***SSD*** i podržava *TRIM/Discard*.
To na ispisu vidimo po tome što on pod stupcima `DISC-GRAN` i `DISC-MAX` ima neke vrijednosti veće od `0B`.

**FSTRIM mehanizam (i naredba)**
Naredba `fstrim` je dostupna na većini sustava i ona se koristi za već montirane datotečne sustave da se odbace (obrišu) blokovi koje datotečni sustav više ne koristi. Ovo se često koristi kod virtualnih računala koja na *hipervizoru* koriste ili ***SSD*** diskove ili tzv. *thin provisioning* sustave za pohranu, kao što je primjerice LVM Thin.
Navedenu naredbu `fstrim` eventualno možemo pokrenuti i ručno za sve diskove; u primjeru koji slijedi:

```
fstrim -a
```

Međutim na novijim distribucijama Linuxa postoji *fstrim* mehanizam (servis) koji se pokreće automatski (obično jednom tjedno) unutar ***Systemd*** servisa ***fstrim,*** kako je definirano u datoteci ***fstrim.timer***.
*Fstrim* mehanizam stoga možemo aktivirati i s ovim servisom (ako je slučajno isključen) i to pomoću naredbe:

```
systemctl enable fstrim.timer
```

---

## Odabir disk kontrolera

Kako smo vidjeli KVM/QEMU odnosno *hipervizor* pod Linuxom za svako virtualno računalo nudi nam odabir disk kontrolera prema želji, pogledajmo neke od njih uz njihove osnovne značajke:

- ***IDE***, ***SATA*** i ***SCSI*** kontroleri - njihova osnovna značajka je da su oni emulirani i korisni su u nekim okruženjima.
- ***VirtIO paravirtualizirani*** kontroleri koji su drastično brži od svih gore navedenih kategorija kontrolera:
  - Blok (`virtio-blk`) - predstavlja blok uređaj virtualnom računalu. Svaki *virtio-blk* uređaj pojavljuje se kao disk unutar virtualnog računala. On je i najbrži, ali je moguće imati maksimalno do 28 ovakvih diskova. *TRIM/Discard* opcija je ovdje podržana tek od kernela 5.x. međutim *SSD* emulacija ovdje nije podržana.
  - SCSI (`virtio-scsi-pci`) – predstavlja se virtualnom računalu kao SCSI kontroler. SCSI nudi bogatiji skup naredbi od *virtio-blk* te podržava drastično veći broj diskova. Stoga se može dobro skalirati i na **NUMA** sustavima. On dodatno podržava i napredne značajke, poput već navedene podrške za ***TRIM***, **I/O Thread**, ali i ***SSD* emulaciju**. U nekim implementacijama za ***I/O Thread*** potrebna je dodatna optimizacija upotrebom ***VirtIO SCSI Single*** koji radi tako da se na njega može spojiti samo jedan disk, pa se za više diskova mora koristiti više ***VirtIO SCSI Single*** kontrolera što ionako poboljšava performanse (CPU *pinning*, IRQ, …).

## *I/O Thread* optimizacije

Moguće je koristiti i tzv. ***IO Thread*** za *VirtIO* disk kontrolere, koji za svaki novi disk kreira novi komunikacijski kanal kroz *hipervizor* do Linux računala (hardvera), što također može ubrzati diskovne operacije. To radi tako da se otvara zasebna programska nît izvan QEMU sustava na koji se blok uređaji mogu prikvačiti kako bi se značajno poboljšala izvedba takvog diskovnog sustava. Ova opcija je obično vidljiva kao (`iothread`):

```
-device virtio-scsi-pci,iothread=XYZ,...
```

← Konkretne vrijednosti (**XYZ**) ovise o inačici *KVM/QEMU* komponente.
***IO Thread*** poboljšava paralelizaciju i raspodjelu opterećenja među dostupnim CPU jezgrama.

## *Multi-queue virtio-scsi*

*Multi-Queue Virtio-SCSI* je napredna opcija koja omogućuje korištenje više redova za upravljanje ulazno/izlaznim (I/O) operacijama između hipervizora i virtualnih računala (VM-ova). Ova tehnologija omogućava paralelnu obradu I/O operacija, poboljšavajući performanse, posebno na sustavima s više CPU jezgri.
Naime tradicionalni *virtio-scsi* koristi jedan I/O red za sve zahtjeve, što može postati usko grlo u scenarijima s velikim opterećenjem. S druge strane ***Multi-Queue Virtio-SCSI*** koristi više redova (*queues*) za paralelnu obradu I/O operacija, gdje se svaki red može dodijeliti različitoj CPU jezgri.
***Multi-Queue Virtio-SCSI*** omogućava:

- ***Paralelnu obradu*** – više redova omogućava paralelnu obradu I/O operacija na više CPU jezgri, čime se povećava ukupna propusnost.
- ***Smanjenje latencije*** – zbog raspodjele zahtjeva na više redova, latencija I/O operacija je manja jer ne postoji zagušenje u jednom redu.
- ***Bolje iskorištavanje resursa*** – više redova smanjuje opterećenje pojedinih CPU jezgri i omogućava bolju skalabilnost na sustavima s više CPU jezgri.

Za omogućavanje *multi-queue* podrške za *virtio-scsi* uređaje, koristi se opcija `num-queues` u definiciji diska kada se pokreće virtualno računalo. Ova opcija određuje broj redova koje će koristiti *virtio-scsi*. To primjerice izgleda ovako:

```
qemu-system-x86_64 -enable-kvm -smp 4 \
-drive file=/path/to/disk.img,if=none,id=disk0,format=qcow2,cache=none \
-device virtio-scsi-pci,id=scsi0,num-queues=4
```

← U ovom konkretnom slučaju `num-queues=4` određuje broj redova za paralelnu obradu I/O operacija.
Ovaj broj treba biti usklađen s brojem dostupnih CPU jezgri na virtualnom računalu.

## *Asinkrona metoda pristupa diskovnom podsustavu - Async I/O* (AIO)

Kada QEMU *VirtIO* obrađuje zahtjeve pristupa diskovima on može koristiti i različite načine asinkronog rada odnosno pristupa tzv. I/O diskovnom podsustavu. ***AIO*** se može koristiti samo za *Virtio SCSI* kontrolere. Ovdje imamo nekoliko mogućnosti rada:

- `aio=io_uring` – najmoderniji *višenîtni asinkroni ne blokirajući* način rada (koristi novi ***io_uring*** mehanizam).
- `aio=native` – nešto je sporiji od *io_uring*, a brži od *thread* zbog svoje specifične asinkrone metode rada. Međutim u slučajevima kada CPU postaje usko grlo i kada se hardverski nîz [*hardware queue buffer*] često zapuni, tada *thread* metoda rada pokazuje bolje rezultate.
- `aio=thread` – specifičan po tome što može koristiti više programskih nîti za raspodjelu poslova i bolju paralelizaciju obrade u odnosu ne nekorištenje *AIO* metode uopće ili u gore navedenim slučajevima zagušenja (kod *native* metode).

## Optimizacije upotrebe međumemorije prema diskovnom podsustavu

Dodatno je moguće optimizirati i upotrebu međumemorije prema *hipervizorovom* mehanizmu koji komunicira s diskovnim podsustavom. Moguće je odabrati nekoliko modela komunikacije prema diskovnom podsustavu *hipervizora (fizičkog računala)*:

- `Writethrough` – odnosno s posebnim načinom upotrebe međumemorije od *hipervizora* tj. od Linuxa ispod njega (***O_DSYNC***) kako bi se smanjila mogućnost gubitka podataka uslijed naglog nestanka struje ili nasilnog gašenja poslužitelja. Naime na razini *hipervizora* (*Host*) koristi se sâmo međumemorija za čitanje (*read cache*) dok je međumemorija za zapisivanje isključena. Dakle tijekom svakog zapisivanja se radi sinkronizacija na disk odnosno potvrđuje se snimanje podataka tek kad su oni primljeni prema diskovnom sustavu. Ako nikakva opcija nije dana QEMU/KVM, tada se podrazumijeva ova opcija. Ova opcija je dobra za integritet podataka i sporija za zapisivanje.

- `Writeback` – opcija je suprotnost prethodnoj i ona radi tako da se koristi „*Page cache*" međumemorija (fizičkog) računala pa se podaci snimaju asinkrono dakle potvrđuje se snimanje podataka već kada su oni zaprimljeni u „*Page cache*" međumemoriju, bez obzira kada će stvarno biti poslani na druge slojeve diskovnog podsustava (pr. niz/***queue*** za zapisivanje) i u konačnici snimljeni na površinu diska. Ova metoda je najbrža, ali potencijalno nesigurna u slučaju nestanka struje ili nasilnog gašenja računala. Nadalje virtualni disk kontroler (na virtualnom računalu) informiran je o ovoj međumemoriji, pa on i ima određenu kontrolu nad njom. Naime on je u mogućnosti poslati *hipervizoru* prema Linuxu ispod njega naredbu za sigurno pražnjenje ove memorije (tzv. *Flush*) prema potrebi, kako bi se podaci iz ove međumemoriji uredno snimili na površinu (fizičkog) stvarnog diska (opcija `cache=writeback`).

- `Directsync` – opcija je mješavina prethodnima, tako da ona na razini hipervizora (*Host*) ne koristi "*Page cache*" međumemoriju, dakle radi s metodama ***O_DSYNC*** ili ***O_DIRECT***. S druge strane za zapisivanje sa strane virtualnog računala se koristi „*writethrough*" metoda odnosno sâmo međumemorija za čitanje (opcija `cache=directsync`).

- `None` – opcija je ona koja izričito isključuje upotrebu „*Page cache*" međumemorije (fizičkog) računala pa se podaci snimaju na način da kada su pristigli ispod razine „*Page cache*" međumemorije (fizičkog) računala koja se zaobilazi, dakle u razinu memorije nîza za izvršavanje operacija zapisivanja (*write queue*), šalje se naznaka virtualnom disk kontroleru virtualnog računala da postoji „*writeback cache*". On tada po potrebi može zahtijevati i sinkrono zapisivanje, ali i ne mora. Ova opcija je solidne brzine i sigurnija je, a ima dobre performanse za zapisivanje.

Pogledajmo konfiguraciju odnosno opcije i parametre s kojima *hipervizor* pokreće virtualno računalo koje koristi *VirtIO SCSI* disk kontroler, na koji će se potom spajati *VirtIO SCSI* (virtualni) diskovi:

```
-device virtio-scsi-pci,id=scsihw0,bus=pci.0,addr=0x5
-device scsi-hd,bus=scsihw0.0,channel=0,scsi-id=0,lun=0,drive=drive-scsi0,id=scsi0
```

Sada slijede parametri i opcije s kojima definiramo disk (pr. ***QCOW2***) te metodu upotrebe disk međumemorije `Writeback`, a dodatno smo uključili i opcije:

- ***IO Thread*** **–** postavljanjem na najoptimalniju metodu (`io_uring`).
- ***Discard -*** uključivanjem DISCARD opcije (`discard=on`).
- ***Multi-Queue Virtio-SCSI*** s četiri kanala (`num-queues=4`).
- Te optimizaciju za detekciju zapisivanja nula prema diskovnom podsustavu (`detect-zeroes`).

Pogledajmo ogledni primjer ovakve konfiguracije

```
-drive file=/VIRTUALKE/vm-harddisk.qcow2,if=none,id=drive-
scsi0,cache=writeback,format=qcow2,aio=io_uring, discard=on,detect-zeroes=on
-device virtio-scsi-pci,id=scsi0,num-queues=4
```

Prvo moramo instalirati programe i alate potrebne za virtualizaciju, pri tome su nazivi paketa navedeni za ***RedHat/CentOS*** 7+:

```
yum -y install qemu-kvm qemu-img
```



Kada se ***IO Thread*** koristi s ***Async I/O*** **(AIO)** i s ***multi-queue*** *virtio-scsi* mogu se postići još bolje performanse, jer svaka nit može obrađivati zahtjeve s različitih (I/O) redova, smanjujući latenciju i povećavajući propusnost diskovnog podsustava!

**Izvori informacija: (877),(883),(884),(1414),(1415),(1416),(1417),(1418),(1763),(K-8), `man qemu-img`, `man partprobe`, `man fdisk`,`man e2fsck`,`man resize2fs`,`man growpart`,`man fstrim`, `man 5 fstab`,`man 5 mount`.**

### 27.1.2.4. Disk image formati

*Disk image* formati kod hipervizora odnose se na datoteke koje predstavljaju virtualne tvrde diskove korištene od strane virtualnih računala (VM-ova). Ti formati omogućuju pohranu svih podataka počevši od operativnog sustava i aplikacija unutar jednog ili više datotečnih sustava, a u konačnici simuliraju fizički disk za virtualno računalo.

Namjena *disk image* formata za virtualna računala jest sljedeća:
- ***Virtualna pohrana*** – *disk image* sadrži sve podatke koje bi fizički disk imao: operativni sustav, datoteke, konfiguracije, itd.
- ***Migracija i backup*** – budući da su *disk image* datoteke prijenosne, omogućuju lako premještanje virtualnih računala između različitih fizičkih poslužitelja (npr. tijekom migracije ili backup procesa).
- ***Snapshot podrška*** – neki formati podržavaju "*snapshote*", koji omogućuju brzo vraćanje na raniji trenutak u slučaju kvara ili potrebe za povratom stanja diska virtualnog računala.
- ***Performanse i skalabilnost*** – ovi formati također mogu utjecati na performanse virtualnog diska, ovisno o tome kako su optimizirani za određene scenarije korištenja (npr. kompresija, "*thin provisioning*", podrška za TRIM/DISCARD).

U ovom koraku kreirat ćemo direktorij `/VIRTUALKE` te ćemo unutar njega kreirati disk za virtualno računalo, veličine 10GB:

```
mkdir /VIRTUALKE
cd /VIRTUALKE
qemu-img create -f qcow2 vm-harddisk.qcow2 10G
```

Za potrebu kreiranja (tvrdog) diska za novo virtualno računalo koristili smo naredbu `qemu-img` kojoj smo s prekidačem `-f qcow2` rekli kako će format (virtualnog) diska odnosno slike diska (Engl. *Disk Image*) biti takozvani ***QCOW2***.
Virtualna računala preko QEMU emulatora koriste emulirani disk kontroler na koji se spaja disk koji zapravo na razini Linux *hipervizora* predstavlja posebna *disk image* datoteka.
***QEMU*** podržava više vrsta ovakvih *slika* diska *(disk image)* datoteka, od kojih svaka ima svoje prednosti i mane (navest ćemo njih samo nekoliko) s pripadajućom oznakom koju možemo koristiti:
- `raw` – ovo je čisti format diskovne slike koji radi najbrže, ali ne podržava snimanje stanja diska u vremenu (Engl. *Snapshot*) pa, ako nam je to potrebno, moramo se pouzdati u neke druge mehanizme (pr. na datotečni sustav na kojem je ona pohranjena, na *hipervizoru* to jest na fizičkom računalu). Ako je na datotečnom sustavu na kojem se nalazi ova datoteka podržano snimanje kao prorijeđene datoteke (Engl. *Sparse*), onda je i to moguće i s ovim formatom.



Vezano za ***Sparse*** značajku, pogledajte poglavlje***:***
**4.6. Prorijeđene (*Sparse*) datoteke.**

***Raw*** format također ne podržava (jednostavno) proširenje kapaciteta i druge napredne mogućnosti.

- `qcow2` – ovo je napredni format diskovne slike koji radi nezamjetno sporije od *RAW*, ali zato podržava veliki broj naprednih mogućnosti poput snimanja stanja u vremenu (*snapshot)* te proširivanja kapaciteta diska u radu i slično.
- `nbd` – ovo je posebni takozvani „*network block device*“ format.
- `vdi` – ovo je ***Oracle VirtualBox*** format za diskove (*disk image*).
- `vpc` – ovo je stariji (*VHD*) ***Microsoft Hyper-V*** format za diskove (*disk image*).
- `vhdx` – ovo je noviji ***Microsoft Hyper-V*** format za diskove (*disk image*).
- `vmdk` – ovo je ***VMware*** format za diskove (*disk image*).

**Vratimo se na kreiranje virtualnog diska za virtualno računalo.**
S naredbom `qemu-img` je moguće raditi i mnoge druge operacije na slikama diskova, poput konvertiranja iz jednog formata u drugi i slično, što ćemo sada pogledati na neovisnim primjerima:
**1.** Konvertirajmo naš *disk image* iz **QCOW2** u **RAW** format:

```
qemu-img convert -f qcow2 -O raw vm-harddisk.qcow2 vm-harddisk.img
```

**2.** Proširimo kapacitet našeg *disk image-a* u **QCOW2** formatu, s primjerice 10GB na 15GB (odnosno povećanje za +5GB):

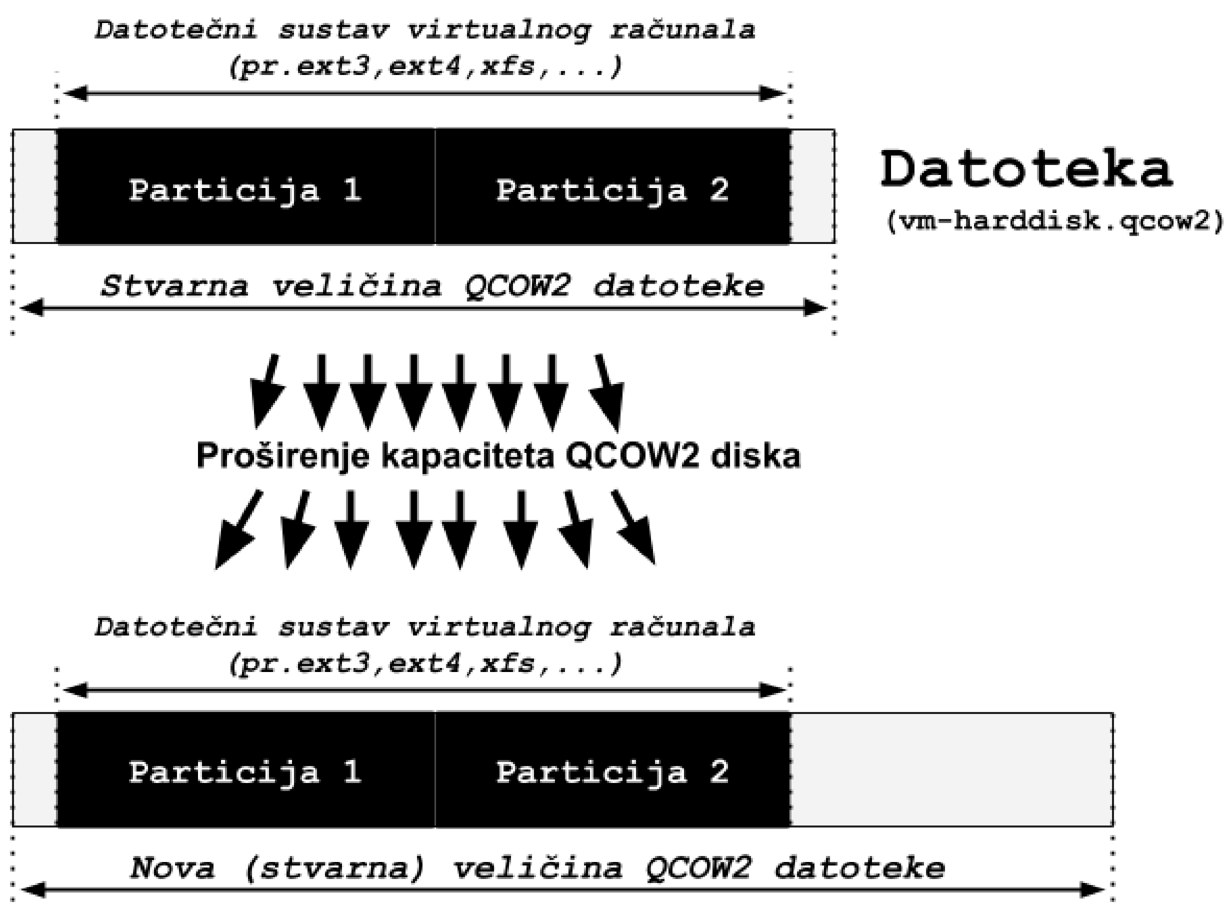
```
qemu-img resize vm-harddisk.qcow2 +5G
```

Nakon proširivanja kapaciteta diska odnosno slike diska tj. konkretno *QCOW2* datoteke, mi smo praktično povećali veličinu diska koji virtualno računalo vidi kao svoj tvrdi disk.

***Međutim datotečni sustav (etx3,ext4, xfs ili bilo koji drugi) virtualnog računala koje koristi taj disk je i dalje ostao na pozicijama unutar granica od 10GB. To znači da je ove granice sada potrebno nekako proširiti odnosno prilagoditi na novo stanje!***

Pogledajte logičku shemu ovakvog proširenja diska, bez spoznaje datotečnog sustava virtualnog računala o tome, kako je vidljivo na slici 256.

Slika 256: QCOW2 disk slika: prije i poslije proširenja odnosno povećanja kapaciteta




Kako bismo uopće napravili ovu promjenu kapaciteta, virtualno računalo mora biti ugašeno. Nakon ove promijene, kada ponovno pokrenemo virtualno računalo, na njemu ćemo morati naložiti datotečnom sustavu da se proširi do granica novog kapaciteta. Međutim ne zaboravimo da prvo moramo povećati (proširiti) particiju na tom disku, a tek onda proširiti datotečni sustav na njoj. U našem konkretnom slučaju morali bi prvo proširiti particiju (**Particija 2**) te potom na njoj proširiti njen datotečni sustav.

Ako nam je to sistemski disk, tada bi taj disk obično morali *spojiti* na neko drugo virtualno računalo te na njemu pokrenuti proširenje particije, a tek potom kapaciteta datotečnog sustava, jer to nije izvedivo da *živom* sistemskom disku.
Potom, tako prošireni datotečni sustav i disk možemo *odspojiti* i ponovno vratiti izvornom virtualnom računalu.

Ako to ipak nije bio sistemski već dodatni (drugi) disk, tada ovu operaciju možemo napraviti na živo, unutar virtualnog računala na koje je ovaj disk: (primjerice: `vm-harddisk.qcow2`) i spojen.

Pogledajmo kako to možemo napraviti, ako je particija koju proširujemo oznake: `/dev/vda2`. Naime, ako koristimo klasični *VirtIO* disk (znan i kao ***virtio-blk***) za virtualno računalo, on se neće vidjeti kao `/dev/sda` ili `/dev/sdb` disk već kao `/dev/vda` disk. Ako ipak koristimo najnoviju inačicu *VirtIO* odnosno takozvani ***VirtIO SCSI***, tada ćemo vidjeti normalne diskove poput `/dev/sda`.

U primjeru ćemo koristiti klasični *VirtIO* te ***VDA*** diskove.

Prvo pogledajmo stanje particija ovog diska prije povećanja (skratili smo ispis na samo ovaj disk):

```
lsblk /dev/vda

NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda      252:0    0 10G  0 disk
├─vda1   252:1    0  5G  0 part /DATA1
└─vda2   252:2    0  5G  0 part /DATA2
```

Sada pogledajmo novo stanje, s povećanim diskom (nakon što smo mu proširili kapacitet) za +5GB (koristimo naredbu `lsblk`).

```
lsblk /dev/vda
```

```
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda      252:0    0  15G  0 disk
├─vda1   252:1    0   5G  0 part /DATA1
└─vda2   252:2    0   5G  0 part /DATA2
```

Vidimo da je disk (`vda`) sada veličine 15GB (`15G`), ali su particije ostale iste veličine; po 5GB svaka (`5G` i `5G`).

U slučaju ako se stanje nakon promjene nije osvježilo, možete pokrenuti naredbu `partprobe` na sljedeći način:

```
partprobe /dev/vda
```

Pošto u našem slučaju imamo montirane obje particije kao `/DATA1` i kao `/DATA2` stoga ih prvo demontirajmo (`umount`):

```
cd /
umount /DATA1
umount /DATA2
```

Sada možemo proširiti našu particiju `/dev/vda2` na sljedeći način (to je moguće izvesti i s drugim naredbama).
Naime kako bismo što više pojednostavili ovaj proces instalirat ćemo jednu novu naredbu, baš za ovu namjenu.

```
yum -y install cloud-utils-growpart
```

U ovom paketu dobit ćemo novu naredbu `growpart` s kojom ćemo promijeniti drugu (**2**) particiju na `/dev/vda` disku.

```
growpart /dev/vda 2
```

```
CHANGED: partition=2 start=10485760 old: size=10485760 end=20971520 new:
size=20971487 end=31457247
```

Sada vidimo kako je druga particija (`/dev/vda2`) povećana do kraja diska, što će nam potvrditi i sljedeća naredba:

```
lsblk /dev/vda
```

```
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda      252:16   0  15G  0 disk
├─vda1   252:17   0   5G  0 part
└─vda2   252:18   0  10G  0 part
```

Ovdje vidimo da je particija `/dev/vda2` sada veličine `10GB`.

A sada kada smo proširili particije moramo proširiti i datotečni sustav. Međutim prvo ćemo pokrenuti program koji će napraviti provjeru ispravnosti (integriteta) datotečnog sustava. Mi koristimo ***ext4*** datotečni sustav pa koristimo naredbu `e2fsck` ovako:

```
e2fsck -f /dev/vda2
```

U našem slučaju, koristili smo ***ext4*** datotečni sustav pa ćemo za proširenje koristiti naredbu `resize2fs` na sljedeći način:

```
resize2fs /dev/vda2
```

**Međutim, da ste imali *XFS* datotečni sustav, ovo proširenje bi napravili sa slijedećom naredbom:**

```
xfs_growfs /dev/vda2
```

Kada se ovaj proces završi, imat ćemo uredno proširen datotečni sustav na particiji `/dev/vda2`.
Stoga ga ponovno montirajmo kako je bio montiran i prije ove operacije proširivanja kapaciteta:

```
mount /dev/vda2 /DATA2
```

Zatim pogledajmo njegovu veličinu s naredbom `df` na sljedeći način:

```
df -h /DATA2
```

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda2       9.8G   23M  9.2G   1% /DATA2
```

Vidimo kako je datotečni sustav ***ext4*** stvarno narastao na 10GB odnosno zapravo malo manje, zbog sâmog formata (9.8GB).

***Razina hipervizora***
Više informacija o slici diska (*disk image*) možemo dobiti sa sljedećom naredbom (izvršavamo ju na *hipervizoru*):

```
qemu-img info vm-harddisk.qcow2
```

Dodatne informacije na ovoj razini možete vidjeti s programom za particioniranje diska `fdisk`, ako ga pokrenete ovako:

```
fdisk -l vm-harddisk.qcow2
```

**Izvori informacija: (877),(883),(884),(1414),(1415),(1416),(1417),(1418),(K-8), `man qemu-img`, `man partprobe`, `man fdisk`,`man e2fsck`,`man resize2fs`,`man growpart`,`man fstrim`,`man 5 fstab`,`man 5 mount`.**

### 27.1.3. Optimizacije

O ovoj većoj cjelini upoznat ćemo se s mnogim optimizacijama vezanim za virtualizaciju.

#### 27.1.3.1. CPU i RAM

Vezano za *virtualizirane* procesore (CPU-ove) važno je znati kako hipervizor virtualizira funkcionalnosti fizičkog procesora odnosno njegove funkcionalnosti koje vidimo kao CPU zastavice (engl. *CPU flags*). Dakle funkcionalnosti (značajke) fizičkog procesora se prevode u one koje može koristiti naše virtualno računalo odnosno njegov procesor (CPU).



*Centralni procesor (CPU) virtualnog računala se naziva i **vCPU** ili jednostavno virtualni CPU.*

Međutim ovdje u procesu translacije: fizički ←→ virtualni procesor zapravo postoje dvije mogućnosti:

- Virtualizacija CPU-a (što je standardno), a što znači kako se funkcionalnosti fizičkog procesora (CPU) prevode u one koje predstavlja virtualni procesor. Drugim riječima one CPU zastavice koje ima naš fizički procesor se translatiraju u CPU zastavice *virtualiziranog* procesora koji virtualno računalo jedino i vidi. To možemo vidjeti pokretanjem sljedeće naredbe; prvo na fizičkom računalu (hipervizoru), a potom na virtualnom:
  ```
  grep -m1 -i flags /proc/cpuinfo
  ```
  Vrste *virtualiziranih* procesora, koje *hipervizor* daje virtualnom računalu, mogu biti:
  - Standardni *virtualizirani* procesori poput: `qemu32`, `qemu64` ili malo stariji (kompatibilniji i s prilično starim hardverom): `kvm32` ili `kvm64`. Pri tome se odabir definira s opcijom `-cpu`. Primjerice: `-cpu kvm64`.
  - *Virtualizirani* procesori za koje želimo da imaju samo željeni niz funkcionalnosti koje imaju točno određeni fizički procesori poput: **Intel**: *486,CoreDuo,Core2Duo,Haswell,…* ili **AMD**: *Athlon,Phenom,Opteron,…* .
    Što je korisno, ako testiramo ponašanje nekog softvera ovisnog o točnom modelu procesora.
- Druga kategorija je propuštanje svih funkcionalnosti fizičkog procesora (svih CPU zastavica) te svih informacija o procesoru (Engl. *CPU type, model, stepping,…*) prema *virtualiziranom* CPU-u. Ovakav rad se ponekada preferira, ako želimo maksimalne performanse virtualnog računala, bez obzira na potencijalne sigurnosne rizike. Dodatni problem ovakvog rada je i nemogućnost žive migracije virtualnog računala na drugo fizičko računalo, ako se njegov fizički procesor imalo razlikuje, što je moguće za očekivati u ***klasterima za virtualizaciju*** kod kojih nikada ne znamo na kojem fizičkom hardveru će naše virtualno računalo biti pokrenuto. Ova vrsta procesora se označava kao ***host*** jer se odnosi na fizički procesor *hipervizora* odnosno *hosta (fizičkog računala - hipervizora)*. Postiže se **QEMU** opcijom: `-cpu host`.

Razlike u performansama između recimo: ***QEMU64***, ***KVM64*** i ***HOST*** postoje, a vidljive su sâmo za ekstremno opterećene sustave (procesore) i na strani su prvo: ***HOST***, pa ***QEMU64*** te potom ***KVM64*** kao sâmo malo sporijeg.
Prema potrebi je moguće virtualiziranom procesoru dodjeljivati odnosno uključiti i CPU zastavice (funkcionalnosti) koje naš fizički CPU podržava, a mi ih želimo imati dostupne unutar našeg virtualnog CPU-a, koji ih trenutno nema. Dodatne zastavice uključujemo dodavanjem opcija `+` na postojeći CPU. Tako bi recimo za CPU `kvm64` zastavica ***SEP*** koja ubrzava inicijalizaciju kernela i zastavica ***AES*** koja ubrzava sve operacije vezane za *AES* algoritam bile dodane ovako: `-cpu kvm64,+sep,+aes`. Isto tako je moguće i oduzimati zastavice, također nizanjem istih, odvojenih sa znakom `,` ali uz oznaku `-` ispred zastavice.

---

Zatim imamo mogućnost da virtualni procesor (po CPU jezgri) ne koristi svaki takt fizičkog procesora, već sâmo određeni postotak, što u konačnici znači kako će fizički procesor *hipervizora* biti malo rasterećen, a virtualni procesor malo sporiji.
To se može koristiti kada se radi takozvani ***overprovisioning*** odnosno kada na fizičkom računalu *hipervizora* kreiramo veći broj virtualnih računala s više virtualnih CPU jezgri nego ih imamo na fizičkom stroju, u nadi kako neće sve biti u potpunosti i maksimalno opterećene cijelo vrijeme, što i inače *hipervizor* uredno odrađuje do neke granice. Međutim s ovom opcijom još više možemo rasteretiti fizički procesor od zadataka koje odrađuju virtualni procesori, naravno usporavajući upravo njih.

---

Nadalje moderni *hipervizori* kao što je **KVM,** imaju podršku i za ***NUMA*** arhitekturu što je vrlo važno na *NUMA* sustavima (koji se uglavnom koriste na poslužiteljima). Stoga kako bi se pristup RAM memoriji preko *hipervizora* što bolje prilagodio *NUMA* sustavima. Ovu funkcionalnost uključujemo s opcijom: `-numa node` iza koje slijede parametri ovisno o *NUMA* arhitekturi i broju CPU jezgri koje dodjeljujemo virtualnom računalu. S ovom opcijom uključenom na *NUMA* sustavima, postižu se poprilično velike razlike u brzini pristupanja RAM memoriji fizičkog računala odnosno hipervizora.



Za *NUMA* arhitekturu pogledajte poglavlje:
**10.7.2.2. NUMA.**

I na kraju, potrebno je naznačiti koliko CPU jezgri želimo dodijeliti našem virtualnom računalu, što se postiže s osnovnim prekidačem (u primjeru dodjeljujemo dvije CPU jezgre): `-smp 2`, a nakon toga možemo dodavati i detaljne opcije, pa bi to moglo izgledati ovako: `-smp 2,sockets=1,cores=2,maxcpus=2` (što znači 2 CPU jezgre s jednog *NUMA* CPU-a).



Za druge optimizacije pogledajte i poglavlje:
**26.4.4.2. Upotreba hardverski ubrzanih kripto funkcija.**

## CPU Pinning (CPU affinity)

*CPU pinning*, ponekad nazivan i *CPU affinity* je tehnika koja omogućava dodjeljivanje točno definiranih CPU jezgri određenim virtualnim računalima (ili procesima) unutar hipervizora.
U virtualizaciji svaka CPU jezgra fizičkog procesora (CPU) standardno može biti dodijeljena bilo kojem virtualnom računalu. *CPU pinning* omogućuje hipervizoru kontrolu izvršavanja zadataka (u ovom kontekstu virtualnih računala), sprječavajući zadano ponašanje hipervizora da dinamički premješta zadatke između različitih CPU jezgri, što može uzrokovati gubitak performansi zbog stalnog premještanja izvršavanja između jezgri procesora, a dodatno povećava i latenciju.

Naime, kada *CPU pinning* nije konfiguriran, *hipervizor* dinamički dodjeljuje i raspodjeljuje CPU jezgre prema potrebi.
To u praksi radi identično kao i za bilo koju drugu aplikaciju; kernel odnosno *task scheduler* komponenta konstantno preraspodjeljuje izvršavanje procesa (programa) na neku drugu jezgru procesora, brinući se tako da sve CPU jezgre budu jednoliko opterećene. Taj mehanizam prebacivanja na izvršavanje drugog zadatka (programa) se zove *context switching* i on nije idealan za vršno opterećene sustave, a pogotovo ne za vršno opterećene hipervizore.



Za detalje o ***context switching*** mehanizmu, pogledajte poglavlje:
**9.3.1. Context switching.**

S *CPU pinningom*, svako virtualno računalo ili aplikacija koristi unaprijed definirane CPU jezgre. *CPU pinning* to jest CPU *affinity* može se postavljati na dva načina: pomoću naredbi `taskset` ili `numactl` ili unutar konfiguracije *hipervizora*.



Za detalje o ***CPU afinitetu*** pogledajte poglavlja:
**10.7.2.1.1. CPU afinitet (CPU Affinity).**
**10.7.2.2.2. NUMA CPU i RAM afinitet.**

Vezano za konfiguraciju unutar Linux hipervizora (KVM/QEMU) dostupne su opcije poput:
`-numa node,cpus=0-1,nodeid=0 -numa node,cpus=2-3,nodeid=1`.
U ovom primjeru, prva dva CPU-a (0 i 1) su vezana na *NUMA node 0*, a druga dva (2 i 3) na *NUMA node 1*.
Na ovakav način (s *CPU pinningom*) smo osigurali predvidljiv odziv i snagu procesiranja virtualnih računala.

**Međutim, to nije sve**
Naime u NUMA sustavima (s dva ili više fizičkih procesora), važno je pripaziti i na to da, ako primjerice imamo mrežno intenzivan rad virtualnog računala, da se CPU jezgre koje želimo rezervirati i dodijeliti određenom virtualnom računalu, budu što bliže sabirnici na koju je spojena primjerice mrežna kartica koja je u ovom primjeru vršno opterećena.



Za detalje o ***NUMA I/O*** topologiji, pogledajte poglavlje:
**10.7.2.2.1. NUMA I/O topologija.**

Nadalje, i cijelo fizičko računalo odnosno *hipervizor*, mora biti detaljnije optimizirano, ako govorimo o zadovoljavanju visokih performansi. Optimizacije na koje trebamo paziti su po pitanju procesora odnosno procesorskih jezgri koje se preporuča rezervirati za zadatke kernela na *hipervizoru*. Odnosno za izolaciju željenih CPU jezgri samo za kernel *hipervizora*, koje tada postaju nedostupne za virtualna računala. To se radi zbog osiguravanja dovoljne procesorske snage za zadatke na najnižoj razini *hipervizora:* pr. za *task scheduler*, pristup mreži i diskovnom podsustavu, signalima prekida [*IRQ*] i slično.



Za detalje o izolaciji CPU jezgri pogledajte poglavlje:
**10.7.2.4. Izolacija jezgri procesora.**

Važno je znati i to da je po pitanju primjerice obrade mrežnih paketa, za brzine od 10Gbps na više, važno i još puno toga. Odnosno da je prvo na strani *hipervizora* potrebno napraviti mnoge optimizacije.



Za detalje o optimizaciji mreže za brzine veće od 10Gbps pogledajte poglavlje:
**25.5.6. Optimizacije mrežnih sučelja propusnosti veće od 10Gbps.**



Po pitanju analiza potrebnih za optimizacije mreže, pogledajte i poglavlje:
**25.3. Statistike, analiza i praćenje mrežnih paketa.**

---

## RAM memorija

Što se dodjeljivanja **RAM** memorije virtualnom računalu tiče, ona se dodaje sâmo s pomoću prekidača `-m` iza kojeg slijedi količina RAM memorije u megabajtima koju želim dodijeliti virtualnom računalu: recimo za 2GB bi to bilo `-m 2048`.
Na ovaj način smo fiksno našem virtualnom računalu dodijelili 2GB RAM memorije za upotrebu. Naravno o njemu ovisi koliko od te memorije će iskoristiti u datom trenutku. Sa strane hipervizora je važno da je on tu količinu memorije rezervirao za upotrebu, za naše virtualno računalo.

---

## *Balloon*

Dodatno postoji i mogućnosti dinamičkog dodjeljivanja memorije upotrebom ***Balloon*** funkcionalnosti koja je dostupna upotrebom posebnog ***VirtIO*** mehanizma, koji memoriju koju dodjeljujemo virtualnom računalu dinamički alocira do željene maksimalne vrijednosti. Međutim za ovu funkcionalnost je potrebno posebno emulirano *PCI* sučelje (*VirtIO balloon*) čiju konfiguraciju i upotrebu ćemo kratko opisati. Naime ovo posebno *PCI* sučelje (`virtio-balloon-pci`) je potrebno pokrenuti tijekom pokretanja virtualnog računala, što se postiže kao opcija poput:

```
-device virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x3
```

Tek kada je ovo *VirtIO PCI balloon* sučelje uredno konfigurirano i pokrenuto, tada možemo koristiti i ***Balloon*** funkcionalnost.
Linux kao virtualno računalo već ima upravljački program za njega (`virtio_pci`) te prateći kernel modul imena:
`virtio_balloon` (ako nam je *hipervizor* Linux *KVM+QEMU*), koji zapravo odrađuje ovu funkcionalnost odnosno komunicira s *hipervizorom* vezano za dostupnu i iskorištenu RAM memoriju unutar virtualnog računala. Za druge *hipervizore* potrebni su drugi upravljački programi; primjerice za ***VMware*** je to: `vmw_balloon` dok je za ***Hyper-V*** to: `hv_balloon`.
Za ***Windows*** OS kao virtualno računalo, potrebno je *balloon* instalirati s *VirtIO ISO* slike***** navedene u cjelini ***Qemu guest agent***, a osim upravljačkog programa potrebno je pokrenuti i **Balloon** servis za *Windowse*. Nakon instalacije i konfiguracije ***balloon*** upravljačkog programa i servisa na virtualnom računalu, kada virtualno računalo oslobodi određenu količinu RAM memorije, to će uredno javiti *hipervizoru*, pa će ju i on moći uredno osloboditi, što bez ove funkcionalnosti nije uvijek (ili uopće) moguće. Dakle s njenom upotrebom se može drastično uštedjeti RAM memorija na *hipervizoru* odnosno fizičkom računalu koje pokreće virtualna računala. Mana je zanemarivo sporije alociranje memorije za virtualno računalo: kada mu zatreba više memorije, a koju je prethodno oslobodio. To u normalnoj upotrebi nije primjetno.



*Čak i kada se koristi fiksna veličina RAM memorije, omogućite upotrebu **balloon** uređaja, jer on pruža korisne informacije **hipervizoru**, kao što je podatak koliko RAM memorije virtualno računalo stvarno koristi (i treba).*

## *Ivshmem*

U slučajevima kada imamo potrebu za ekstremno velikom brzinom dijeljenja podataka između računala *hipervizora* i virtualnih računala, kao i među virtualnim računalima, moguće je koristiti poseban RAM disk naziva ***Ivshmem*** *(opcija je:* `-device ivshmem-plain,memdev=HMB,...`*)*. *Ivshmem* je zapravo virtualni *PCI* uređaj u virtualiziranom OS-u koji emulira ***KVM/QEMU***. On postavlja Linuxovu dijeljenu memoriju (**SHM**) između VM-a i *Hipervizora*.

Naime *ivshmem* omogućuje komunikaciju *VM→Hipervizor* i obratno, bez kopiranja sadržaja memorije (*zero-copy*), što je vrlo učinkovito s obzirom na propusnost i kašnjenje, jer ne postoji interni međuspremnik podataka. On se također može koristiti za komunikaciju između virtualnih računala korištenjem ***SHM*** memorije *hipervizora* kao posrednika. *Ivshmem* je implementiran preslikavanjem virtualne memorije *Ivshmem* PCI uređaja na *SHM* memoriju *hipervizora*. Odnosno to je moguće jer ***QEMU*** emulira memoriju kao strukturu podataka unutar sebe i zato što više QEMU-a može međusobno komunicirati unutar *hipervizora*.
Prema nekim mjerenjima[(926)], upotrebom ovog posebnog RAM diska postižu se bolje performanse u komunikaciji između virtualnih računala, u odnosu na mrežnu komunikaciju, od tri do deset puta (300% - 1.000%).

**Izvori informacija: (927),(928).**

---

### 27.1.3.2. Chipset, BIOS i grafička kartica (VGA)

S obzirom na to kako je potrebno virtualizirati (*emulirati*) i cijeli čipset (*chipset)* računala koje virtualiziramo, i ovdje (ovisno o inačici) trenutno imamo dvije mogućnosti:

- Emulacija klasičnog ***i440FX*** *čipseta* sa ***PIIX3/4*** djelom zaduženima za rad s ***PCI*** i ***ISA*** sabirnicama, ***IDE*** i ***USB*** kontrolerom te drugom periferijom. Ovo je obično standardna opcija, a definira se kao: `-machine pc`.
- Emulacija novijeg i modernijeg ***Q35***[886] *chipseta* s integriranim *IOMMU* te sa ***ICH9*** dijelom zaduženima za rad s novijom ***PCI*** express sabirnicom kao i ***ISA*** sabirnicom, ali sada s AHCI ***SATA*** kontrolerom te ***USB*** kontrolerom i drugom periferijom. Ovo je obično napredna opcija, a definira se kao: `-machine q35`.

Stoga ako tražite novije i naprednije funkcionalnosti za vaše virtualno računalo, ***Q35*** bi mogao biti dobar odabir.
Međutim prije svega, odnosno prema logici na početku, potrebno je emulirati i cijeli ***BIOS*** računala (matičnu ploču).

***QEMU*** standardno emulira ***PC BIOS*** iz ***Seabios*** projekta, a ***VGA BIOS*** iz ***Bochs*** projekta (za primarnu grafičku karticu) te to nije potrebno eksplicitno navoditi pri pokretanju virtualnog računala jer se podrazumijeva.
U slučaju kada moramo koristiti UEFI funkcionalnost, poput upotrebe virtualizacije u kojoj radimo takozvani ***PCI passthrough*** fizičke grafičke kartice prema onoj unutar virtualnog računala, tada moramo koristiti drugačiji *BIOS* koji ima podršku za *UEFI*.

Za ovaj ***BIOS*** moramo imati i binarnu sliku (*image*). Pogledajmo okvirni primjer ovakve konfiguracije virtualnog računala:

```
-drive if=pflash,unit=0,format=raw,readonly,file=/usr/share/pve-edk2-firmware//OVMF_CODE.fd
-drive if=pflash,unit=1,format=raw,id=drive-efidisk0,file=/tmp/100-ovmf.fd
```

Vezano za grafičke kartice, moguće je koristiti onu s osnovnom funkcionalnosti, ali i neke malo naprednije grafičke kartice:

- Emulirane (virtualizirane) poput: standardne (`vga`) koju podržavaju svi operativni sustavi. Ona je poglavito *VMware* kompatibilna (`vmware-svga`)
- *Paravirtualizirane* poput ***SPICE QXL*** (`qxl-vga`) koja koristi SPICE protokol i njegove funkcionalnosti.
- Upotrebu fizičke grafičke kartice kao prolazne kroz PCI sabirnicu (***PCI passthrough***) prema virtualnom računalu.

Grafička kartica se u *hipervizoru* (***QEMU/KVM***) uobičajeno definira ovako (primjer za ***Spice QXL***):

```
-device qxl-vga,id=vga,max_outputs=4,bus=pci.0,addr=0x2
```



Za točan naziv opcija i parametara uvijek pogledajte zadnju inačicu ***QEMU*** dokumentacije:
**https://www.qemu.org/docs/master/**

### 27.1.3.3. IOMMU

Hardver koji zahtjeva više performanse kao što su grafičke i mrežne kartice, koristi **DMA** metodu za izravan pristup memoriji. Pošto u virtualnom okruženju sve memorijske adrese ponovno preslikava sustav unutar virtualnog računala, to uzrokuje probleme u **DMA** načinu rada. ***IOMMU*** upravlja ovim ponovnim preslikavanjem memorije, dopuštajući da se izvorni upravljački programi uređaja koriste unutar operativnog sustava virtualnog računala, što može drastično ubrzati njihov rad.
Ipak, da bi to sve radilo, pogotovo s podrškom u hardveru, potrebna je podrška za *IOMMU* u procesoru (pr. za *Intel* procesore je to tehnologija *VT-d*, dok je za AMD to *AMD-Vi* ), na matičnoj ploči (BIOS/UEFI), ali i u kernelu *hipervizora*. Nadalje potrebna je podrška i od strane sâmog *hipervizora* (KVM/QEMU), ali i unutar virtualnog računala.



Za rad ***IOMMU*** i optimizacije u virtualizaciji, pogledajte poglavlje:
**10.7.1.4. Ulazno-izlazni memorijski kontroler (IOMMU).**

### 27.1.3.4. Ugniježđena virtualizacija (*nested virtualization*)

Ugniježđena virtualizacija odnosi se na virtualizaciju koja se izvodi unutar već virtualiziranog okruženja.
Drugim riječima, to je mogućnost pokretanja hipervizora unutar virtualnog računala (VM), koje i samo već radi na hipervizoru. Dakle možete imati *hipervizor* instaliran direktno na hardver, koji pokreće gostujući *hipervizor* kao virtualno računalo (VM), koje potom može pokretati vlastita virtualna računala, ali još uvijek koristeći hardversko ubrzanje s fizičkog računala (procesora). Ovo očito dodaje opterećenje ugniježđenoj okolini, ali bi moglo biti korisno u nekim slučajevima. Primjerice omogućuje da testirate (ili naučite) kako upravljati *hipervizorima* prije stvarne implementacije ili upotrebu raznih emulatora i simulatora koji moraju raditi u ovakvom načinu rada, kako bi zadržali prihvatljivu brzinu izvršavanja.
Kako bi imao najbržu moguću izvedbu, blizu izvorne hardverske brzine, svaki *hipervizor* trebao bi imati pristup stvarnim hardverskim značajkama koje su važne za virtualizaciju, takozvanim *'hardverski potpomognutim virtualizacijskim proširenjima*'. Naime u ugniježđenoj virtualizaciji, gostujući *hipervizor* također bi trebao imati pristup hardverski potpomognutim virtualizacijskim proširenjima, a to implicira da bi centralni *hipervizor* (instaliran na hardver) trebao izložiti ta proširenja (instrukcije procesora vezane za virtualizaciju) svojim virtualnim strojevima.

U načelu sve to može raditi i bez tih proširenja, ali s lošim performansama što nije opcija za produkcijska okruženja. Izlaganje tih instrukcija procesora zahtijeva procesore koji to podržavaju. Pri tome su potrebni kernel 3.10 ili noviji te hardverska podrška za:

- **Virtualizaciju**: ***Intel VT-x*** [`vmx` flag] ili ***AMD-V*** [`svm` flag].
- **Second Level Address Translation** (**SLAT**): **Intel EPT** [`vmx` flag] ili **AMD RVI** [`rvi` flag].

Centralni kernel modul koji nam daje podršku za hardverske instrukcije za virtualizaciju je kernel modul imena `kvm`.
Na njega se oslanja kernel modul specifičan za procesor:

- Za ***Intel*** CPU je to kernel modul imena `kvm_intel`.
- Za ***AMD*** CPU je to kernel modul imena `kvm-amd`.

Dakel za Intel procesore moramo imati pokrenuta dva kernel modula:

```
lsmod | grep ^kvm
kvm_intel              360448  0
kvm                   1019904  1 kvm_intel
```

Dakle moramo imati kernel module: `kvm` i `kvm_intel`.

Provjerimo je li podrška za ugniježđenu virtualizaciju podržana u trenutnim postavkama kernela:

```
cat /sys/module/kvm_intel/parameters/nested
Y
```

Ako imamo **Y** sve je u redu i ova podrška postoji.

U slučaju da nismo imali **Y**, to za:
Intel CPU možemo napraviti dodavanjem opcije za kernel modul (koja se aktivira nakon restarta računala):

```
echo "options kvm-intel nested=Y" > /etc/modprobe.d/kvm-intel.conf
```

Dok za AMD CPU možemo napraviti dodavanjem ovakve opcije za kernel modul (koja se aktivira nakon restarta računala)

```
echo "options kvm-amd nested=1" > /etc/modprobe.d/kvm-amd.conf
```

Zatim je potrebno na željenom hipervizoru, za svako pojedino virtualno računalo, kao vrstu CPU-a odabrati "*Host*", a to je QEMU/KVM opcija `--cpu host`. Nakon što se virtualno računalo pokrenulo, provjerimo imamo li ove instrukcije unutar virtualnog procesora:

```
egrep -o -m1 -i '(vmx|svm)' --color=always /proc/cpuinfo
vmx
```

Za Intel CPU ove instrukcije su vidljive kao `vmx`, a za AMD kao: `svm`.
Dakle ako imamo jednu od njih to znači da imamo ove instrukcije dostupne unutar virtualnog računala.



Za ugniježđenu virtualizaciju, pogledajte i poglavlje:
**12.3.2.3. Proces translacije adresa.**

### 27.1.3.5. QEMU Guest agent i ACPI

Servis ***qemu guest agent*** je servis koji je obavezno potrebno instalirati unutar virtualnog računala. On se koristi za razmjenu informacija između domaćina (*hosta* odnosno ***hipervizora***) te gosta odnosno virtualnog računala, za izvršavanje naredbi za posebne namijene unutar virtualnog računala. Naime u komunikaciji između *hipervizora* i virtualnog računala, poput primjerice slanja naredbe za gašenjem virtualnog računala, *hipervizor* virtualnom računalu obično šalje takozvani ***ACPI*** sistemski poziv za gašenjem koji, kada ga virtualno računalo primi, treba pokrenuti proceduru gašenja. Međutim ovisno o operativnom sustavu virtualnog računala i njegovim servisima, to ne mora uvijek raditi kako treba.
Sve ovisno o tome je li u sustavu virtualnog računala uopće instaliran ***ACPI*** servis i prihvaća li on uopće ***ACPI*** pozive.

Za *RedHat/CentOS* (unutar virtualnog računala) ovaj servis možemo instalirati na sljedeći način:

```
yum -y install acpid
```

Naravno servis je potrebno i pokrenuti i aktivirati (za ***RedHat/CentOS 7.x***+):

```
systemctl start acpid
systemctl enable acpid
```

Dodatno u slučaju kada *hipervizor* želi krenuti s procedurom izrade sigurnosne kopije (Engl. *Backup*) cijelog virtualnog računala, a pogotovo s metodama ***suspend*** ili ***snapshot***, u tom slučaju bi operativni sustav virtualnog računala to trebao nekako prepoznati te preko svojih servisa odmah pokrenuti proceduru postavljanja diskovnog sustava u takozvano stanje zamrzavanja.

Za to se primjerice u *Windows* OS-u brine servis imena ***VSS*** (engl. *Volume shadow copy service*) pomoću kojeg se osigurava da su svi podaci na disku virtualnog računala uredno zaključani te da tijekom procesa izrade sigurnosne kopije virtualnog računala to jest njenog diska, neće doći do neki promjena koje mogu dovesti do nekonzistentnosti podataka.

Za sve navedene potrebe, na strani *hipervizora* je potrebno aktivirati posebno ***VirtIO*** serijsko sučelje (`virtio-serial`) koji se emulira, pa je stoga za njega potrebno instalirati upravljačke programe unutar virtualnog računala koje ga treba koristiti.
Ako je virtualno računalo Linux, ovaj upravljački program je obično već dostupan te je potrebno samo instalirati *Qemu guest agent* servis te ga pokrenuti. Dakle prvo unutar virtualnog računala instalirajmo ovaj ***QEMU*** agent (servis).

```
yum install qemu-guest-agent
```

Za ***RedHat/CentOS*** 7+ ga sada i pokrenimo te ga aktivirajmo tijekom svakog pokretanja sustava virtualnog računala:

```
systemctl start qemu-guest-agent
systemctl enable qemu-guest-agent
```

Sada je na ***hipervizoru*** potrebno aktivirati *Qemu guest agent* funkcionalnost tako da se aktivira posebno *VirtIO* serijsko sučelje.
To se postiže sa sljedećim opcijama tijekom pokretanja virtualnog računala (*okvirni primjer*):

```
-device virtio-serial,id=qga0,bus=pci.0,addr=0x8
-device virtserialport,chardev=qga0,name=org.qemu.guest_agent.0
```

Dodatno je potrebno definirati i komunikacijski *socket* za ovo *VirtIO* serijsko sučelje (primjerice):

```
-chardev socket,path=/var/run/qemu-server/100.qga,server,nowait,id=qga0
```

Hipervizor od tog trenutka preko ovog *VirtIO* posebnog serijskog sučelja (porta) sada uredno može komunicirati sa ***Qemu guest agent*** servisom te dalje s operativnim sustavom virtualnog računala.
Ako je virtualno računalo ***Windows***, tada je potrebno kopirati *ISO* sliku (CD/DVD-ROM) ***VirtIO*** s adrese:
https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/stable-virtio/virtio-win.iso*****

Tu *ISO* sliku je potrebno ubaciti u virtualni CD/DVD-ROM uređaj i s njega instalirati ***VirtIO*** serijsko sučelje (***vioserial***) koje će se pojaviti. Potom je potrebno instalirati *Qemu guest agent* servis: ***qemu-ga-x64.msi*** (64-bitni) ili ***qemu-ga-x86.msi*** (32-bitni).

## *Vratimo se na priču o KVM i QEMU*

Naime ideja je da za *RedHat/CentOS* Linux ne pokrećemo ručno QEMU/KVM, već preko nekog sučelja poput ***libvirt***, a svi ovi programi (barem većina) pohranjeni su u poseban direktorij `/usr/libexec/` koji je namijenjen za posebne korisničke programe. Mi ćemo zaobići ovo pravilo jer želimo ručno pokrenuti virtualno računalo, pa ćemo dodati novu putanju u profil.

```
echo "export PATH=$PATH:/usr/libexec/" >> /etc/profile
export PATH=$PATH:/usr/libexec/
```

Od ovog trenutka imamo dostupne sve potrebne programe.

Sada pokrenimo naše virtualno računalo uz napomenu kako opcije i parametri ovise o točnoj inačici **QEMU** emulatora koji koristimo.
Dakle moguće je imati jednu od dvije moguće inačice programa:

- `qemu-kvm` ili
- `qemu-system-x86_64`

Za točne opcije i parametre (na inačici na kojoj radite), možete pokrenuti:

```
qemu-kvm -h
```

Mi ćemo koristiti `qemu-kvm` pa ćemo pokrenuti naše virtualno računalo uz sljedeće pretpostavke:

- Da smo kreirali *Bridge* mrežno sučelje `br0` (u prethodnim koracima) i to bez našeg kreiranja ***TAP*** sučelja.
- Da imamo tzv. ***IFup*** datoteku za inicijalizaciju virtualne mrežne kartice te da smo već kreirali ***QCOW2*** disk.

Datoteku ***IFup*** tj. „*Interface UP*“ nemamo, pa ćemo ju kreirati. Kreirajte datoteku imena `/etc/qemu-ifup` koja sadrži:

```
#!/bin/bash
switch=br0
if [ -n "$1" ];then
    ip tuntap add $1 mode tap user `whoami`
    ip link set $1 up
    sleep 0.5s
    ip link set $1 master $switch
    exit 0
else
    echo "Error: no interface specified"
    exit 1
fi
```

Ova datoteka to jest skripta tijekom pokretanja virtualnog računala automatski kreira ***TAP*** mrežno sučelje i dodaje ga u ***bridge*** (`br0`) pa stoga i ne moramo sâmi kreirati ***TAP*** mrežno sučelje.



Dakle pratimo sâmo korake **1,2,3** te **6** i **7** iz cjeline o kreiranju **TAP/Bridge** sučelja:
**27.1.2. Rad s virtualizacijom (KVM+QEMU).**

Sada promijenimo ovlasti na našu skriptu od gore, kako bi se mogla pokretati:
```
chmod a+x /etc/qemu-ifup
```

Kreirajmo i skriptu imena `/etc/qemu-ifdown` za gašenje *TAP* sučelje kada se virtualno računalo ugasi:
```
#!/bin/bash
switch=br0
if [ -n "$1" ];then
    ip link set dev $1 nomaster
    sleep 0.5s
    ip link set dev $1 down
    exit 0
else
    echo "Error: no interface specified"
    exit 1
fi
```

Sada i za nju promijenimo ovlasti na našu skriptu od gore, kako bi se mogla pokretati:
```
chmod a+x /etc/qemu-ifdown
```

Kreirajmo i direktorij (mapu):
```
mkdir /etc/qemu/
```

Te novu datoteku `bridge.conf` u njemu; vezano za *bridge* sučelje, koja će sadržavati dozvolu upotrebe svih *bridge* sučelja:
```
allow all
```

Sada promijenimo ovlasti na tu datoteku (kako bi svi imali prava čitanja, te dodatno postavimo *sticky bit*):
```
chmod a+sr bridge.conf
```

I sada možemo pokrenuti naše virtualno računalo sa sljedećim opcijama (razlomili smo opcije sa znakom `\`):
```
qemu-kvm -machine pc -accel kvm -smp 1 -boot c -m 512 \
-device virtio-scsi-pci,id=scsihw0,bus=pci.0,addr=0x5 \
-drive file=/VIRTUALKE/vm-harddisk.qcow2,if=none,id=drive-
scsi0,cache=writeback,format=qcow2 \
-device scsi-hd,bus=scsihw0.0,channel=0,scsi-id=0,lun=0,drive=drive-scsi0,id=scsi0 \
-nic tap,model=virtio-net-pci,ifname=tap0 \
-monitor telnet:127.0.0.1:55555,server,nowait \
-display vnc=127.0.0.1:0
```
Pogledajmo što smo sve ovdje definirali:

- Prvo smo rekli ***QEMU*** sustavu da se emulira standardni *chipset* (s: `-machine pc`). Potom govorimo ***QEMU*** sustavu da koristi ***KVM*** hardverski (CPU) ubrzanu virtualizaciju (s: `-accel kvm`).
- Slijede opcije i parametri koje smo već opisali.

Standardni prikaz ekrana virtualnog računala odnosno onoga na što je „spojena“ virtualna grafička kartica, se definira u opciji: `-display vnc=127.0.0.1:0`, a kod nas je definirano da se za tu namjenu koristi ***VNC*** protokol, na lokalnom računalu, na standardnom portu 5900 (oznaka :0). Dakle u našem slučaju se možemo s ***VNC*** klijentom spojiti na ovo fizičko računalo na navedeni port, kako bi dobili pristup „*ekranu*“ virtualnog računala. Da smo recimo htjeli emulirati ***QXL*** grafičku karticu unutar virtualnog računala, te omogućiti pristup na nju pomoću SPICE klijenta, tada bi umjesto navedene opcije, koristili opciju:
```
-vga qxl -spice port=5900,addr=127.0.0.1,disable-ticketing
```

Kada se iz menadžera za virtualizaciju (*OpenStack*, *ProxmoxVE*, *VMware*,…) spajate na takozvanu konzolu odnosno ekran ili monitor virtualnog računala, zapravo vas on preusmjerava upravo na ovaj uređaj.

## *QEMU monitor*

*QEMU* monitor se koristi za slanje posebnih naredbi *QEMU* emulatoru, prema željenom virtualnom računalu kao primjerice za gašenje ili zamrzavanje rada virtualnog računala, dodavanja CD/DVD-ROM diska u virtualni CD/DVD-ROM uređaj, ali i cijelog nîza drugih opcija, kao i provjere stanja rada virtualnog računala. Opcija s kojom definiramo servis s kojim se možemo spojiti na *QEMU* monitor našeg virtualnog računala je ovdje konkretno:
`-monitor telnet:127.0.0.1:55555,server,nowait`.
To znači kako se samo s fizičkog računala možemo preko telnet protokola spojiti na *QEMU* monitor.

Pa probajmo to i napraviti. Međutim prvo instalirajmo telnet klijent:
```
yum -y install telnet
```

Sada se možemo spojiti na ***QEMU*** monitor konzolu na port 55555 (kako smo definirali):
```
telnet 127.0.0.0 55555
```

Pojavit će nam se *qemu* konzola [`(qemu)`] te možemo za naše virtualno računalo pokretati željene radnje, poput (nabrojat ćemo ih samo nekolicinu, od nekoliko stotina dostupnih):

- `savevm` - za izradu kopije sustava virtualnog računala u vremenu (dakle u ovom trenutku) (Engl. *Snapshot*).
- `system_powerdown` - za slanje signala za uredno gašenje virtualnog računala.
- `system_reset` - za resetiranje virtualnog računala.
- `system_wakeup` - za vraćanje virtualnog računala iz stanja pauziranja u normalni način rada.
- `info` – ovaj nîz naredbi daje nam informacije o svim dijelovima *QEMU* emulatora (navest ćemo ih samo nekoliko):

- `network` – daje nam informacije o emuliranim mrežnim karticama koje koristi ovo virtualno računalo.
- `pci` – daje nam informacije o emuliranim *PCI* sabirnicama.
- `status` – daje nam informacije o stanju virtualnog računala (pokrenuto/zaustavljeno).
- `vnc` – daje nam informacije o *VNC* poslužitelju (emuliranom ekranu/monitoru virtualnog računala).

Bez obzira želite li ručno kreirati virtualno računalo kao što smo mi radili ili vas zanimaju mogućnosti i optimizacije ***QEMU*** i ***KVM*** sustava, ovo poglavlje bi vam trebalo pomoći u tome. Svi primjeri opcija i prekidača koje smo ovdje pokazivali ovise o točnoj inačici ***QEMU*** i ***KVM*** programa i sustava. Međutim sve navedene mogućnosti postoje, i na vama je da ih razumijete i koristite želite li iole optimalan sustav virtualizacije.



Za poveznicu između mrežnog stôga Linuxa, **TAP** i ***bridge*** sučelja, pogledajte poster **P5**.



U virtualizaciji se za izolaciju mreže virtualnih računala mogu koristiti:
**VLAN-ovi**, kako je objašnjeno u poglavlju:
**20.6.2. VLANovi odnosno virtualne lokalne mreže.**

**VXLAN-ovi**, kako je objašnjeno u poglavlju:
**26.8.2. Druga mrežna sučelja (*MACVLAN,IPVLAN,VXLAN,MACVTAP/IPVTAP*).**

Za izolaciju mreže se koristi i **VRF** (*Virtual routing and forwarding*) tehnologija.
Za više detalja o njemu pogledajte poglavlja:
**25.7.9. Naredba ip** – cjelina **VRF (***Virtual routing and forwarding***).**
**25.6.8.1. *Routing policy* (*Policy based routing*).**



U slučaju kada od postojećeg virtualnog računala želite napraviti predložak (***klon*** i ***template***) obično prvo želite obrisati svu konfiguraciju sustava (mrežne IP parametre i sl.)

Naime ako na sustavu u vršnom korijenskom direktoriju (mapi) / postoji datoteka imena: `.unconfigured`
tada će sustav nakon prvog restarta pokrenuti proceduru ponovne konfiguracije sustava odnosno imena računala, mrežnih postavki i slično. Ako to želimo ručno napraviti, dovoljno je kreirati ovu datoteku:

```
touch /.unconfigured
```

Dakle ako ova datoteka postoji na sustavu sustav će odmah pokrenuti proceduru ponovne (re)konfiguracije sustava, kao da je računalo tek instalirano (bilo da se radi o fizičkom ili virtualnom računalu). Međutim, postoje i dodatni alati za tu namjenu.

**Za Red Hat/CentOS do v.7.x.**
Možemo instalirati sljedeći programski paket:

```
yum -y initial-setup
```

I potom pokrenuti naredbu za čišćenje konfiguracije sustava:

```
sys-unconfig
```

Nakon toga virtualno računalo je očišćeno od konfiguracije sustava (mrežne i druge postavke sustava).

**Za Red Hat/CentOS od v.8.x.**
Možemo instalirati sljedeći programski paket:

```
yum -y cloud-init
```

I potom pokrenuti naredbu za čišćenje konfiguracije sustava:

```
virt-sysprep
```

Nakon toga virtualno računalo je očišćeno od konfiguracije sustava (mrežne i druge postavke sustava).

### 27.1.3.6. Diskovni podsustav

Za optimizacije diskovnog podsustava na najnižoj razini pripazite na odabir sustava za pohranu diskovnih slika virtualnih računala:

- Na hardverskom RAID polju diskova (na **hardverskim RAID kontrolerima**), uz optimizacije istog.
- Na softverskom RAID polju diskova (***ZFS***, **LVM** ili ***LVM Thin*** *i drugi*), oz optimizacije istih.
- Na mrežnim sustavim za pohranu (***NAS/SAN*** sustavi).

Optimizirajte diskovni podsustav:

- Pravilnim odabirom disk kontrolera (pr. *virtio-scsi-pci*) i *disk image* formata (*RAW, QCOW2, ...*).
- Korištenjem TRIM i DISCARD opcije i SSD emulacije prema potrebi.
- Upotrebom ***I/O Thread i SSD emulacije***.
- Upotrebom ***Multi-queue virtio-scsi***.
- Upotrebom ***Async I/O*** (AIO).
- Optimizacijom upotrebe međumemorije prema diskovnom podsustavu.



Za navedene optimizacije na razini diskovnog podsustava *hipervizora* i virtualnih računala, pogledajte poglavlje: **27.1.2.3. Disk kontroleri.**

Pogledajte i detalje o pravilnom odabiru **I/O scheduler** algoritma:
**14.1. Diskovni ulazno/izlazni sustav (*I/O*)** → za *I/O scheduler* (*načine rada*)

Pogledajte i druge cjeline:
**13.8.2.2. *Swappiness* i druge opcije.**
**14.1.3. Optimizacija *Filesystem* sloja** → za ***sysctl*** varijablu **`fs.aio-max-nr`**.
**14.1.4. Optimizacija *Generic Block Layera* i I/O *Schedulera*** → za odabir I/O *schedulera* (***/sys/block/sdXY/queue/scheduler***).

**Izvori informacija: (1763),(1764) i gore navedena poglavlja.**

### 27.1.3.7. Sistemski sat virtualnih računala

Na fizičkim računalima tijekom pokretanja računala i učitavanja operativnog sustava, operativni sustav od hardverskog sata (RTC) dohvaća datum i točno vrijeme te ih postavlja na sustav (Linux) kao sistemski sat. Nakon toga sustav (Linux) se više ne oslanja na hardverski (RTC) sat već sâm računa vrijeme oslanjajući se na hardverske signale prekida (IRQ) i druge hardverske komponente računala koji se koriste kao izvori (mjerači) vremena.
Pogledajmo neke od najčešće dostupnih:

- ***i8254 – PIT*** - *programmable interrupt timer* – on je i izvor (mjerač) vremena.
- ***RTC*** – *realtime clock* (fizički sat).
- ***APIC*** - *Advanced Programmable Interrupt Controller* – on je i izvor (mjerač) vremena..
- ***HPET*** - *High Precision Event Timer* – obično dostupan kako registar unutar CPU-a.
- ***TSC*** - *Time Stamp Counter* - obično dostupan kako registar unutar CPU-a.
- ***ACPI Power Management Timer*** – on je izvor (mjerač) vremena.



Za detalje o radu hardverskog i sistemskog sata fizičkog računala (*hipervizora*), pogledajte poglavlje: **10.4.1. O hardverskom i sistemskom satu.**

Međutim kod virtualizacije takav pristup je problematičan jer se virtualno računalo ne može osloniti na hardverske signale prekida ili neke druge mjeraće vremena koje mu može proslijediti *hipervizor*. Stoga jer je pitanje primjerice može li *hipervizor* isporučiti svim virtualnim računalima u svakom trenutku signale prekida na koje se oslanja sistemski sat. Stoga što signali prekida u virtualnim računalima nisu stvarni hardverski signali prekida (IRQ).
Umjesto toga, *hipervizor* ih dodjeljuje i ubacuje u virtualna računala. Nadalje *hipervizor* (fizičko računalo) možda u određenom trenutku pokreće drugo virtualno računalo ili neki drugi proces (program). Zbog svega toga precizno izračunavanje vremena koje obično zahtijeva signale prekida ili oslanjanje ne neke druge (fizičke) brojače vremena možda nije uvijek moguće.
Važno je razumjeti da virtualna računala bez točnog mjerenja vremena mogu imati problema s mrežnim aplikacijama i drugim programima (servisima), budući da valjanost sesije, migracija i druge mrežne aktivnosti ovise o vremenskim oznakama da bi ostale točne.

Mehanizam sinkronizacije vremena virtualnog stroja je sljedeća. Naime, prema zadanim postavkama, virtualno računalo sinkronizira svoje vrijeme s *hipervizorom* na sljedeći način:

- Kada se sustav učita, čita se točno vrijeme iz emuliranog sata stvarnog vremena (**RTC**).
- Nakon što se mreža inicijalizira i NTP protokol pokrene, on automatski sinkronizira sat. Nakon toga, tijekom normalnog rada virtualnog računala, NTP vrši podešavanja i korekcije sata.
- Međutim u situacijama kada se virtualno računalo pauzira (obično kada se rade ***snapshot*** ili ***backup***) te zatim nastavi s radom nakon pauze ili procesa migracije virtualnog računala, *hipervizor* bi trebao izdati naredbu za sinkronizaciju sata virtualnog računala na određenu vrijednost. Ova sinkronizacija radi samo ako je **QEMU guest agent** instaliran na virtualnom računalu te *hipervizor* podržava tu značajku. Vrijednost s kojom se sat virtualnog računala sinkronizira obično je vrijednost sata domaćina (hipervizora).

Pogledajmo dostupne izvore mjerenja vremena koje imamo na našem virtualnom računalu:

```
cat /sys/devices/system/clocksource/clocksource0/available_clocksource
kvm-clock tsc hpet acpi_pm
```

Dakle vidimo da ih je dostupno nekoliko:

- `tsc` - Označava dostupan **TSC** (*Time Stamp Counter [CPU flag* `tsc`*]*). Njegova dostupnost se može provjeriti sa:

```
cat /proc/cpuinfo | grep constant_tsc
```

- `hpet` - Označava dostupan **HPET** (*High Precision Event Timer*).
- `acpi_pm` - Označava dostupnu ***APCI*** *power management* komponentu za mjerenje vremena.
- `kvm-clock` - Označava dostupan poseban pseudo KVM uređaj koji predstavlja (pseudo) hardverski sat.

Za virtualno računalo, provjerimo koji pseudo ili simulirani izvor sata ono trenutno koristi:

```
cat /sys/devices/system/clocksource/clocksource0/current_clocksource
kvm-clock
```

Ovo (`kvm-clock`) je najbolji mogući izbor za bilo koje virtualno računalo jer koristi pseudo mjerač vremena preko KVM sustava za virtualizaciju. On inicijalno dohvaća sat (vrijeme) *hipervizora* te može kompenzirati vremenska odstupanja periodičkim provjerama vremena *hipervizora* u odnosu na vrijeme virtualnog računala.
U slučaju kada `kvm-clock` nije odabran, njega možemo postaviti sa sljedećom naredbom:

```
echo "kvm-clock" > /sys/devices/system/clocksource/clocksource0/current_clocksource
```

**U svakom slučaju koristite i NTP servis, po mogućnosti** `chrony`**, jer brže može reagirati na odstupanja sata (vremena).**

### 27.1.3.8. Hot plugging disk, CPU, NIC, RAM

Pojam **Hot plug** odnosi se na sposobnost računala, da mu se u radu dodaju PCI kartice. Primjerice mrežna kartica, disk kontroler s diskovima i slično, RAM memorija ili čak i procesor (CPU). *Hot plug* se obično odnosi na ovakvu mogućnost u virtualiziranom okruženju, iako je to moguće i na posebnim poslužiteljskim matičnim pločama (i to samo posebno građenim za tu namjenu). Ako se radi o hardverskoj podršci, jasno je da je tu potrebna sklopovska (elektronička) i softverska podrška.
Kod hardverske podrške je važno znati da je iz Linuxa moguće isključiti određeni hardver (CPU, karticu na sabirnici i sl.), naravno ako to hardver podržava. Primjerice hardver *PCI express* sabirnice i kartica u načelu to podržavaju (barem u teoriji).

Mi ćemo ovdje govoriti o virtualiziranom okruženju odnosno mogućnosti *hipervizora*, da tijekom rada virtualnog računala, tom istom virtualnom računalu isključi ili nadoda komponente poput procesorskih jezgri, RAM memorije, diskova, mrežnih ili drugih kartica. Međutim da bi to sve radilo, potrebna je podrška u kernelu hipervizora te u samom *hipervizoru* (pr. **KVM+QEMU**) te u konačnici i u kernelu virtualnog računala. Što se KVM+QEMU komponenti *hipervizpra* tiče to je standardno već podržano, samo je obično potrebno uključiti tu sposobnost.

Linux kao *hipervizor* na razini kernela i virtualno računalo po tom pitanju moraju imati i takvu podršku u kernelu: ili već kompiliranu u kernel ili dostupnu kao kernel module.
Bazični kernel moduli (ako to već nije kompilirano u samom kernelu) su: `acpiphp` i `pci_hotplug`.

Preostale važne opcije tijekom kompiliranja kernela koje moraju biti dostupne su:

- `CONFIG_HOTPLUG_CPU` – omogućuje podršku za dinamičko dodavanje i uklanjanje CPU-a tijekom rada sustava.
- `CONFIG_MEMORY_HOTPLUG` – omogućuje podršku za dinamičko dodavanje i uklanjanje RAM-a.
- `CONFIG_HOTPLUG_PCI_PCIE` – omogućuje *hotplug* podršku za ***PCIe*** hardver.
- `CONFIG_ACPI_HOTPLUG_CPU` – omogućuje **ACPI** podršku za dinamičko dodavanje i uklanjanje CPU-a.
- `CONFIG_SCSI_VIRTIO` –ova opcija omogućuje *hotplug* podršku za ***VirtIO SCSI*** uređaje.
- `CONFIG_VIRTIO_PCI_HOTPLUG` – omogućuje *hotplug* podršku za ***VirtIO PCI*** uređaje (pr. mrežne kartice i diskove).

Ako želite provjeriti kako je vaš kernel kompiliran po pitanju ovih opcija, to možete učiniti na sljedeći način:

```
grep HOTPLUG /boot/config-`uname -r`
```

**Izvori informacija: (886),(887),(1396),(1397),(1414),(1484),(1487),(1583),(1584),(1585),(1586),(1587),(K-8),** `man qemu-kvm`**,**`man sys-unconfig`**,**`man virt-sysprep`**. Pogledajte i poster P5.**

# 27.2. Linux kontejneri

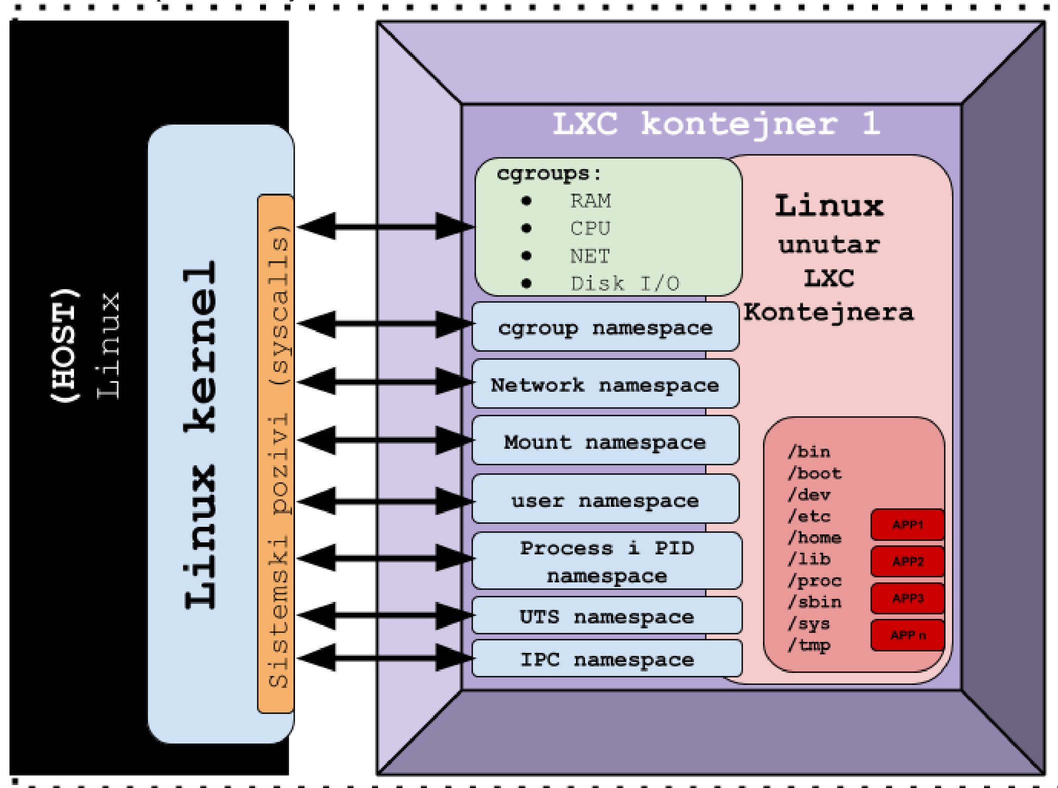
Linux kontejneri su jedan oblik virtualizacije, ali na razini operativnog sustava Linux, unutar kojeg se pokreću izolirani Linux operativni sustavi odnosno Linux kontejneri. Ovdje se ne radi o virtualnim računalima već o izoliranom pokretanju novih instanci linuxa. Dakle praktično o pokretanju linuxa unutar postojećeg linuxa. U svakom slučaju nema cijelog nîza slojeva virtualizacije i emulacije virtualiziranog hardvera. Rad linux kontejnera se svodi na pristup već postojećem (instaliranom) linuxu, odnosno njegovom kernelu i sistemskim pozivima, kroz sâmo jedan kontrolni sloj, koji dodaje potrebna ograničenja odnosno prava pristupa na dostupne resurse. U ovom sloju se ograničava pristup na dostupan broj jezgri centralnog procesora (primjerice dozvoljavamo sâmo dvije CPU jezgre za pojedini linux kontejner), količinu RAM memorije, diskovnog prostora i slično, kako je vidljivo na slici dolje (257**)**. Za ovu potrebu počevši od Linux kernela v.3.8. do danas, kreirane su posebne kategorije za izolaciju resursa sustava. Od ovih kategorija izoliranih prostora linuxa, svaka je zadužena za jedno područje izolacije. Trenutno je u kernelima 4.15.x dostupno njih sedam (7+**1**). Pri tome svaki od navedenih izoliranih prostora ima odgovarajuću *oznaku* u linuxovom virtualnom datotečnom sustavu (***VFS***). Dakle ovdje imamo sljedeće metode izolacije:

***Cgroups*** (`cgroup`) - ovo zapravo nije izolirani prostor (*namespace*) sâm po sebi već se koristi za kontrolne grupe: za izolaciju pripadnosti kontrolnih grupa pokrenutih procesa koji koriste neki *izolirani prostor*. Dakle ove kontrolne grupe su zadužene za dodjeljivanje, ograničavanje i izolaciju resursa računala, poput: CPU, RAM, disk I/O i mreže.

Slijedi lista izoliranih prostora Linuxa, koji su redom [(2019.g.)]:

1. ***Cgroup namespace*** (`cgroup`) - za skrivanje identiteta kontrolne grupe (*cgroup*) čiji je proces član.
2. ***IPC namespace*** (`ipc`) - za procese i njihove mehanizme komunikacije (*IPC — Interprocess Communication*).
3. ***PID namespace*** (`pid`) - za izolaciju procesa i ***PID*** brojeva (***PID****=Process ID)* pokrenutih procesa.
4. ***Network namespace*** (`net`) - za izolaciju mreže i mrežnih sustava.
5. ***User namespace*** (`user`) - za izolaciju korisnika i njihovog rada (prema: ***UID*** i ***GID*** brojevima/identifikatorima).
6. ***Mount namespace*** (`mnt`) - zaduženih je za *montiranje (mount)* particija unutar izoliranog prostora; primjerice unutar ***LXC*** linux kontejnera.
7. ***UTS namespace*** (`uts`) - omogućava promjene *hostname* i *domainname* imena unutar izolacije, za različite procese.

**Slika 257. Linux *namespaces* i Linux kontejner.**






Vezano za izolirane prostore Linuxa (Engl. *Namespaces*) pogledajte poglavlje:
**9.4.5. Izolirani prostori Linuxa (*Linux namespaces*).**

Iz navedenog je vidljivo kako tehnologija izolacije mora raditi brže nego klasična virtualizacija, ali i istovremeno biti štedljivija prema dostupnim resursima. Jedan linux kontejner je za linux koji ga "opslužuje" samo novi nîz linux procesa (pokrenutih programa), ništa više. Ova tehnologija se koristi za razne Linux kontejnerske tehnologije, poput ***LXC*** Linux kontejnera ili ***Dockera*** koje su naizgled slične virtualizaciji, ali bez potrebe za virtualiziranjem odnosno emuliranjem svih komponenti računala.

Vezano za Linux kontejnere u primjeni, u pravilu imamo dva pristupa kod dizajna njihovog načina rada:

- Klasični „*punokrvni*" linux kontejner unutar kojega se nalazi potpuno funkcionalan operativni sustav Linux sa svim dostupnim programima i alatima te svim komponentama računala: mrežna kartica, disk i svim ostalim resursima računala. Predstavnici ovakvog rada su ***OpenVZ*** te noviji ***LXC*** Linux kontejneri.
- Mikro kontejner kod kojeg unutar definiranog radnog okruženja (*environmenta*) nemamo direktan pristup disku ili mreži linux kontejnera, već su svi ti resursi skriveni. Unutar ovakvog kontejnera se u pravilu pokreće sâmo jedna aplikacija (program) prema principu: jedan linux kontejner – jedna aplikacija. Najčešći predstavnik ovakvog rada je ***Docker*** tehnologija. Ovakav način rada se zove rad s mikro kontejnerima.

Kako bismo uopće mogli raditi s izoliranim prostorima linuxa (*namespace*) za to moramo imati i podršku u kernelu.

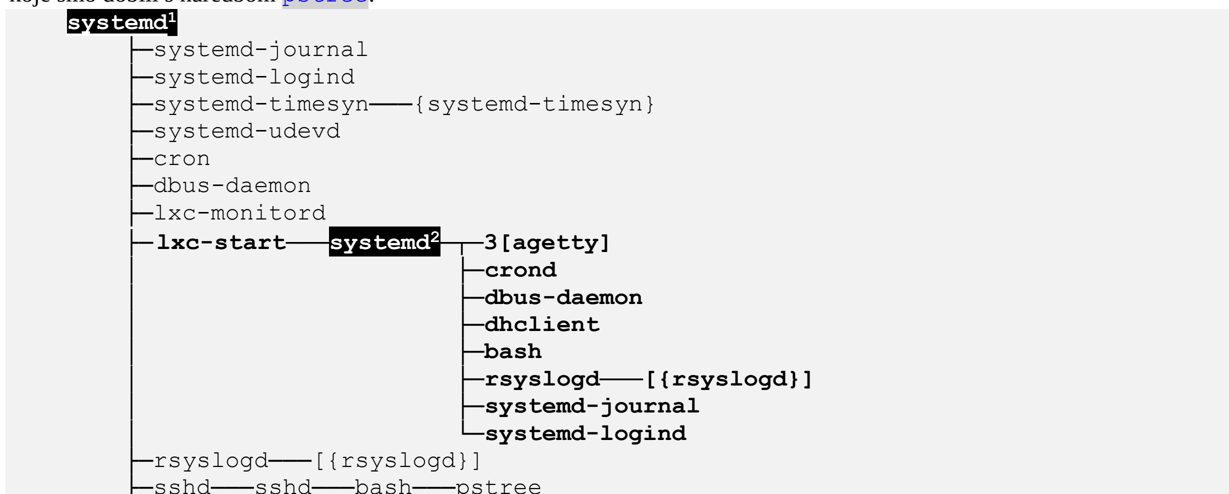
Ispis odnosno listu izoliranih prostora *linuxa* koji su kompilirani u vaš kernel možete vidjeti sa sljedećim nîzom naredbi:

```
egrep "^CONFIG_NAMESPACES|_NS=" /boot/config-$(uname -r) | grep -v CONN
```

Prva i najvažnija je opcija: `CONFIG_NAMESPACES=y`, a potom i druge koje slijede u ovom ispisu:

```
CONFIG_UTS_NS=y
CONFIG_TIME_NS=y
CONFIG_IPC_NS=y
CONFIG_USER_NS=y
CONFIG_PID_NS=y
CONFIG_NET_NS=y
```

Pogledajmo i kako izgleda stablo procesa jednog klasičnog ***LXC*** linux kontejnera, gledano s fizičkog računala i njegovog Linuxa, koje smo dobili s naredbom `pstree`:



```
systemd1
  ├─systemd-journal
  ├─systemd-logind
  ├─systemd-timesyn───{systemd-timesyn}
  ├─systemd-udevd
  ├─cron
  ├─dbus-daemon
  ├─lxc-monitord
  ├─lxc-start───systemd2─┬─3[agetty]
  │                      ├─crond
  │                      ├─dbus-daemon
  │                      ├─dhclient
  │                      ├─bash
  │                      ├─rsyslogd───[{rsyslogd}]
  │                      ├─systemd-journal
  │                      └─systemd-logind
  ├─rsyslogd───[{rsyslogd}]
  ├─sshd───sshd───bash───pstree
```

Vidimo kako je naše fizičko računalo kao prvi proces pokrenulo `systemd`[1] servis koji je i uvijek prvi servis (***PID*** broj **1**) koji pokreće sve ostale servise i programe (ispod njega). Potom dolje niže u stablu vidimo proces `lxc-start` koji pokreće novi `systemd`[2] servis, koji zatim pokreće druge servise i programe, logički ispod njega. Ovaj drugi `systemd`[2] je zapravo prvi proces (*systemd* ili *init*) unutar ***LXC*** linux kontejnera koji pokreće sve njegove servise i programe kao da se radi o zasebnom Linuxu, koji se i ponaša tako.

Dakle unutar Linux kontejnera imamo isto okruženje kao da se nalazimo unutar virtualnog računala na kojem možemo instalirati nove programe i servise, konfigurirati ih i raditi sve druge stvari kako da se radi o virtualnom ili običnom (fizičkom) računalu na koje je instaliran Linux.

Unutar ovakvog Linux kontejnera čak možemo imati instaliranu drugačiju distribuciju Linuxa od one koja se nalazi na fizičkom računalu na kojem smo ga pokrenuli.

**Izvori informacija: (316),(1054),(1055),(K-8),** `man pstree`, `man 7 cgroups`, `man 7 namespaces`, `man 7 network_namespaces`, `man 7 cgroup_namespaces`, `man 7 ipc_namespaces`, `man 7 mount_namespaces`, `man 7 pid_namespaces`, `man 7 time_namespaces`, `man 7 user_namespaces`, `man 7 uts_namespaces`.

### 27.2.1. Konfiguracija i upotreba Linux kontejnera

Konfiguracija i upotreba Linux kontejnera slijedi u ovoj cjelini.



Za detalje rada s izoliranim prostorima Linuxa pogledajte poglavlje:
***9.4.5. Izolirani prostori Linuxa (Linux namespaces).***
Mi ćemo se nadovezati na navedeno poglavlje kako se ne bi ponavljali.

Prvo instalirajmo potrebne programe za rad s kontrolnim grupama (*cgroups*) odnosno s *cgroup* mehanizmima[(746)] preko kojih se radi s izoliranim prostorima linuxa:

```
yum install -y libcgroup-tools
```

Dodajmo i ***EPEL*** repozitorij jer se na njemu nalaze dodatni programi i alati koji će nam biti potrebni:

```
yum -y install epel-release
```

Instalirajmo i dodatne potrebne softverske pakete:

```
yum -y install debootstrap perl libvirt lxc lxc-templates lxc-doc
```

I sada trebamo pokrenuti prvenstveno servis na koji ćemo se kasnije oslanjati u radu s LXC kontejnerima:

```
systemctl start lxc.service
```

Te eventualno možemo pokrenuti i ***libvirt*** servis, ako ga kasnije baš želimo koristiti, premda on nije nužan za LXC kontejnere na način kako ih mi sada želimo konfigurirati i koristiti.

Dakle slijedeća naredba nije nužna za ručni način rada.

```
systemctl start libvirtd
```

Potrebno je i trajno aktivirati servis zadužen za **LXC** Linux kontejnere:

```
systemctl enable lxc.service
```

I sada trebamo pokrenuti naredbu `lxc-checkconfig` koja će provjeriti postavke našeg sustava (skratili smo ispis):

```
lxc-checkconfig

Kernel configuration not found at /proc/config.gz; searching...
Kernel configuration found at /boot/config-5.6.4-1.el8.elrepo.x86_64
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled …
```

Ako je sve u redu s našim računalom i cijelim sustavom, vidljivo u svim ovim ispisima od gore, možemo nastaviti dalje.

Možemo krenuti s kreiranjem *LXC* kontejnera. Prvo ćemo ga kreirati povlačeći predloške (*template*) s repozitorija na internetu:

```
lxc-create -n CentOS -t download
```

Sada će se ova procedura spojiti na repozitorij i ponuditi nam cijeli nîz predložaka poput ovih (skratili smo ispis):

```
centos  6       amd64   default 20200416_07:08
centos  6       i386    default 20200416_07:08
centos  7       amd64   default 20200416_07:08
debian  buster  amd64   default 20200416_05:24
debian  jessie  amd64   default 20200416_05:24
```

I na kraju se trebamo odlučiti za jedan od njih, pa će nam se postaviti pitanja na koja ćemo mi ovako **odgovoriti**:

```
Distribution:
centos
Release:
7
Architecture:
amd64
```

Dakle upisali smo: `centos`, pa `7` te `amd64` (64 bitnu inačicu).
Potom sustav povlači ovaj predložak za **LXC** kontejner, konkretno za *CentOS* Linux inačice 7, 64 bitno, te kreira ***LXC***.



Lista **LXC** *templatea* (predložaka) se u ovom koraku povlači s adrese: **http://uk.images.linuxcontainers.org/**

Lista svih dostupnih predložaka na repozitoriju (*online*) možete vidjeti s naredbom:

```
lxc-create -t download -n NULL -- --list
```

Pošto je u prethodnom koraku kreiran *LXC* kontejner, listu kreiranih kontejnera na našem računalu možemo vidjeti sa:

```
lxc-ls

CentOS
```

Vidimo da imamo pripremljen novi *LXC* kontejner baziran na *CentOS* Linuxu koji se zove `CentOS` **(jer smo ga tako i nazvali).**

Svi novi *LXC* Linux kontejneri se kreiraju unutar vršnog direktorija `/var/lib/lxc/` odnosno naš *LXC* kontejner naziva **`CentOS`** se nalazi u direktoriju: `/var/lib/lxc/CentOS/`.

Ovdje se nalazi i njegova konfiguracijska datoteka koja je imena `config`. Pogledajmo što ona standardno sadrži:

```
# Distribution configuration
lxc.include = /usr/share/lxc/config/common.conf
lxc.arch = x86_64

# Container specific configuration
lxc.rootfs.path = dir:/var/lib/lxc/CentOS/rootfs
lxc.uts.name = CentOS

# Network configuration
lxc.net.0.type = veth
lxc.net.0.link = lxcbr0
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:9f:15:b7
```

Trenutno nas zanima mrežna konfiguracija, pošto još moramo kreirati posebno mrežno sučelje za (ovaj) naš *LXC* linux kontejner. Inicijalni dio konfiguracije mreže za *LXC* kontejnere koji se učitava prije nego se kreira ova datoteka je globalni *LXC* mrežni predložak zapisan u datoteci: `/etc/lxc/default.conf` prema kojem se i kreira unos u našoj datoteci.

Pogledajmo kako izgleda predložak (`/etc/lxc/default.conf`) uz naše opise uz svaki redak (→):

```
lxc.net.0.type = veth            #→ Definicija vrste virtualnog mrežnog sučelja na koje se LXC kontejner spaja.
lxc.net.0.link = lxcbr0          #→ Definicija bridge mrežnog sučelja na koje se LXC kontejner spaja.
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx    #→ Inicijalni dio opsega MAC adresa koji će se koristiti.
```

Standardno se za komunikaciju između fizičkog linux računala i mreže unutar *LXC* linux kontejnera koristi posebna ***VETH*** mreža, iako je moguće koristiti i druga posebna mrežna sučelja (primjerice: ***MACVLAN*** ili ***IPVLAN***). Mi ćemo ipak koristiti standardno ***VETH*** sučelje. Za tu potrebu ne moramo ručno kreirati ***VETH*** pâr mrežnih sučelja, već je potrebno sâmo promijeniti konfiguracijsku datoteku za naš *LXC* kontejner, a *LXC* servis će to odraditi za nas. Dakle otvorimo datoteku za naš *LXC* kontejner (`/var/lib/lxc/CentOS/config`) te u dijelu `# Network configuration` ostavimo samo sljedeće:

```
# Network configuration
lxc.net.0.type = veth
lxc.net.0.link = br0
lxc.net.0.name = eth0
lxc.net.0.veth.pair = veth1
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:9f:15:b7
```

Moguće je dodati i druge parametre *LXC* kontejneru poput njegove IP adrese te IP adrese *Default Gateway-a* i drugog:

```
lxc.net.0.ipv4.address = 192.168.1.111/24
lxc.net.0.ipv4.gateway = 192.168.1.1
```

**Ako ovdje postavimo IP parametre[(892)] oni će pregaziti one koje postavljamo unutar LXC kontejnera!.**

Ovime smo naložili sustavu da se koristi ***VETH*** vrsta sučelja (`veth`) te da *LXC* servis pri pokretanju ovog *LXC* kontejnera koristi `br0` *bridge* mrežno sučelje koje ćemo kreirati sâmi. Vezu na `br0` *bridge* mrežno sučelje će napraviti sustav, kreirajući `veth1` sučelje koje će biti povezano s `br0` *bridge* mrežnim sučeljem. Nadalje *LXC* servis će naložiti našem *LXC* kontejneru da na mrežu bude povezan preko svog lokalnog mrežnog sučelja koje će se zvati `eth0`, a koje će biti zapravo povezano s ***VETH*** sučeljem `veth1`. Sada kreirajmo naše *bridge* mrežno sučelje i spojimo ga s našom fizičkom mrežnom karticom.



***VETH*** mrežno sučelje se koristi za mrežu unutar *LXC* kontejnera jer se *VETH* sučelje kreira u pâru: jedan dio *VETH* pâra sučelja se veže za Linux *bridge* sučelje, a drugi dio se zapravo ubacuje u mrežni izolirani prostor (*network namespace*) unutar kojega je naš *LXC* linux kontejner. Naime sav mrežni promet koji uđe u prvo povezano *VETH* sučelje (u nazovimo *lijevu* stranu) mora izaći u drugo povezano *VETH* sučelje (desnu stranu) i obratno. Dakle *VETH* pâr (dva *VETH* povezana sučelja) se ponaša kao tunel kroz koji podaci prolaze: sve što uđe na jednoj strani mora izaći na drugoj.



Vezano za ***VETH*** mrežno sučelje pogledajte poglavlje:
**20.6.4. VETH - posebno mrežno sučelje.**

Vezano za mrežni most (*Bridge*) pogledajte poglavlje:
**20.6.1. Mrežni most (*bridge*) odnosno premosnik.**

Moguće je konfigurirati i mnoge druge parametre rada mreže unutar *LXC* linux kontejnera poput primjerice ***MTU*** vrijednosti, a to bi za *MTU* vrijednost od 9000 bilo otprilike ovako: `lxc.net.0.mtu = 9000`.



*U primjerima koji slijede, sve radimo na testom računalu, s direktnim pristupom na njega, jer ćemo u jednom trenutku izgubiti sve mrežne veze prema tom računalu, koje ćemo kasnije vratiti.*

***Prvo zapišimo naše IP parametre mrežne kartice `eth0`(ili koju već imate): IP adresu i masku mreže, te Default gateway.***

**1.** Kreirajmo *bridge* mrežno sučelje (`br0`) upotrebom naredbe `ip` na sljedeći način:
```
ip link add name br0 type bridge
```

**2.** Sada aktivirajmo ovo novo `br0` mrežno sučelje upotrebom naredbe `ip` na sljedeći način:
```
ip link set dev br0 up
```

**3.** Potom s *bridge* mrežnim sučeljem (`br0`) povezimo naše fizičko mrežno sučelje (`eth0`):
```
ip link set dev eth0 master br0
```

I pogledajmo kako sada izgleda naše *bridge* mrežno sučelje `br0`:
```
ip link show master br0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
         master br0 state UP
```
Vidimo da imamo vezu između naše fizičke mrežne kartice (`eth0`) preko *bridge* sučelja (`br0`).
Dakle vidimo vezu između mrežnog sučelja `br0` i mrežnog sučelja `eth0`. Odnosno vezu: `br0`←→`eth0`.

**4.** Pošto sada na `eth0` sučelju ne smijemo imati IP parametre, obrišimo ih (s mrežnog sučelja `eth0`), pa stoga pokrenimo:
```
ip addr flush dev eth0
```

**5.** Ako želimo konfigurirati IP adresu: 192.168.1.10 uz masku mreže (*netmask*): 255.255.255.0 za naše mrežno sučelje `br0`, tada ćemo morati napraviti sljedeće:
```
ip addr add 192.168.1.10/24 dev br0
```

Ne zaboravimo dodati podrazumijevani usmjerivač (*Default gateway)*, a u našem slučaju je to IP adresa 192.168.1.1:
```
ip route add default via 192.168.1.1
```

Svi navedeni primjeri konfiguriraju sva navedena sučelja i IP parametre privremeno, pa će nakon restarta računala sve biti vraćeno na staro stanje.

*Ne zaboravimo da se IP parametri definiraju na* ***bridge*** *sučelju, a ne više na fizičkoj mrežnoj kartici koja je dio bridge sučelja.*

**Sada možemo pokrenuti naš novi *LXC* Linux kontejner.**
Kontejner ćemo pokrenuti tako da definiramo i log datoteku za snimanje statusa i grešaka u radu našeg *LXC* kontejnera:
```
lxc-start -n CentOS --logfile /var/log/lxc.log
```

Naime *LXC* servis je sada u pozadini kreirao *VETH* pâr mrežnih sučelja te jedan dio pâra stavio u *bridge* sa `br0` sučeljem, a drugi dio pâra ubacio u izolirani mrežni prostor u kojem se nalazi naš *LXC* Linux kontejner.

Sada kada je naš *LXC* kontejner pokrenut, spojimo se direktno na njega, s naredbom `lxc-attach`, ovako:

```
lxc-attach -n CentOS
```

Potom možemo promijeniti *root* lozinku unutar našeg *LXC* kontejnera:

```
passwd
```

Te upišimo novu lozinku i potvrdimo ju.



Ne zaboravimo kako smo dobili minimalnu instalaciju ***CentOS* 7** Linuxa unutar *LXC* kontejnera bez cijelog nîza programa i alata, koje naravno, prema želji možemo i kasnije instalirati.

**Dakle na našem fizičkom računalu odnosno na Linuxu se tada kreiralo posebno VETH mrežno sučelje u pâru.**
Prvo vidimo vezu između našeg *bridge* sučelja `br0` i naše mrežne kartice `eth0`. Potom vidimo kako je jedan dio pâra VETH sučelja koji vidimo kao `veth1@if12` isto povezan s našim *bridge* sučeljem `br0`.

Pogledajmo kako izgledaju sva mrežna sučelja spojena u *bridge* sučelje `br0` (ovo radimo na fizičkom računalu):

```
ip link show master br0

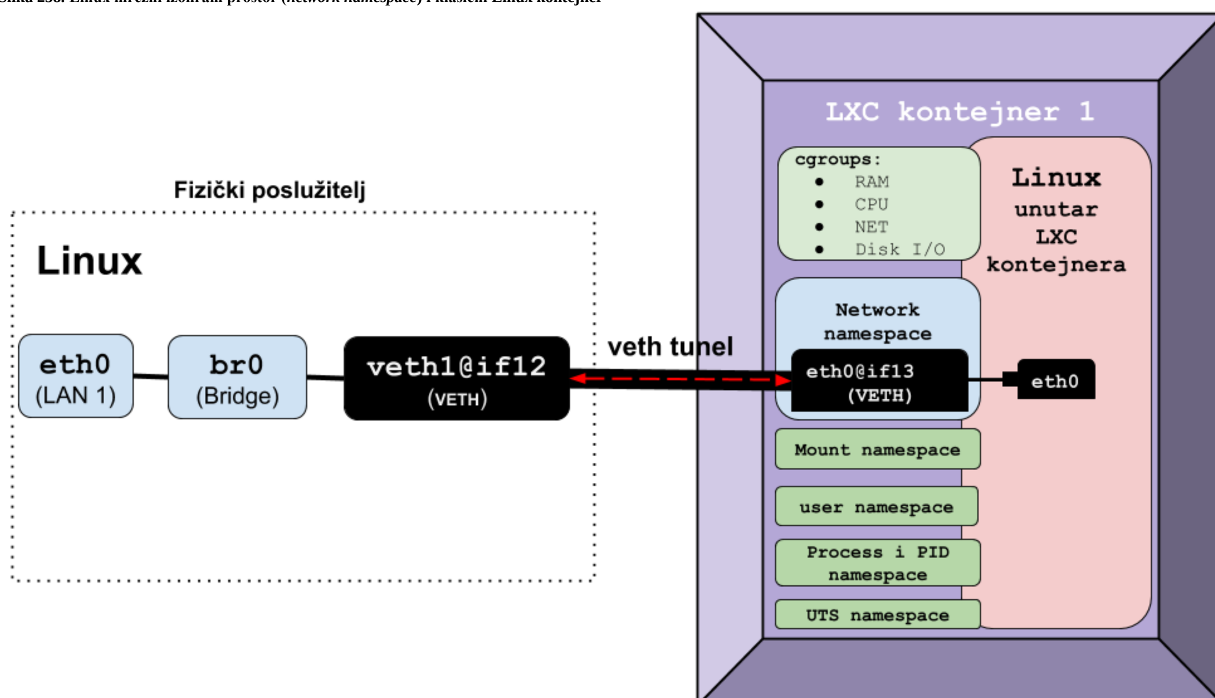
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master br0 state UP mode
DEFAULT group default qlen 1000
    link/ether 00:30:05:ba:dd:bb brd ff:ff:ff:ff:ff:ff
13: veth1@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0
state UP mode DEFAULT group default qlen 1000
    link/ether fe:9c:6d:d2:ae:01 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Dakle vezano za *VETH* pâr mrežnih sučelja: prvi dio pâra je ovo sučelje ovdje vidljivo kao `veth1@if12`. Naime ovo je zapravo mrežno sučelje naziva `veth1` koje ima posebnu vezu koju identificira oznaka `@if12`. Drugi dio pâra je unutar našeg ***LXC*** kontejnera odnosno unutar mrežnog izoliranog prostora koji koristi naš *LXC* linux kontejner. Važan podatak je kako je ovo mrežno sučelje `veth1` s vezom na `@if12` (`veth1@if12`) povezano s izoliranim mrežnim prostorom linuxa (*network namespace*) kako je vidljivo: `link-netnsid 0`. Indeks `@if12` koji je poveznica sa sučeljem `veth1` te indeksom drugog dijela *VETH* pâra je vidljiv sa sljedećom naredbom (dakle druga strana ovog *VETH* pâra prema *LXC* ima indeks broj **12**):

```
cat /sys/class/net/veth1/if{index,link}
13
12
```

Naime drugi dio pâra *VETH* mrežnog sučelja se nalazi unutar mrežnog izoliranog prostora broj **0** (`netnsid 0`**)**.
Pogledajmo kako to izgleda na logičkoj shemi na slici 258 (fokus ovdje je sâmo na mrežnom djelu komunikacije).

**Slika 258. Linux mrežni izolirani prostor (*network namespace*) i klasični Linux kontejner**



**Nastavljamo s radom unutar našeg LXC Linux kontejnera.**
Pogledajmo mrežna sučelja unutar našeg *LXC* Linux kontejnera (skratili smo ispis):

```
ip link show

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
6: eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode
DEFAULT
    link/ether 00:16:3e:9f:15:b7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Unutar *LXC* Linux kontejnera vidimo drugi dio pâra *VETH* sučelja, koji je zapravo unutar *LXC* linux kontejnera vidljiv kao `eth0` sučelje s posebnom vezom, koju označava indeks `@if13` pa je onda ovdje sučelje vidljivo s ovom naredbom kao `eth0@if13`. Ova poveznica indeksa unutra *LXC* Linux kontejnera je vidljiva sa sljedećom naredbom, pri čemu ćemo vidjeti vezu indeksa poveznice *VETH* sučelja: **12** (VETH pâr sučelja unutar LXC-a) → **13** (*VETH* na fizičkom Linuxu):

```
cat /sys/class/net/eth0/if{index,link}
12
13
```

Sučelje `eth0` pripada istom mrežnom izoliranom prostoru kao i prvi dio *VETH* pâra (sa fizičkog Linux računala) na što nam pokazuje i oznaka `link-netnsid 0`. Dakle i fizički Linux i *LXC* Linux kontejnera imaju mrežnu vezu prema `netnsid 0`.

Sada krenimo na konfiguraciju servisa pa prvo potencijalno instalirajmo SSH Servis te *net-tools* paket programa:

```
yum -y install openssh-server net-tools
```

Te aktivirajmo *SSH* servis (za udaljeni pristup na *LXC* Linux kontejner) i pokrenimo ga:

```
systemctl start sshd
systemctl enable sshd
```

Provjerimo koju IP adresu naš *LXC* Linux kontejner ima, te koja sve mrežna sučelja ima dostupna za programe i servise. Ako ista mrežna sučelja pogledamo s naredbom `ifconfig` vidimo kako nam je stvarno dostupno samo `eth0` mrežno sučelje. Naime naredba `ifconfig` radi na višoj aplikacijskoj razini od naredbe `ip` pa vidimo ono što je dostupno za uporabu i svim drugim programima i servisima (skratili smo ispis):

```
ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.58  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::216:3eff:fe9f:15b7  prefixlen 64  scopeid 0x20<link>
```

Konfiguracija ovog sučelja se radi normalno unutar datoteke (unutar *LXC* kontejnera): `/etc/sysconfig/network-scripts/ifcfg-eth0` pa je jasno da je ovo jedino mrežno sučelje koje konfiguriramo. Vidimo kako smo preko *DHCP* poslužitelja dobili IP adresu `192.168.1.58` na našem mrežnom sučelju imena: `eth0`.

Sada pogledajmo diskove (blok uređaje) unutar našeg *LXC* Linux kontejnera:

```
lsblk

NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
fd0      2:0    1    4K  0 disk
sda      8:0    0 74.6G  0 disk
├─sda2   8:2    0  6.9G  0 part [SWAP]
└─sda1   8:1    0 67.1G  0 part /
```

Te pogledajmo njihovo zauzeće:

```
df -h

Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        68G  5.3G   62G   8% /
none            492K  4.0K  488K   1% /dev
tmpfs           989M     0  989M   0% /dev/shm
tmpfs           989M  8.1M  981M   1% /run
tmpfs           989M     0  989M   0% /sys/fs/cgroup
```

Ovdje zapravo vidimo sve tvrde diskove od našeg fizičkog računala, jer nismo ništa ograničavali (što nije dobro za produkcijsku upotrebu). Moguće je postavljati diskovna i mnoga druga ograničenja prema svim resursima fizičkog računala.



***Moguće je kao LXC disk koristiti i neki od QEMU disk slika/formata poput RAW ili QCOW2, kao i koristiti disk ili particiju s fizičkog (Linux) računala.***

Ako bi koristili ***QEMU*** formate: ***QCOW2*** ili ***RAW***, kao slike diskova, metoda njihovog povezivanja je upotrebom posebnog *QEMU* programa koji i inače služi za rad s ovim diskovnim slikama, a zove se: `qemu-nbd`. Ovakvu konfiguraciju nećemo posebno objašnjavati jer je malo kompleksnija odnosno zahtjeva nekoliko koraka pripreme i dodatne konfiguracije.



Sve promjene radimo u konfiguracijskoj datoteci za naš LXC kontejner: `/var/lib/lxc/CentOS/config` *.

Opcije koje se tiču ***root*** datotečnog sustava LXC kontejnera definiraju se u opciji `lxc.rootfs.path` te eventualno u njenim dodatnim opcijama koje se definiraju unutar: `lxc.rootfs.options`.

**Moguće je definirati i dodatne točke montiranja unutar LXC linux kontejnera.**
Primjer u kojem bi dodali direktorij s fizičkog Linux računala `/podaci` unutar našeg *LXC* kontejnera kao `/DISK` direktorij:

```
lxc.mount.entry=/podaci DISK none bind 0 0
```

Važno je da se za direktorij (mapu) koji montiramo unutar *LXC* kontejnera na navodi početni znak /.

Sa sljedećim unosom bi dodali fizički blok uređaj (disk) `/dev/sdc` unutar *LXC* kontejnera:

```
lxc.mount.entry = /dev/sdc dev/sdc none bind,optional,create=file
```

I ovdje je važno da se za uređaj koji dodajemo unutar LXC kontejnera na navodi početni znak /.

S ovakvim unosom bi dodali **LVM2** particiju `/dev/mapper/lvm-home` unutar *LXC* kontejnera kao `/home` direktorij:

```
lxc.mount.entry = /dev/mapper/lvm-home home ext4 defaults 0 2
```

I ovdje je važno da se za uređaj koji dodajemo unutar *LXC* kontejnera na navodi početni znak /.



LXC Linux kontejneri za izolaciju se oslanjaju na sve izolirane prostore Linuxa (*Linux namespaces*) te na ***cgroup*** mehanizme za izolaciju i kontrolu procesa. Svaki pojedini proces unutar *LXC* linux kontejnera dobiva ***PID*** i ***GID*** brojeve iz opsega koji obično počinje od brojeva bližih 10.000 te se unutar *LXC* kontejnera povezuju na ***PID*** i ***GID*** brojeve koji normalno kreću od jedan (**1**).



Vezano opsege ovih **PID** i **GID** brojeva unutar kontejnera pogledajte datoteku `login.defs` odnosno poglavlje: **7.1.6.2. Datoteka /etc/login.defs.**

Naime sva ograničenja resursa sustava se rade preko takozvanih kontrolnih grupa odnosno ***cgroup*** mehanizama pa je logično da se ona konfiguriranju preko njih. Sve konfiguracije vezane za *LXC* Linux kontejnere se spremaju i definiraju tako da se u konačnici zapisuju u posebne datoteke unutar vršnog direktorija: `/sys/fs/cgroup/` pri čemu ispod tog direktorija slijedi vrsta ***cgroup*** mehanizma te ime našeg *LXC* Linux kontejnera.

**Za takozvane *Unprivileged* LXC kontejnere je moguće definiranje *PID* i *GID* brojeva koje će imati procesi od LXC linux kontejnera na fizičkom Linux računalu.**
Sa sljedećim opcijama možemo definirati od kojih ***PID*** i ***GID*** brojeva će se kretati s upotrebom za *LXC* Linux kontejnerske procese, kako bi se sa sigurnosne strane što više umanjila vjerojatnost približavanja ***PID*** brojevima od fizičkog Linuxa odnosno njegovih procesa. Definirajmo početne ***PID*** brojeve na 100.000 te ***GID*** brojeve isto na 100.000, s time da unutar LXC Linux kontejnera ***PID*** i ***GID*** brojevi kreću od 0 do 10.000.

Tada u našu konfiguracijsku datoteku* trebamo dodati sljedeće retke:

```
lxc.idmap = u 0 100000 10000
lxc.idmap = g 0 100000 10000
```

Naime normalni UNIX/Linux sustavi koriste ***PID/GID*** brojeve do maksimalno 32768 (vidljivo u varijabli `/proc/sys/kernel/pid_max`), a sve preko toga se smatra upotrebom od strane korisnika „*nobody*“ (*nitko*) koji gotovo da nema nikakva prava.

**Ova mogućnost znatno podiže razinu sigurnosni i koristi se za takozvane neprivilegirane (*engl.* *Unprivileged*) LXC Linux kontejnere, ali ima svojih ograničenja. Mi <u>nećemo</u> koristiti ovakvu metodu rada!**

**Ograničenja CPU-a**
Vezano za ograničenje broja CPU jezgri, to možemo postići sa sljedećom opcijom: ako recimo ograničavamo sustav na samo dvije CPU jezgre: 0 i 1, tada u našu konfiguracijsku datoteku* trebamo dodati sljedeći redak:

```
lxc.cgroup.cpuset.cpus = 0,1
```

U našem slučaju će ove postavke biti vidljive u datoteci:
`/sys/fs/cgroup/cpuset/lxc.payload/CentOS/cpuset.cpus`. Moguće je optimizirati raspodjelu dijeljenja CPU vremena između više *LXC* kontejnera s opcijom: `lxc.cgroup.cpu.shares`.

**Ograničenja RAM memorije**
Naravno moguće je ograničiti i maksimalnu količinu RAM memorije koju želimo dati na upotrebu *LXC* Linux kontejneru. Ako ograničenje radimo na 1GB RAM memorije tada bi morali dodati ovakvu opciju (u našu konfiguracijsku datoteku*):

```
lxc.cgroup.memory.limit_in_bytes = 1G
```

U našem slučaju će ove postavke biti vidljive u datoteci:
`/sys/fs/cgroup/memory/lxc.payload/CentOS/memory.limit_in_bytes`.

**Ograničenje upotrebe SWAP particije**
Moguće je ograničiti i koliko SWAP particije *LXC* kontejner smije koristiti, Ako ga recimo ograničimo na 200MB tada trebamo dodati sljedeći redak u našu konfiguracijsku datoteku*:

```
lxc.cgroup.memory.memsw.limit_in_bytes = 200M
```

**Omogućavanje upotrebe određenog uređaja odnosno dodjeljivanje prava pristupa na njega**
Moguće je dodavati i prava na upotrebu odnosno pristup određenom fizičkom uređaju, pa bi tako za blok uređaj **8:0** to bilo:

```
lxc.cgroup.devices.allow = b 8:0 rw
```

Ispis svih uređaja s pravima pristupa na njih možemo vidjeti s naredbom (za naš *LXC* Kontejner imena: `CentOS`):

```
lxc-cgroup -n CentOS devices.list
```

Za promjene maksimalne brzine čitanja ili zapisivanja na disk unutar *LXC* kontejnera imamo razne opcije. Mi ćemo koristiti opciju za ograničenje maksimalne brzine zapisivanja (u bajtima u sekundi) za disk uređaj *MAJ*:*MIN* oznake **8:0** na **10** MB/s:

```
lxc.cgroup.blkio.throttle.write_bps_device = 8:0 10485760
```

Ova vrijednost se potom zapisuje u datoteku (za naš LXC linux kontejner imena `CentOS`):
`/sys/fs/cgroup/blkio/lxc.payload/CentOS/blkio.throttle.write_bps_device`.

**Sve *cgroup* mehanizme možemo mijenjati direktno, tijekom rada LXC kontejnera s naredbom `lxc-cgroup`.**
Tako recimo, ako želimo napraviti ograničenje na naš *LXC* Linux kontejner imena: `CentOS` da može koristiti maksimalno 1GB RAM memorije, to možemo napraviti sa sljedećom naredbom:

```
lxc-cgroup -n CentOS memory.limit_in_bytes 1G
```

Moguće je konfigurirati automatsko pokretanje *LXC* Linux kontejnera prilikom pokretanja sustava, dodavanjem sljedećih redaka u konfiguracijsku datoteku* *LXC* kontejnera, a dodali smo i zadršku od pet sekundi prilikom pokretanja sljedećeg *LXC* kontejnera (ako ih se pokreće više):

```
lxc.start.auto = 1
lxc.start.delay = 5
```

## I na kraju pogledajmo još neke korisne naredbe koje pokrećemo s *„fizičkog"* Linuxa

**1. Kako vidjeti informacije o našem LXC kontejneru:**

```
lxc-info CentOS

Name:           CentOS
State:          RUNNING
PID:            26236
IP:             192.168.1.58
CPU use:        21.90 seconds
BlkIO use:      1.31 MiB
Memory use:     118.18 MiB
KMem use:       5.78 MiB
Link:           veth1
 TX bytes:      416.66 KiB
 RX bytes:      16.66 MiB
 Total bytes:   17.07 MiB
```

Sada za naš *LXC* Linux kontejner za koji vidimo da je pokrenut sa ***PID*** brojem `26236`, možemo vidjeti koje sve izolirane prostore Linuxa (Linux *namespace*) koristi (dužinu ispisa smo skratili ali ne i listu *namespace-ova*) s naredbom `lsns`:

```
lsns -p 26236

        NS TYPE   NPROCS   PID USER COMMAND
4026531837 user      135     1 root /usr/lib/systemd/systemd --switched-root . . .
4026532301 mnt        13 26236 root /sbin/init
4026532302 uts        13 26236 root /sbin/init
4026532303 ipc        13 26236 root /sbin/init
4026532304 pid        13 26236 root /sbin/init
4026532306 net        13 26236 root /sbin/init
4026532391 cgroup     13 26236 root /sbin/init
```

U prvom stupcu (`NS`) vidimo identifikatore svakog izoliranog prostora, a u drugom (`TYPE`) i njihovu vrstu. Sada, ako želimo vidjeti koji sve procesi koriste mrežni izoliran prostor (identifikator: `4026532306`) tada to možemo vidjeti s:

```
lsns 4026532306

  PID  PPID USER COMMAND
 8953  8946 root /sbin/init
 8979  8953 root ├─/usr/lib/systemd/systemd-journald
 8995  8953 dbus ├─/usr/bin/dbus-daemon --system --address=systemd: --nofork . . .
 8997  8953 root ├─/usr/lib/systemd/systemd-logind
 9004  8953 root ├─/sbin/agetty --noclear --keep-baud pts/2 115200,38400,9600 vt220
 9006  8953 root ├─/usr/sbin/crond -n
 9185  8953 root ├─/sbin/dhclient -1 -q -lf /var/lib/dhclient/dhclient--eth0.lease
 9240  8953 root ├─/usr/sbin/sshd -D
 9241  8953 root ├─/usr/sbin/rsyslogd -n
```

I ovdje zapravo vidimo procese s pripadajućim ***PID*** brojevima na fizičkom Linux računalu.

**2.Kako zaustaviti naš Linux kontejner:**

```
lxc-stop -n CentOS
```

**3.Kako obrisati naš LXC kontejner (nepovratno):**

```
lxc-destroy -n CentOS
```

**4.Kako pratiti performanse svih pokrenutih LXC linux kontejnera:**

```
lxc-top
```

Moguće su i druge radnje nad *LXC* linux kontejnerima, poput: kopiranja *LXC* kontejnera, izrade kopije *LXC* kontejnera u radu (*snapshot*) kao i zamrzavanja (*freeze*) rada svih njegovih procesa, te kasnijeg odmrzavanja (*unfreeze*) te drugih mogućnosti.



Za veću razinu sigurnosti pogledajte i poglavlje:
**9.4.6. Secure Computing Mode.**



**U praksi su Linux kontejneri brži od Virtualnih računala jer nemaju razne slojeve virtualizacije (emulacije i slično). Međutim, Podman, LXC ili bilo koji drugi Linux kontejneri potencijalno nisu dobar odabir za ekstremno visoko perfomantne primjene koje zahtjevaju što direktniji pristup hardveru!**

**Izvori informacija: (888),(889),(890),(891),(892),(893),(894),(K-8),** `man 7 lxc`, `man lxc.container.conf`, `man lxc-checkconfig`, `man lxc-create`, `man lxc-ls`, `man lxc-start`, `man lxc-attach`, `man lxc-info`, `man lxc-stop`, `man lxc-destroy`, `man lxc-top`, `man lxc-cgroup`, `man sysfs`. **Pogledajte i poster P5.**

## 27.2.2. Podman Linux kontejneri

*Podman* odnosno ***POD*** *manager* je alat otvorenog koda za razvoj, upravljanje i pokretanje Linux kontejnera na Linux sustavu. Izvorno je razvijen od strane tvrtke **Red Hat**® zajedno sa zajednicom otvorenog koda. Podman upravlja cijelim ekosustavom kontejnera koristeći **libpod** biblioteku. Njegova arhitektura bez dodatnih servisa čini ga sigurnijom i pristupačnijom opcijom za upravljanje Linux kontejnerima, a njegovi popratni alati i značajke, kao što su ***Buildah*** i ***Skopeo***, omogućuju prilagodbu okruženja kontejnera kako bi najbolje odgovaralo našim potrebama.

Pogledajmo sličnosti i različitosti ***Podman*** i ***LXC*** kontejnerske tehnologije:

| Značajka | LXC | Podman | Sličnosti |
|---|---|---|---|
| **Arhitektura** | Sistemski kontejneri | **OCI** (*Open Container Initiative*) kontejneri | Kontejnerska arhitektura |
| **Pristup** | Monolitni alat za upravljanje | Modularna arhitektura s alatima poput "podman" | Upravljanje |
| **Servisi** | **LXC** servis (`lxc`) | **Podman** servis (`conmon`) i **Systemd**. | Jednostavni servis za upravljanje kontejnerima |
| **Upravljanje resursima** | Napredno upravljanje resursima (`cgroups`) | Napredno upravljanje resursima (`cgroups`) | Kontrola resursa |
| **Instalacija** | Zahtijeva skup alata | Zahtjeva skup alata | Lakša instalacija |
| **Sigurnost** | Koristi se za izolaciju procesa na razini korisnika (***Namespaces***, ***SELinux***, ***Seccomp***) | Koristi se za izolaciju procesa na razini korisnika (***Namespaces***, ***SELinux***, ***Seccomp***) | Izolacija procesa |
| **Performanse** | Brže zbog manje slojeva apstrakcije | Može biti sporije zbog više slojeva apstrakcije i modularnosti | Efikasno izvršavanje |
| **Kompatibilnost** | Kompatibilan s nekim ***Docker*** alatima | Kompatibilan s ***Docker*** alatima, a podržava i **OCI** | Kompatibilnost s alatima za kontejnere |
| **Izoliranje resursa** | Upravljanje ***namespace***-ima integrirano u alate | Upravljanje ***namespace***-ima integrirano u alate | Korištenje ***namespace***-a |
| **Održavanje** | Zahtijeva više upravljanja zbog monolitne arhitekture | Jednostavnije održavanje zbog modularne arhitekture | Održavanje |

**Pokretanje s root ili non-root ovlastima**
*Podman* kontejnere se može pokretati s *root* ovlastima (tzv. *rootful*) i tada ne postoje neka posebna ograničenja u njihovom radu. S druga strane ako se Podman kontejneri pokreću kao tzv. *rootless*, odnosno s pravima običnih korisnika, tada oni primjerice nemaju prava pristupa do sistemskih direktorija te nemaju prava otvaranja mrežnih rezerviranih portova ispod 1024.

**Podman kontejneri**
*Podman* je moguće pozvati iz naredbenog retka da dohvati kontejnere iz repozitorija i pokrene ih. *Podman* koristi *systemd* jedinične datoteke (*unit files*) za definiranje, konfiguriranje i pokretanje kontejnera kao *systemd servisa*. Stoga se *Systemd* i koristi za kontrolu životnog ciklusa kontejnera, kao što su pokretanje, zaustavljanje i ponovno pokretanje, te može pružiti dodatne značajke kao što su sigurnosna ograničenja. Nadalje, korisnici mogu upravljati vlastitim spremištima (repozitorijima) na sustavu, kao i *systemd* jedinicama za kontrolu automatskog pokretanja i upravljanja vlastitim kontejnerima.
Podman osim Podman kontejnera odnosno *Open Container Initiative* (***OCI***) formata podržava i ***Docker*** kontejnere.
Posljednje, ali ne najmanje važno, za svaki kontejner se pokreće po jedan `conmon` (*Container Monitor*) proces. C*onmon* proces pokreće se prije kontejnera stoga što on daje upute donjoj komponenti kontejnera (kao što su `runc` ili `crun`) da stvori i pokrene kontejner. *Conmon* se brine i za prosljeđivanje dnevničkih (*log*) zapisa i izvođenje svih potrebnih radnji čišćenja nakon što se kontejner ugasi.

Što je **POD** ?
***Podovi*** su grupe Linux kontejnera koji rade zajedno i dijele iste resurse, slično ***Kubernetes POD-ovima***. Podman upravlja i tim podovima putem jednostavnog sučelja naredbenog retka (CLI) i *libpod* biblioteke, koja pruža API za upravljanje kontejnerima, podovima i diskovnim slikama kontejnera. Podmanov CLI stvara i podržava kontejnere tzv. *Open Container Initiative* (**OCI**), koji su dizajnirani prema industrijskim standardima te koriste točno definirane formate kontejnera.



Podman je moguće koristiti na raznim distribucijama Linuxa, kao što su ***Red Hat Enterprise Linux*** ili ***Rocky i Oracle Linux***, ***Fedora***, ***CentOS***, ali i **Ubuntu**.

Svaki **POD** sastoji se od jednog *infra* kontejnera i proizvoljnog broja uobičajenih kontejnera. *Infra kontejner* održava rad POD-ova i održava imeničke korisničke prostore (engl. *Linux namespace*), koji izoliraju kontejnere od glavnog računala. Svaki drugi kontejner ima monitor koji prati njegove procese i pazi na nefunkcionalne kontejnere (zablokirane ili u nekom drugom neupotrebljivom stanju) koji se ne mogu ukloniti iz okoline jer se neki od njihovih resursa još uvijek koriste.

**Primjena Podman kontejnera**
Podman kontejneri su obično privremeni i zamišljeni su kao mikro kontejneri, poput *Docker* kontejnera. Mikro kontejnerska arhitektura podrazumijeva da je jedan *Podman kontejner* zadužen za samo jedan servis; primjerice; samo za jedan web poslužitelj, samo za bazu podataka i slično. Sami *Podman kontejneri* obično se pokreću kada su potrebni, a zatim se također prema potrebi zaustavljaju ili čak i uklanjaju. Ipak, ako je to baš potrebno, moguće je i trajno spremiti podatke koje generira pokrenuti kontejner na diskovni sustav za trajnu pohranu.
Prema svom dizajnu, načinu rada i namjeni ***Podman kontejneri*** su bliže ***Docker*** nego ***LXC*** kontejnerima.



Vezano za *Linux kontejnere* te *Linux namespaces* i ***mikro kontejnere*** pogledajte:
**9.4.5. Izolirani prostori Linuxa (*Linux namespaces*).**

**Buildah i Skopeo**
Podman je modularni kontejnerski sustav, i kao takav može raditi zajedno s alatima kao što su **Buildah** i **Skopeo** za izgradnju i premještanje svojih kontejnera. S ***Buildahom*** možete izgraditi kontejnere ili od nule ili koristeći njihovu diskovnu sliku kao početnu točku. ***Skopeo*** premješta diskovne slike kontejnera između različitih vrsta sustava za pohranu, omogućujući vam kopiranje slika između registara kao što su: **docker.io**, **quay.io**, vašeg internog registra ili različitih vrsta pohrane na vašem lokalnom sustavu. *Podman*, *Buildah* i *Skopeo* koriste direktorij: `/var/lib/containers/` za pohranu diskovnih slika kontejnera i sâmih kontejnera.

**Pokretanje Podman kontejnera**
Za početak, potrebno je instalirati *Podman* programski paket (osim kojega će se instalirati još nekoliko programskih paketa):
```
yum -y install podman
```

Kada je instalacija gotova, možemo provjeriti inačicu *Podmana*, pomoću naredbe `podman`:
```
podman version
Client:        Podman Engine
Version:       4.6.1
API Version:   4.6.1
Go Version:    go1.20.12
Built:         Mon Apr 29 20:47:09 2024
OS/Arch:       linux/amd64
```
← U ispisu vidimo detalje o inačici *Podman* sustava koji je upravo instaliran.
Za detalje o konfiguraciji *Podman* sustava, možete pokrenuti:
```
podman info
```

Programski paket *Podman* dolazi sa zadanom konfiguracijom koja se nalazi u direktoriju: `/etc/containers/`.
Pogledajmo što se nalazi u ovom direktoriju:
- `policy.json` – politika pravila za potpisivanje diskovnih slika kontejnera.
- `registries.conf` – popis dostupnih registara (*repozitorija*) diskovnih slika kontejnera kao što su: *Docker registar*, *RHEL registar*, *Fedora registar* i drugi. Ovdje dodajemo i nove registre!
- `storage.conf` – konfiguracija zadanog sustava za pohranu (*storages*), kao i upravljački program i druge postavke sustava za pohranu kontejnera.
- `registries.d/` – direktorij s dodatnim (***YAML***) datotekama vezanim za konfiguraciju i potpisivanje diskovnih slika.
- `registries.conf.d/` – direktorij s dodatnim konfiguracijskim datotekama *aliasima* naziva kontejnera i slično.

U primjerima koji slijede mi imamo puna administratorska prava (`root`). Dakle sve radimo kao korisnik `root`.
Sada s naredbom `podman` možemo testno pokrenuti jedan (ograničeni) *Podman* kontejner:
```
podman run hello
!... Hello Podman World ...!

         .--"--.
       / -     - \
      / (O)   (O) \
   ~~~| -=(,Y,)=- |
    .---. /`  \   |~~
 ~/  o  o \~~~~.----. ~~
  | =(X)= |~  / (O (O) \
   ~~~~~~~  ~| =(Y_)=-  |
  ~~~~    ~~~|   U      |~~
```
← Ovdje zapravo vidimo da se učitao *alias* imena "***hello***", koji dohvaća *Podman* kontejner s registra (*repozitorija*) odnosno adrese: https://quay.io/repository/podman/hello.
⇨ Ovaj *Podman* kontejner zapravo ništa ne radi već ispisuje jednostavnu poruku "`!...Hello Podman World...!`" koju smo vidjeli gore, a nakon nje se kontejner automatski gasi. Dakle ovaj (mini) mikro kontejner se koristi praktično samo za testiranje rada *Podman* kontejnerskog sustava.

Sada pogledajmo statuse svih *Podman* kontejnera:

```
podman ps -a
CONTAINER ID  IMAGE                         COMMAND              CREATED         STATUS
8ee930c5a20a  quay.io/podman/hello:latest  /usr/local/bin/po...  1 minutes ago   Exited(0)
```

← Vidimo da je ovaj *Podman* kontejner ugašen prije 1. minute (`STATUS` → `Exited(0)`).

**Kako pronaći već postojeći Podman kontejner s dostupnih registara (repozitorija)?**
Za pronalaženje već izrađenih *Podman* kontejnera ponovno ćemo koristiti naredbu: `podman` i to ovako.

```
podman search nginx
```

← Mi smo konkretno tražili *Podman* kontejner s ***Nginx*** servisom (web poslužiteljem).

Od pronađenih *Podman* kontejnera s nekoliko registara, mi ćemo odabrati onaj imena: "**nginx**" s repozitorija **`docker.io/kasmweb/`**:

```
podman pull docker.io/kasmweb/nginx

Trying to pull docker.io/kasmweb/nginx:latest...
Getting image source signatures
Copying blob a0d0a0d46f8b done
Copying blob 61074acc7dd2 done
Writing manifest to image destination
```

Nakon prethone procedure možemo pogledati koje *Podman* diskovne slike imamo kopirane lokalno:

```
podman images

REPOSITORY                TAG        IMAGE ID      CREATED      SIZE
quay.io/podman/hello      latest     0e9d283b971f  2 days ago   787 kB
docker.io/kasmweb/nginx   latest     95cbbc869690  2 years ago  26.9 MB
```

← Zadnji (označeni) je naš upravo kopirani (`nginx`) kontejner, koji je veličine (`26.9 MB`).



Ako sami želite napraviti svoju diskovnu sliku *Podman* kontejnera, preporuka je to učiniti korištenem tzv. *Universal Base Image* (**UBI**) diskovne slike. Ona je minimalna moguća instanca **Red Hat** Linuxa (v.8 ili v.9) na čijoj osnovi možete izgraditi svoju diskovnu sliku sa željenom aplikacijom (servisom) unutar nje.

**UBI** diskovne slike u nazivu imaju riječ ***ubi***, primjerice `ubi7` (za *Red Hat* 7.x), `ubi8` (za *Red Hat* 8.x) i tako dalje.
Sve ih možete pronaći sa:

```
podman search ubi
```

**I konačno slijedi pokretanje Podman kontejnera**
U ovom koraku ćemo pokrenuti *Podman* kontejner koji smo prethodnom procedurom kopirali na svoje računalo:

```
podman run -it --rm -d -p 8080:80 --name web nginx
```

Pogledajmo što smo sve koristili od prekidača i što oni znače:

- `-it` – koristimo interaktivni način rada (`-i`) i dodjeljujemo mu sesiju za rad u terminalu (`-t`).
- `--rm` – kontejner će biti uklonjen kada se zaustavi njegovo izvršavanja (kako bi se oslobodili resursi sustava).
- `-d` – kontejner će biti pokrenuti u pozadini (kao servis) u načinu rada u kojem neće zauzeti trenutni terminal.
- `-p 8080:80` – povezujemo lokalni port **8080** s portom **80** unutar kontejnera (s web poslužiteljem u njemu).
- `--name web nginx` – `web` je naziv kontejnera, a `nginx` je ime diskovne slike kontejnera koju pokrećemo.

Sada pogledajmo što se pokrenulo:

```
podman ps
CONTAINER ID  IMAGE            COMMAND    CREATED    STATUS PORTS                 NAME
7cf61cccccd2  . .nginx:latest /start.sh  13 second  Up     0.0.0.0:8080->80/tcp  web
```

← Ovdje vidimo ID kontejnera, status (`Up`) te vezu TCP portova: 8080→80 i ime kontejnera (`web`).
To znači da je na računalu na kojem smo pokrenuli ovaj kontejner (*Host računalo*) otvoren TCP port 8080 kojem možemo pristupiti izvana (s mreže), a taj port se translatira na port 80 unutar ovog kontejnera.
Na portu 80 unutar kontejnera je pokrenut **nginx** web poslužitelj.

**Log datoeke**
Moguće je u bilo kojem trenutku pregledavati i dnevničke (log) datoteke *Podman* kontejnera, što konkretno možemo učiniti sa:

```
podman logs web
crond[2]: crond (busybox 1.33.1) started, log level 8
2024/05/08 09:49:24 [notice] 3#3: using the "epoll" event method
2024/05/08 09:49:24 [notice] 3#3: nginx/1.21.3
2024/05/08 09:49:24 [notice] 3#3: start worker processes
```

← Vidimo da log datoteka unutar kontejnera počinje s pokretanjem **nginx** servisa (`nginx/1.21.3` i `start worker processes`), a zatim slijede i druge poruke (koje nismo prikazali).

**Pokretanje naredbenog retka i ulazak u njega (unutar kontejnera)**
Za pokretanje naredbenog retka i ulazak u njega (unutar kontejnera) trebamo naptaviti sljedeće:

```
podman exec -it web /bin/sh
/ #
```

← Sada se nalazimo u ***SH*** ljusci (naredbenom retku) unutar *Podman* kontejnera.
Ovdje možemo mijenjati sve što želimo promjeniti unutar kontejnera.
Iz ovog naredbenog retka izlazimo sa: `exit`.

**Pokretanje naredbi unutar kontejnera i izolacija resursa**
Moguće je i pokrenuti samo željenu naredbu unutar kontejnera (pr. našeg koji se zove `web`):

```
podman exec web ps -ef
PID   USER     TIME  COMMAND
    1 root      0:00 {start.sh} /bin/sh /start.sh
    2 root      0:00 crond -f
    3 root      0:00 nginx: master process nginx -g daemon off;
    4 nginx     0:00 nginx: worker process
    5 nginx     0:00 nginx: worker process
    6 root      0:00 ps -ef
```

← U ovom primjeru smo unutar kontejnera (`web`) pokrenuli naredbu **`ps -ef`** koja nam ispisuje sve procese unutar kontejnera.

Ovdje je važno uočiti procese (pokrenute programe) čiji **PID** brojevi počinju od 1, kao što je i očekivano.
To znači da su unutar kontejnera izolitrani i PID prostori.

Ako se vratimo na naše *Host* računalo s kojeg smo pokrenuli kontejner, pogledajmo stablo procesa:

```
pstree
systemd─┬─NetworkManager───2*[{NetworkManager}]
        ├─agetty
        ├─auditd─┬─sedispatch
        │        └─2*[{auditd}]
        ├─chronyd
        ├─conmon───start.sh─┬─crond
        │                   └─nginx───2*[nginx]
        ├─crond
```

← Vidimo da je normalno prvi (**PID** 1) proces `systemd`, koji je u trenutku kada smo pokrenuli kontejner, pokrenuo: `conmon` proces koji je potom pokrenuo kontejner. A unutar tog kontejnera je pokrenut `nginx` web poslužitelj.
Potpune detalje izolacije s *Linux namespace* mehanizmima možemo vidjeti sa:

```
pstree -s -S
systemd─┬─NetworkManager(mnt)───2*[{NetworkManager}]
        ├─agetty
        ├─auditd(mnt)─┬─sedispatch
        │             └─2*[{auditd}]
        ├─chronyd(mnt,uts)
        ├─conmon───start.sh(cgroup,ipc,mnt,net,pid,uts)─┬─crond
        │                                               └─nginx───2*[nginx]
```

← Dakle za naš kontejner (`conmon`) odnosno skriptu koja ga je pokrenula (`start.sh`) vidimo sve izolacijske mehanizme, poput `cgroup`, `ipc`, `mnt`, `net`, `pid` i `uts`.



Vezano za *Linux kontejnere* te *Linux namespaces* i ***mikro kontejnere*** pogledajte:
**9.4.5. Izolirani prostori Linuxa (*Linux namespaces*).**

**Izolacija UID i GID brojeva**
Izolacija *PID* i *GID* brojeva untar kontejnera odrađuje se pomoću navedenih *Linux namespaces* tehnika, a u konkretnom slučaju se izolacija *PID* i *GID* opsega radi pomoću ***PID namespace*** mehanizma.
Naime u Linux sustavu (kao i UNIX), svakom korisniku se dodjeljuje jedinstveni ID korisnika (**UID**) i ID grupe (**GID**) za kontrolu pristupa datotekama, procesima i resursima sustava. Međutim, kako se broj korisnika i aplikacija povećava, povećava se i rizik od sukoba UID-a ili GID-a, a pogotovo u okruženjima kao što su Linux kontejneri (svih vrsta). Stoga se definiraju dodatni rasponi ID brojeva (UID) podređenih korisnika koje možemo dodijeliti pojedinačnim korisnicima ili procesima.
To omogućuje stvaranje i rad UID-a (ili GID-a) unutar izoliranih korisničkih imenskih prostora (*PID namespaces*).
Definicija opsega podređenih *PID* i *GID* brojeva za imenske prostore definiraju se u datotekama: `/etc/subuid` i `/etc/subgid`, a inicijalno u datoteci: `/etc/login.defs`.



Vezano za `login.defs` datoteku pogledajte poglavlje:
**7.1.6.2. Datoteka /etc/login.defs.**

**Zaustavljanje Podman kontejnera**
Za zaustavljanje kontejnera (pr. **web**) koristimo sljedeću sintaksu:

```
podman stop web
```

I sada provjerimo je li se stvarno i zaustavio naš *Podman* kontejner:

```
podman ps
CONTAINER ID   IMAGE        COMMAND      CREATED      STATUS       PORTS        NAMES
```

← U ispisu nemamo niti jedan *Podman* kontejner, dakla zaustavljanje je napravljeno bez problema.

**Brisanje diskovne slike kontejnera**
U slučaju kada nam diskovna slika kontekenra više ne treba, jer konkretni kontejner više neželimo ili ne trebano koristiti, i to je moguće napraviti.
Obrišimo primjerice diskovnu sliku kontejenera koji smo prvi kreirali (onog imena `hello`).

```
podman rmi --force 0e9d283b971f
```

← **Ovdje smo kao identifilator koristili ID kontejnera!**

Identifikator (**ID**) kontejnera smo dobili s naredbom:

```
podman images
```

# Napredni rad s kontejnerima

Za početak naprednog rada s *Podman* kontejnerima, prvo moramo instalirati još neke alate:

```
yum -y install container-tools
```

Provjeru sadržaja određenih diskovnih slika kontejnera s udaljenog repozitorija također je moguća.
Primjerice za naš kontejner ***nginx*** koji smo dohvatili s repozitorija: **docker.io/kasmweb/nginx**.

Za tu namjenu potrebna nam je naredba `skopeo`:

```
skopeo inspect docker://docker.io/kasmweb/nginx

{
    "Name": "docker.io/kasmweb/nginx",
    "Digest":
"sha256:03957e8cace460d39b1e5709cb5282ce68e36a50dcef6c289d12924cff5685bb",
    "RepoTags": [
        "1.25.1",
        "1.25.3",
        "develop",
        "develop-rolling",
        "latest",
        "latest-rolling"
]

"Architecture": "amd64",
    "Os": "linux",

"LayersData": [
        {
            "MIMEType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
            "Digest":"sha256:a0d0a0d46f466318f479767577551a53ffc9074c9fa7035982e",
            "Size": 2814446,
            "Annotations": null
        },

"Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "NGINX_VERSION=1.21.3",
. . .
```

← U ovom skraćenom ispisu vidimo:

- Dodatne repozitorije koji se koriste ("`RepoTags`").
- Kao i za koju arhitektutu računala je diskovna slika namijenjena (`Architecture`).
- Osim toga vidljive su datoteke koje sadrže diskovnu sliku datotečnog suatava koji će se koristiti unutar kontejnera (`vnd.docker.image.rootfs.diff.tar.gzip`).
- Vidljive su i varijable sustava (`Env`) i drugi detalji.

## Dodavanje registara
Moguće je dodavati registre s interneta (ili neke druge mreže) ili lokalno kreirane (od naše strane).
Oboje se radi uređivanjem datoteke: `/etc/containers/registries.conf`.

**Dodavanje lokalnog registra**
Ako primjerice sami kreiramo svoj registar (ovdje nećemo objašnjavati sve korake potrebne za to) i želimo ga dodati, to možemo učiniti dodavanjem sljedeće sekcije odnosno redaka u navedenu datoteku:
```
[[registry]]
location="localhost:8000"
insecure=false
blocked=false
```
🡐 Ovakva konfiguracija definira upotrebu lokalnog registra na portu **8000**.

**Dodavanje mrežnog registra (s interneta)**
Naravno moguće je dodati i neki od registata već dostupnih na internetu, primjerice **Fedora** registar.
To ćemo učiniti dodavanjem sljedeće sekcije:
```
[[registry]]
location = "registry.fedoraproject.org"
insecure = false
blocked = false
```

🡐 Dodatno ćemo za pretraživanje i tog registra, postojeći redak konfiguracije:
```
unqualified-search-registries = ["registry.access.redhat.com", "registry.redhat.io",
"docker.io"]
```

🡐 Zamijeniti sa:
```
unqualified-search-registries = ["registry.access.redhat.com", "registry.redhat.io",
"docker.io", "registry.fedoraproject.org"]
```
- Dakle dodali smo novi unos ("`registry.fedoraproject.org`").

Sve dostupne registre sada možemo provjeriti s naredbom:
```
podman info
```

I u ovom podužem ispisu će se u sekciji “`registries:`“, pojaviti i naš novi **Fedora** registar:
```
registries:
  docker.io:
    Blocked: false
    Insecure: false
. . .

  registry.access.redhat.com:
    Blocked: false
    Insecure: false
. . .

  registry.fedoraproject.org:
    Blocked: false
    Insecure: false
    Location: registry.fedoraproject.org
    MirrorByDigestOnly: false
    Mirrors: null
    Prefix: registry.fedoraproject.org
. . .
```

Osim toga i u “`search`“ sekciji ćemo sada imati sve ove registre, uključujući i upravo dodani **Fedora**:
```
search:
  - registry.access.redhat.com
  - registry.redhat.io
  - docker.io
  - registry.fedoraproject.org
```

## Kreiranje vlastite diskovne slike

U ovom primjeru kreirat ćemo novu diskovnu sliku, na osnovi već pripremljene **UBI** disovne slike (za **Red Hat** 9).

U slučaju kada želite koristiti repozitorije poput službenih *Red Hat*, potrebno je prvo se registrirati, te logirati. Primjerice sa:

```
podman login registry.access.redhat.com
Username: XXXX
Password: YYYYY
Login Succeeded!
```

← Ako je sve u redu, dobit ćete pristup do željenog repozitorija.
Ako samo dohvaćate *Podman* diskovne slike, na ovaj repozitorij (kao i `docker.io`), za to imate pravo pristupa.

Međutim, ako želite koristiti *Red Hat* repozitorije za instalaciju programskih paketa, tada je potrebna registracija; primjerice upotrebom `subscription-manager` alata, a potom i uz dodavanje željenih repozitorija; primjerice ovako:

```
subscription-manager repos --enable=rhel-9-server-extras-rpms
```

Sada možemo kreirati predložak konfiguracije *Podman* kontejnera.

Kreirajmo novi direktorij:

```
mkdir podman-web
cd podman-web
```

U tom direktoriju kreirajmo datoteku imena: `Containerfile`, koja će sadržavati:

```
FROM ubi9-init
RUN dnf -y install nginx
RUN systemctl enable nginx
```

← U uvom primjeru:

- `FROM ubi9-init` – dohvaćamo minimalnu (**UBI**) diskovnu sliku od **Red Hat 9 (**`ubi9-init`**)**,
- `RUN dnf -y install nginx` – instaliramo softverski paket “`nginx`“.
- `RUN systemctl enable nginx` – aktiviramo ga kao kao servis, da se pokreće prilikom svakog pokretanja kontejnera.

I konačno pokrenimo proces izrade kontejnera iz naše konfiguracijek datoteke u trenutnom (radnom) direktoriju (`.`):

```
podman build -t net-tool .
```

← Naziv *Podman* kontejnerske slike će biti `net-tool`, a koristit će se predložak (datoteka) koju smo kreirali u ovom direktoriju.

Pogledajmo sada što imamo od novih diskovnih slika (skratili smo ispis):

```
podman images

REPOSITORY                  TAG        IMAGE ID       CREATED              SIZE
localhost/net-tool          latest     51a7c7c8dcdd   About a minute ago   264 MB
docker.io/kasmweb/nginx     latest     95cbbc869690   2 years ago          26.9 MB
```

← Ovaj prvi u listi (`net-tool`) je naš upravo kreirani *Podman* kontejner.



Za veću razinu sigurnosti pogledajte i poglavlje:
**9.4.6. Secure Computing Mode.**



U većim sustavima, za upravljanje Podman kontejnerima i POD-ovima, uobičajeno se koristi ***Kubernetes*** sustav.
U još većim okruženjima uobičajeno je koristiti i neke od specijaliziranih platformi za tu namjenu, poput ***Red Hat OpenShift***.

**Izvori informacija:** (1652),(1653),(1654),(1655),(1656),(1657),(1660),(), `man podman`, `man cgroups`, `man namespaces`, `man seccomp`, `man skopeo`, `man podman-rmi`, `man podman-images`, `man podman-ps`, `man subuid`, `man subgid`, `man login.defs`, `man podman-info`, `man podman-logs`, `man buildah`.

# 27.3. Chroot i chroot jail

Od operativnog sustava Unix, inačice 7 (tzv. **V7**) iz 1979. godine razvijao se mehanizam koji se naziva ***chroot*** i koristio se za izolaciju datotečnog sustava. On je služio kao preteča cjelokupnog modernog koncepta kontejnerizacije.
Danas se po pitanju kontejnerizacije oslanjamo na *imenične prostore* i *cgroups* mehanizme, dok je u prošlosti ***chroot*** bio jedina metoda za uspostavljanje izoliranog okruženja odvojenog od glavnog operativnog sustava. Ova izolacija bila je osobito vrijedna za scenarije testiranja. Osim toga, zanimljivo je primijetiti da je sam izraz "**chroot**" (engl. *change root*) sažeti oblik "*promjene korijena*", označavajući njegovu funkciju u mijenjanju korijena datotečnog sustava.

Istovremno `chroot` je i naredba na Unixu i operativnim sustavima sličnim Unixu (Linuxu) koja mijenja korijenski direktorij za trenutni pokrenuti proces i njegove pod procese. Program koji se izvodi u tako modificiranom okruženju ne može pristupiti datotekama izvan svog stabla direktorija. Ovakvo modificirano okruženje naziva se *chroot jail* (zatvor).
To znači da programi koji se pokreću unutar *chroot* okruženja mogu vidjeti i manipulirati samo s datotekama i direktorijima unutar tog okruženja, a ne mogu izlaziti vani.

Način upotrebe

- **Kreiranje okruženja** - prvo se kreira direktorij koji će služiti kao novi korijenski direktorij datotečnog sustava. U taj direktorij se kopiraju sve datoteke i direktoriji koji su potrebni za rad željenog programa ili više njih (npr. binarne datoteke i pripadajuće biblioteke, konfiguracijske datoteke i slično).
- **Promjena "root" direktorija** – na najnižoj razini koristi se *chroot()* sistemski poziv za promjenu korijenskog tj. "*root*" direktorija u novi korijenski direktorij datotečnog sustava.
- **Pokretanje programa** - program se pokreće unutar *chroot* okruženja. Program može vidjeti i manipulirati samo datotekama i direktorijima unutar tog okruženja.

Prednosti korištenja *chroot jail-a* su sljedeće:

- **Sigurnost** - *chroot jail* se može koristiti za izolaciju programa i sprječavanje zloupotrebe. Na primjer, *chroot jail* se može koristiti za pokretanje nepouzdanih programa koji ne bi trebali imati pristup cjelokupnom sustavu.
- **Ograničenje resursa** - *chroot jail* se može koristiti za ograničenje resursa koje program može koristiti. Na primjer, *chroot jail* se može koristiti za ograničenje dostupnog diskovnog prostora.
- **Virtualizacija** - *chroot jail* se može koristiti kao lagana alternativa virtualnim računalima ili Linx kontejnerima. *Chroot jail* u radu koristi znatno manje resursa i lakše se postavlja.
- **Kontrola ovisnosti o programskim bibliotekama** – željeni softver se može razvijati, kompilirati i testirati u *chroot-u* popunjenom samo s njegovim bibliotekama. To može spriječiti probleme upotrebe, ali i rada s različitim skupovima programskih biblioteka.
  - **Kompatibilnost** - naslijeđeni ili stariji softver odnosno softver koji koristi drugačiji **ABI** ponekad se mora izvoditi u *chroot*-u jer bi se njihove prateće biblioteke ili izvršne (binarne) datoteke inače mogle sukobiti prilikom kompiliranja ili povezivanja s onima od glavnog sustava.

**Chroot osnove**
Naredba `chroot` u Linuxu jest alat koji mijenja prividni korijenski direktorij za trenutni pokrenuti proces i njegove podređene procese. To znači da proces neće 'vidjeti' direktorije iznad novog korijenskog direktorija. Na taj način stvaramo okruženje u kojem će naš proces (program) raditi, ali izolirano od ostatka sustava, po pitanju stabla direktorija i datoteka koje su mu dostupne. Probajmo to napraviti s **bash** ljuskom, uz kreiranje potrebnih direktorija i datoteka unutar *chroot* okruženja:

```
mkdir -p /tmp/newroot/bin
mkdir -p /tmp/newroot/lib{,64}
mkdir -p /tmp/newroot/usr/bin
```

Kopirajmo naredbe `bash` i `tree` u novo *chroot* okruženje u direktorij `/tmp/newroot/`:

```
cp /bin/bash /tmp/newroot/bin
cp /usr/bin/tree /tmp/newroot/usr/bin/
```

Provjerim koje sve bibliotekle trebamo za naredbe: `bash` i `tree`:

```
ldd /bin/bash
ldd /usr/bin/tree
```

← Dobili smo listu svih potrebnih biblioteka, koje treba kopirati.

Sada na osnovi ove liste biblioteka koje nam trebaju, kopirajmo potrebne biblioteke u novo *chroot* okruženje :

```
cp /lib64/libtinfo.so.6 /tmp/newroot/lib64/
cp /lib64/libc.so.6 /tmp/newroot/lib64/
cp /lib64/ld-linux-x86-64.so.2 /tmp/newroot/lib64/
cp /lib64/libcap.so.2 /tmp/newroot/lib64/
cp /lib64/libselinux.so.1 /tmp/newroot/lib64/
cp /lib64/libpcre2-8.so.0 /tmp/newroot/lib64/
```

I sada konačno ulazimo u *chroot* okruženje:

```
chroot /tmp/newroot /bin/bash
```

Pogledajmo stablo direktorija unutar *chroot* okruženja u kojem se upravo nalazimo, s naredbom `tree`:

```
tree -d -L 1 /

/
|-- bin
|-- lib
|-- lib64
|-- usr
`-- var
```

← Ovdje također vidimo da se nalazimo u vršnom direktoriju odnosno u izoliranom *chroot* okruženju.

## Root jail

**Potpuni root jail podrazumjeva kopiranje svih potrebnih binarnih (izvršnih) datoteka i biblioteka u novi direktorij.**
Da bi to radilo, primjerice potrebno je minimalno kreirati posebne direktorije, poput:

```
mkdir -p /tmp/newroot/{proc,sys,dev,run}
```

U sljedećem koraku je potrebno napraviti montiranje posebnih pseudo datotečnih sustava (`/proc`, `/sys`, `/dev`)

```
mount --bind /proc /tmp/newroot/proc
mount --bind /sys /tmp/newroot/sys
mount --bind /dev /tmp/newroot/dev
mount --bind /run /tmp/newroot/run
```



Vezano za ***bind mount*** pogledajte poglavlje:
**13.7. Kako se montiraju particije u direktorije i zašto.**

Nakon ovog koraka potrebno je (što ovisi o softveru koji želite instalirati u *root jail*) kopirati cijelu strukturu direktorija s pripadajućim datotekama unutar *root jaila*, što nećemo objašnjavatti jer to ovisi o mnogim detaljima koji ovise o sâmom softveru.

**Druga jednostavnija metoda jest uporaba root jail-a unutar aplikacija (koje to podržavaju).**
Pogledajmo primjerice samo kraću listu softvera koji podržava *chroot* (uz opis opcije s kojom se to postiže):

- ***FTTP*** i ***SSH*** poslužitelji:
  - ***vsftpd*** - opcija `chroot_local_user`.
  - ***proftpd*** - opcija `ChrootEveryone`.
  - ***sshd*** - opcija `ChrootDirectory` u konfiguracijskoj datoteci.
- ***Web*** poslužitelji:
  - ***Apache*** - opcija `chroot` u konfiguracijskoj datoteci.
  - ***Nginx*** - opcija `chroot` u konfiguracijskoj datoteci.
- ***Mail*** poslužitelji:
  - ***Postfix*** - opcija `chroot` u konfiguracijskoj datoteci.
  - ***Dovecot*** - opcija `chroot` u konfiguracijskoj datoteci.
- Baze podataka:
  - ***MySQL***/***MariaDB*** - opcija `--chroot` pri pokretanju servisa.
  - ***Redis*** - opcija `--dir` pri pokretanju servisa.
  - ***SQLite*** `--vfs=` pri pokretanju servisa
- ***Proxy*** i ***DNS*** poslužitelji:
  - ***SQUID*** - opcija `chroot` u konfiguracijskoj datoteci.
  - ***BIND*** - opcija `chroot` u konfiguracijskoj datoteci.
- Neke od naredbi:
  - `rsync` - opcija `--chroot=` . Koristi se primjerice za ograničenje vidljivosti datotečnog sustava *rsync*-a na direktorij koji sadrži datoteke koje želite kopirati.
  - `yum` - opcija `--installroot`, koja može instalirati softverski paket u alternativni direktorij, pa tako i u *chroot* direktorij.
  - `rpm` - opcija `--root`, koja također može instalirati softverski paket u alternativni direktorij, pa tako i u *chroot* direktorij.



**Što se tiće upotrebe odnosno konfiguracije root jail-a u aplikacijama, to naravno ovisi o sâmoj aplikaciji i njenoj inačici pa stoga nećemo navoditi neke primjere. Umjesto toga, tu temu istražite sami.**

**Izvori informacija: (1633),(1634),(1635),(1636), `man chroot`, `man mount`.**

# Sigurnosni aspekti Linuxa

# 28. Sigurnosni aspekti Linuxa

U ovom poglavlju naročito ćemo obratiti pozornost na sigurnosne aspekte rada operativnih sustava, a poglavito Linuxa.

## 28.1. Važnost redovite nadogradnje operativnog sustava

Svi operativni sustavi kao i sve aplikacije (programi) koje su na njima instalirane, podložne su raznim propustima.
Neki od tih propusta su kritični te iskorištavanjem njihovih slabosti napadači mogu preuzeti kontrolu nad vašim računalom ili cijelim IT sustavom, a neki drugi su malo manje kritični.

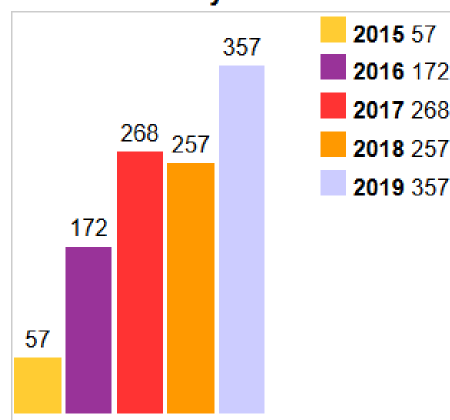
Stoga je kao prva razina zaštite vrlo važno redovito raditi nadogradnje operativnih sustava: od osobnih računala, preko poslužitelja, ali i svih uređaja na mreži poput:

- Usmjerivača (*routera*), preklopnika (*switcheva*), vatrozida (*firewalla*) i drugih mrežnih uređaja.
- Mrežnih pisača i skenera.
- Te drugih posebnih uređaja poput *NAS* ili *SAN* uređaja.

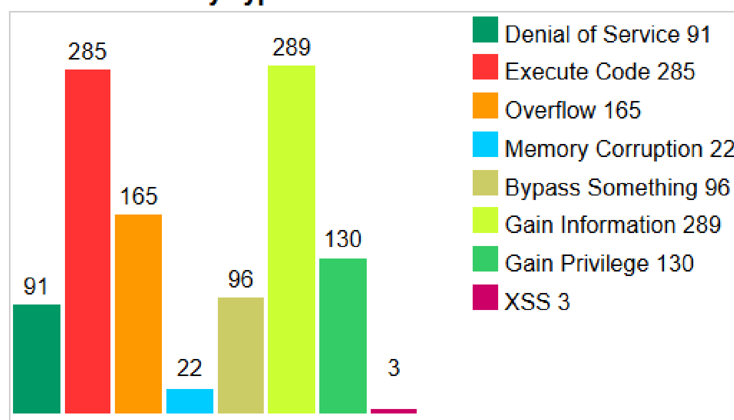
Da bi bili svjesni činjenice koliko su svi operativni sustavi ranjivi i koliko se moraju nadograđivati, pogledajte statistike broja otkrivenih sigurnosnih propusta prema kategorijama i godinama za nekoliko operativnih sustava. Lijevi graf pokazuje broj sigurnosnih propusta koji su otkriveni, kao i njihovih popravaka odnosno zakrpi, a desni dio njihovu kategoriju (težinu).
Pogledajmo statistike za **Windows 10** na slici dolje.

**Slika 259. Windows 10 sigurnosni izvještaji prema godinama**

Vulnerabilities By Year




Vulnerabilities By Type

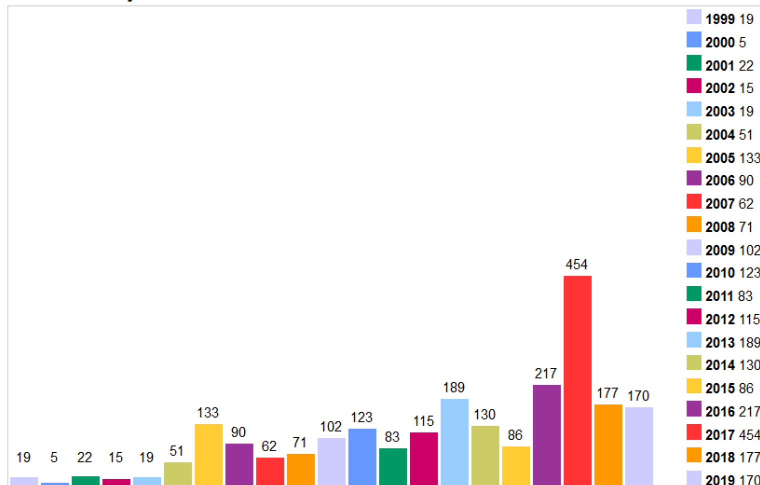



**Izvor informacija:** https://www.cvedetails.com/product/32238/Microsoft-Windows-10.html?vendor_id=26

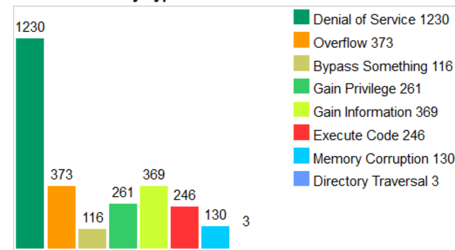
Za **Linux kernel**: koriste ga i razni uređaji poput preklopnika, usmjerivača i vatrozida te *NAS* sustavi, mrežni pisači i skeneri...

**Slika 260. Sigurnosni izvještaji za Linux kernel, prema godinama.**

Vulnerabilities By Year



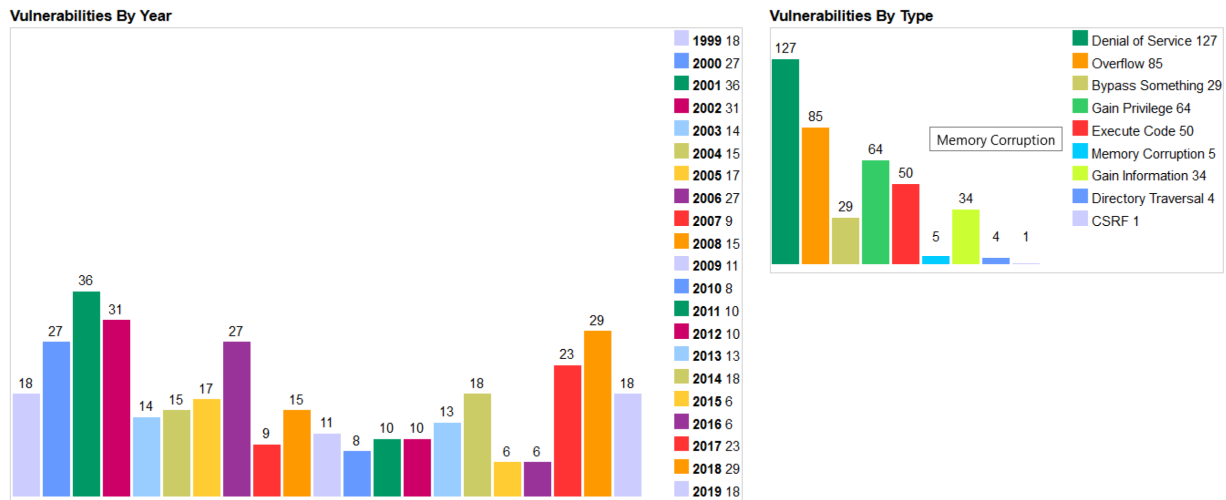

Vulnerabilities By Type




**Izvor informacija:** https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

Za ***FreeBSD*** Unix, koji koriste i preklopnici (*switchevi*), usmjerivači (*routeri*), vatriozidi (*firewalli*) kao i druga mreža oprema te uređaji poput primjerice mrežnih uređaja tvrtki: ***Cisco***, ***Juniper***, ***CheckPoint***, *Netgate* ***(pfSense)*** i drugih.

**Slika 261. Sigurnosni izvještaji za BSD Unix, prema godinama.**




**Izvor informacija:** https://www.cvedetails.com/vendor/6/Freebsd.html

---

# 28.2. SELinux sustav

***SELinux*** (engl. *Security-Enhanced Linux*) sustav implementiran je na razini kernel modula (kernela) koji Linuxu nudi mehanizam za podršku sigurnosnih politika kontrole pristupa, uključujući obavezne kontrole pristupa znane kao ***MAC*** (engl. *Mandatory Access Controls*). *SELinux* je skup funkcionalnosti na razini kernela, ali i nîza alata (programa) na razini korisničkih programa (*User space*) odnosno na razini iznad razine kernela, koji su dodani u mnoge distribucije Linuxa.
Njegova arhitektura nastoji razdvojiti provedbu sigurnosnih postavki od sigurnosnih politika te pojednostaviti količinu softvera koji je uključen u provedbu tih sigurnosnih postavki. Ključni koncepti na kojima se temelji *SELinux* razvijan je u nekoliko ranijih projekata Državne agencije za nacionalnu sigurnost Sjedinjenih Američkih Država ***NSA*** (engl. *National Security Agency*). Ako ***SELinux*** sustav nije omogućen, koriste se sâmo tradicionalne metode kontrole pristupa poput ovlasti nad datotekama. Naime s klasičnim ovlastima definiramo tko (vlasnik, grupa ili ostali) ima koja prava nad datotekama ili direktorijima (pravo čitanja, zapisivanja ili izvršavanja). Međutim korisnici, ali i programi mogu drugima dati nesigurna dopuštenja za datoteke ili pristupiti dijelovima sustava koji inače ne bi trebali biti korišteni za normalan rad.
***SELinux*** ima tri osnovna načina rada, od kojih je **Enforcing** postavljen kao zadani način rada. No, postoji i dodatna granulacija, koja može biti takozvana: ***targeted***, ***minimum*** ili ***mls***, od kojih svaka podiže ili spušta razinu sigurnosti.
No vratimo se na navedena tri moguća način rada ***SELinux*** sustava:

- ***Enforcing*** (`enforcing`) je postavljeni odnosno zadani (Engl. *Default*) način rada u kojem su uključeni sigurnosni mehanizmi koji aktivno nadziru sustav tako da zabranjuju radnje koje nisu dozvoljene i snimaju sve pokušaje pristupa.
- ***Permissive*** (`permissive`) je način rada u kojem se prati sustav, ali se u slučaju nedozvoljenih radnji one sâmo zapisuju (logiraju) te se ne primjenjuju nikakva pravila. Prema tome ovo je način rada koji sâmo prati stanje, ali ne poduzima nikakve radnje.
- ***Disabled*** (`disabled`) je način rada u kojem izričito isključujemo cijeli ***SELinux*** sustav.

## 28.2.1. Konfiguracija SELinux sustava

Konfiguracija *SELinux* sustava koji je u pravilu uvijek instaliran i aktiviran, radi se u inicijalnoj konfiguracijskoj datoteci: `/etc/selinux/config`.
Standardno ovdje imamo dvije varijable koje sustav koristi za postavljanje načina rada *SELinux* sustava:

- `SELINUX=` označava način rada *SELinux* sustava. Ovdje se može postaviti jedna od tri vrijednosti:
    - `enforcing` – ovo je postavljeni odnosno zadani (Engl. *Default*) način rada koji aktivira *SELinux*.
    - `permissive` – ovo je način rada koji aktivira *SELinux*, ali samo da prati i logira poruke.
    - `disabled` – ovo je način rada koji deaktivira odnosno u potpunosti isključuje *SELinux* sustav.
- `SELINUXTYPE=` označava odabir metode odnosno razine ili pravilnika *SELinux* zaštite, koje mogu biti:
    - `targeted` – ovo je postavljeni odnosno zadani (Engl. *Default*) način rada u kojem se prate samo označeni procesi.
    - `mls` – ovo je način rada *SELinux* sustava s aktivacijom rada na više razina.
    - `minimum` – ovo je minimalan način rada *SELinux* sustava koji podrazumijeva aktivaciju *SELinux* sustava samo za odabrane procese.

Standardne postavke datoteke `/etc/selinux/config` su sljedeće:

```
SELINUX=enforcing
SELINUXTYPE=targeted
```

To znači kako je *SELinux* sustav aktiviran te da koristi metodu `targeted`. Ako želimo isključiti *SELinux* sustav, tada je potrebno postaviti varijablu `SELINUX=disabled` te restartati računalo.



Kako trajno onemogućiti upotrebu *SELinux* sustava na razini kernela, pogledajte u poglavlju:
**11.1.1.5. GRUB2 - argumenti i opcije kernela u trenutku pokretanja sustava** (opcija `selinux=0`).

Kako bismo vidjeli trenutni status upotrebe *SELinux* sustava, možemo pozvati naredbu `sestatus` na sljedeći način:

```
sestatus

SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      32
```

Ovdje vidimo kako je *SELinux* aktivan (`enabled`) te da je trenutno odabrana metoda: `targeted`.
Promjenu između metoda: `enforcing` i `permissive` možemo raditi s naredbom `setenforce` na sljedeći način:

```
setenforce permissive
```

Dakle s navedenom promjenom bi se prebacili u `permissive` metodu rada, što ipak nećemo napraviti.

Odabirom `targeted` metode odnosno razine ili pravilnika *SELinux* zaštite imamo četiri oblika kontrole:

- ***Type Enforcement (TE)*** – ovo je primarni mehanizam zaštite pristupa odnosno kontrole.
- ***Role-Based Access Control (RBAC)*** je baziran na *SELinux* korisnicima (ne nužno istima kao korisnicima na sustavu) te njihovim zadanim pravilima.
- ***Multi-Level Security (MLS)*** nije obično u upotrebi te je u pravilu skriven u ovoj metodi.
- ***Multi-Category Security(MCS)*** je proširenje ***MLS*** razine s podrškom za virtualna računala i Linux kontejnere.



*Važno je razumjeti kako se tijekom pokretanja svakog programa (procesa), ako koristimo **SELinux** sustav, procesi pokreću sa **SELinux** sigurnosnim parametrima. Podsjetite se rada procesa u poglavlju: **9. Proces*** **menadžment.**



*Važno je razumjeti i kako se za svaku datoteku uz standardne i napredne ovlasti, ako koristimo **SELinux** sustav, zapisuju i **SELinux** sigurnosni parametri. Podsjetite se ovlasti:**4.3.*** **Ovlasti (*permissions* & *modes*) .**

Prvo pogledajmo stablo procesa (skratili smo ispis na njih samo par) s naredbom `ps` na sljedeći, novi način (`-Z` prekidač):

```
ps -aefZ

LABEL                                           UID   PID PPID C STIME TTY TIME     CMD
system_u:system_r:init_t:s0                    root   1    0    0 04:47  ?   00:00:01 /usr/lib/systemd/systemd
system_u:system_r:kernel_t:s0                  root   2    0    0 04:47  ?   00:00:00 [kthreadd]
system_u:system_r:sshd_t:s0-s0:c0.c1023        root  866   1    0 04:47  ?   00:00:00 /usr/sbin/sshd -D
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 1497 1480  0 05:17 ? 00:00:00 sshd: root@pts/0
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 1498 1497  0 05:17 pts/0 00:00:00 -bash
```

Ovdje s naredbom `ps` korištenjem prekidača (`-Z`) vidimo prvi (novi) stupac (`LABEL`) unutar kojega su vidljivi *SELinux* sigurnosni parametri s kojima je pojedini proces pokrenut.
Slično je i s ovlastima datoteka i direktorija (mapa), jer se osim standardnih ovlasti za datoteke i direktorije (mape) pohranjuju i *SELinux* ovlasti odnosno sigurnosne postavke.



Sve datoteke na sustavu imaju ***SELinux*** sigurnosni kontekst odnosno oznaku povezanu s njim. Ovaj kontekst pohranjen je u proširenim atributima datotečnog sustava. Problemi sa *SELinuxom* često proizlaze iz pogrešnog označavanja datotečnog sustava (tzv. *labeliranja*). To može biti uzrokovano pokretanjem računala s kernelom koji ne podržava *SELinux*. Ako vidite poruku o pogrešci koja sadrži `file_t`, to je obično dobar pokazatelj da imate ozbiljan problem sa *SELinux labeliranjem* datotečnog sustava. Najbolji način za ponovno *SELinux* labeliranje datotečnog sustava jest stvaranje datoteke `.autorelabel` u korjenskom direktoriju (`/`) te ponovno pokretanje računala.

***Ponovno (ručno) SElinux labeliranje datoteka***
Moguće je i ručno (ponovno) pokrenuti **SElinux** (re)labeliranje datotečnog sustava pomoću naredbe `fixfiles`:

```
fixfiles relabel
```

Nakon pokretanja ove naredbe pojaviti će se i pitanje za ponovno *SELinux labeliranje* direktorija `/tmp/` što nije preporučivo odnosno ne treba ga potvrditi. Naime budući da se `fixfiles` pokreće kao root, privremene datoteke koje se spremaju u ovaj direktorij, a na koje se oslanjaju određene aplikacije, biti će obrisane. To bi moglo učiniti sustav nestabilnim.

*SELinux* sigurnosne postavke nad datotekama i direktorijima možemo izlistati s naredbom `ls` korištenjem prekidača (`-Z`) pa stoga pogledajmo nekoliko datoteka (skratili smo ispis):

```
ls -alZ /etc/ssh/

drwxr-xr-x.   3 root root system_u:object_r:etc_t:s0    245 Feb  4 11:01 .
drwxr-xr-x. 110 root root system_u:object_r:etc_t:s0   8192 Apr 22 04:47 ..
-rw-r--r--.   1 root root system_u:object_r:etc_t:s0 577388 Feb  4 11:01 moduli
-rw-r--r--.   1 root root system_u:object_r:etc_t:s0   1716 Feb  4 11:01 ssh_config
drwxr-xr-x.   2 root root system_u:object_r:etc_t:s0     28 Feb  4 11:01 ssh_config.d
-rw-------.   1 root root system_u:object_r:etc_t:s0   4425 Feb  4 11:01 sshd_config
```

U petom stupcu sada vidimo *SELinux* sigurnosne postavke za direktorije (mape) i datoteke.
Prije nego krenemo dalje moramo se prvo upoznati sa osnovnim *SELinux* konceptima rada.

**SELinux korisnici (Engl. *Users*)**
*SELinux* ima skup unaprijed definiranih korisnika. Svaki korisnički račun u Linuxu povezan je na jednog ili više *SELinux* korisnika odnosno korisničkih računa. U Linuxu korisnik pokreće određeni program/aplikaciju (*proces*). Prema *SELinux* terminologiji, proces odnosno servis/demon ili pokrenuti program naziva se ***subjektom***. Pogledajmo kako vidjeti koje su poveznice između Linux korisnika i *SELinux* korisnika trenutno napravljene na našem sustavu, s naredbom `semanage`:

```
semanage login -l

Login Name           SELinux User         MLS/MCS Range        Service
__default__          unconfined_u         s0-s0:c0.c1023       *
root                 unconfined_u         s0-s0:c0.c1023       *
```

U prvom stupcu (`Login Name`) su Linux korisnici te njihova poveznica sa *SELinux* korisnicima u drugom stupcu (`SELinux User`). Naime korisnički računi Linuxa su standardno povezani sa *SELinux* posebnim korisnikom imena: **SELinux _default_ login**, koji se preslikava na *SELinux* korisnika `unconfined_u`. Dakle *SELinux* može ograničiti korisnike Linuxa da iskoriste sigurnosna pravila i mehanizme koji se na njih primjenjuju preslikavanjem Linux korisnika na *SELinux* korisnike koje naknadno možemo kreirati i povezivati s Linux korisnicima.

**SELinux uloge (engl. *Roles*)**
*SELinux* uloge (*role*) definiraju koji korisnici mogu pristupiti kojem procesu (pokrenutom programu/aplikaciji). Uloge nisu poput grupa, već više poput filtera: korisnik može koristiti ili preuzeti ulogu u bilo kojem trenutku, pod uvjetom da ima pravo na tu ulogu. Definicija uloge u *SELinux* politici definira koji korisnici imaju pristup određenoj ulozi. Ona također definira u domeni kojeg procesa se nalazi sâma uloga. Uloge se koriste i primjenjuju zato što *SELinux* implementira ono što se naziva ***Role-based Access Control (RBAC)*** mehanizam odnosno mehanizam pristupa temeljen na ulogama.

***SELinux subjekti i objekti***
Pod subjektom se podrazumijeva proces, a on može potencijalno utjecati na objekt.
Objekt u *SELinuxu* jest sve na osnovu čega se može djelovati. To može biti datoteka, direktorij (mapa), port, TCP *socket*, kursor/pokazivač i slično. Radnje koje subjekt može izvesti na objektu su definirane ovlastima subjekta.
Domena je kontekst unutar kojeg se može pokrenuti *SELinux subjekt* (proces). Kontekst govori procesu što može, a što ne može. Na primjer: domena će definirati koje su datoteke, direktoriji (mape), linkovi, uređaji ili portovi dostupni subjektu.

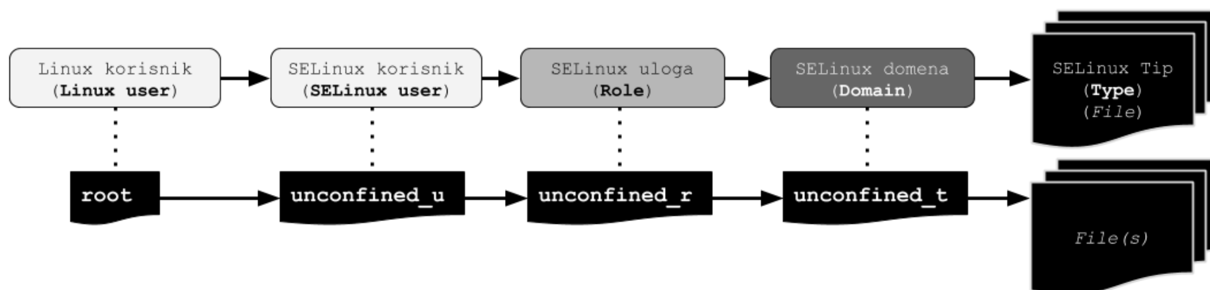
***Vrsta/Tip*** (Engl. *Type*) predstavlja kontekst za datoteke koji određuje svrhu datoteke. Na primjer; kontekst datoteke može nalagati da je to web-stranica ili da datoteka pripada nekom direktoriju ili da je vlasnik datoteke specifični *SELinux* korisnik. Kontekst datoteke naziva se njezinim *tipom* u *SELinux* terminologiji.


*SELinux* pravila definiraju pristup korisnika ulogama (*rolama*) te pristup uloga domenama, a u konačnici pristup domena takozvanim *tipovima* odnosno vrstama (***type***), što mogu biti: datoteke, direktoriji (mape) ili servisi.

To funkcionira tako, da se prvo korisnik mora ovlastiti za korištenje uloge (*role*), a potom mora biti ovlašten za pristup domeni. Domena je pri tome ograničena na pristup sâmo određenim vrstama datoteka. Sâma *SELinux* politika (Engl. *Policy*) se sastoji od pravila koja definiraju kako određeni korisnici mogu preuzimati sâmo određene uloge, a tim će ulogama biti odobren pristup sâmo određenim domenama.

Domene potom mogu pristupiti sâmo definiranim vrstama (*tipovima*) odnosno datotekama.
Slučaj u kojem proces koji se izvodi unutar određene domene može obavljati sâmo određene operacije na određenim vrstama objekata, naziva se ***Type Enforcement (TE)***.

Slika 262. Logička shema SELinux sustava i poveznice s Linux korisnicima.




Na slici 262 pogledajmo ove poveznice za Linux korisnika ***root***.
Za trenutno *logiranog* korisnika možemo vidjeti njegove pripadnosti *SELinux* elementima, s naredbom `id` na sljedeći način:

```
id -Z

unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Ono što ovdje vidimo je (logirani smo kao korisnik ***root***) da se mi kao Linux korisnik ***root*** povezujemo sa *SELinux* korisnikom (`unconfined_u`) kojem pripada uloga/rola (`unconfined_r`), a koja potom pripada domeni (`unconfined_t`).

Dodatno imamo još jedan pojam, a to je *SELinux* **razina** (Engl. *Level*). Ona je atribut ***MLS*** i **MCS** sustava. Raspon **MLS**-a čini nekoliko razina, koje se zapisuju kao: ***niža-viša*** razina, ako se razine razlikuju, ili sâmo ***niža*** razina, ako su razine identične. Primjerice oznaka razine ***s0-s0*** je isto što i razina ***s0***. Svaka razina sadrži par kategorija osjetljivosti, s time da su kategorije neobvezne. Ako postoje kategorije, razina se piše kao ***osjetljivost:kategorija.nîz*** ***(****Engl. Sensitivity:Category.Set****)***.
Ako nemamo kategorija, zapisuje se sâmo takozvana oznaka osjetljivosti. Ako je skup kategorija neprekidni nîz, može se i smanjiti. Na primjer: ***c0.c3*** je ista kao: ***c0, c1, c2, c3***. U konfiguracijskoj datoteci `/etc/selinux/targeted/setrans.conf` se sprema tablica prijevoda ***MCS-a*** za *SELinux* i preslikava nazive ovih razina u ljudski čitljivom formatu, kao što je recimo poveznica **s0:c0.c1023 = SystemHigh**.

***Ne uređujte navedenu datoteku pomoću uređivača teksta već pomoću naredbe*** `semanage`.

## Praćenje rada servisa i usluga

Kada se sustav pokrene s omogućenim *SELinuxom*, pravila se učitavaju u memoriju. *SELinux* pravila dolaze u modularnom formatu, slično kao što se kernel moduli učitavaju u vrijeme pokretanja sustava (ili kasnije). I baš poput kernel modula, mogu se dinamički dodavati i uklanjati iz memorije. Naredba `semodule -l` izlistat će nam sve *SELinux* module trenutno učitane u memoriju. Svaki *SELinux* modul je zadužen za određene funkcionalnosti, a moguće ih je podešavati prema našim potrebama.
Pogledajmo listu svih opcija; odnosno što je sve uključeno, a što isključeno od trenutno dostupnih opcija:

```
semanage boolean -l | less

SELinux boolean                State  Default Description
ssh_chroot_rw_homedirs         (off  ,  off)  Allow ssh to chroot rw homedirs
ssh_keysign                    (off  ,  off)  Allow ssh to keysign
ssh_sysadm_login               (off  ,  off)  Allow ssh to sysadm login
ssh_use_tcpd                   (off  ,  off)  Allow ssh to use tcpd
tftp_home_dir                  (off  ,  off)  Allow tftp to home dir
```

Ispis smo skratili na njih sâmo nekoliko, iako ih standardno imamo učitano na stotine. Tako primjerice zadnja opcija koju smo naveli (`tftp_home_dir`) definira ponašanje sustava kod praćenja rada ***TFTP*** servisa, na način da se dozvoljava (prema ***SELinux*** pravilima) ili ne dozvoljava upotreba ***TFTP*** servisa za *home* direktorij.
Sada pogledajmo samo opciju `tftp_home_dir` s naredbom `getsebool` na sljedeći način:

```
getsebool tftp_home_dir

tftp_home_dir --> off
```

Vidimo kako je ova opcija praćenja postavljena u stanje: isključeno (`off`) što znači kako je zabranjeno ovakvo ponašanje za ***TFTP*** servis, što opet znači kako će *SELinux* sustav prevenirati ovakvu upotrebu. Ako ipak želimo naložiti *SELinux* sustavu da ne prevenira ovo ponašanje (za ovu opciju), to možemo napraviti s naredbom `setsebool` na sljedeći način:

```
setsebool tftp_home_dir on
```

## Rad s korisničkim računima, datotekama i procesima

Prvo pogledajmo tablicu sa standardno kreiranim *SELinux* korisnicima (**ne Linux korisnicima**) na sustavu:

| ***SELinux*** **korisnik** | **Opis** | **Za što se koristi** |
|---|---|---|
| `unconfined_u` | Ovaj korisnik namijenjen je neograničenim korisnicima. Ovi korisnici jedva da imaju ograničenja u *SELinux* kontekstu i namijenjeni su sustavima u kojima sâmo određene usluge trebaju biti zabranjene. | Obično za sve korisnike sustava (standardno postavljeno). |

| *SELinux* korisnik | Opis | Za što se koristi |
|---|---|---|
| root | *SELinux* korisnik za *root* (*Admin*) korisnika. | Za korisnike s *administratorskim* pravima odnosno privilegijama rada. |
| sysadm_u | *SELinux* korisnik za administracijske uloge (*role*). | Samo za one korisnike koji trebaju obavljati sistemske poslove. |
| staff_u | *SELinux* korisnik za administracijske uloge (*role*) ali i neadministracijske. | Dobar dio Linux korisnika s osnovnim i ponekim administracijskim pravima. |
| user_u | *SELinux* korisnik za neadministracijske uloge (*role*). | Linux korisnici s ograničenjima. |
| system_u | *SELinux* korisnik za rad sa sistemskim servisima. | Posebni Linux korisnici koji rade sa servisima. |

Osim navedenih standardnih *SELinux* korisnika, moguće je i prema potrebi kreirati i svoje (dodatne) korisnike, mada to u većini slučajeva nije potrebno, jer su s navedenima pokrivena sva važnija područja primjene.

Pogledajmo i koje su standardno kreirane *SELinux* uloge (***role***) i koje je njihovo značenje:

| *SELinux* uloga (rola) | Opis |
|---|---|
| user_r | Ovo je standardna uloga (za „obične“ korisnike) koja daje dozvole za aplikacije i ne privilegirane domene. |
| staff_r | Slična kao user_r, ali dozvoljava pristup na više informacija sustava od običnog korisnika te omogućuje prebacivanje na druge uloge. |
| sysadm_r | Za administraciju sustava, omogućuje pristup svim domenama uključujući i one privilegirane. |
| system_r | Sistemska uloga, obično za potrebe servisa. |

A sada pogledajmo sve *SELinux* korisnike, razine i uloge koje imamo definirane na našem sustavu:

```
semanage user -l

                Labeling   MLS/       MLS/
SELinux User    Prefix     MCS Level  MCS Range          SELinux Roles
guest_u         user       s0         s0                  guest_r
root            user       s0         s0-s0:c0.c1023      staff_r sysadm_r system_r
unconfined_r
staff_u         user       s0         s0-s0:c0.c1023      staff_r sysadm_r system_r
unconfined_r
sysadm_u        user       s0         s0-s0:c0.c1023      sysadm_r
system_u        user       s0         s0-s0:c0.c1023      system_r unconfined_r
unconfined_u    user       s0         s0-s0:c0.c1023      system_r unconfined_r
user_u          user       s0         s0                  user_r
xguest_u        user       s0         s0                  xguest_r
```

Instalirajmo i dodatne programske pakete koji će nam kasnije biti od koristi:

```
yum -y install setools-console policycoreutils-newrole policycoreutils-devel
```

Ako želimo provjeriti koje su domene dozvoljene za određenu *SELinux* ulogu (**role**) to možemo napraviti s novom naredbom (koju smo upravo instalirali). Ako primjerice želimo provjeriti kojim domenama je dozvoljen pristup za ulogu sysadm_r:

```
seinfo -r sysadm_r -x
```

I vidjet ćemo cijeli nîz domena kojima ova uloga (sysadm_r) ima pravo pristupati.

**Kreirajmo novog *SELinux* korisnika korisnici_u koji će koristiti ulogu (rolu) imena staff_r.**

```
semanage user -a -R "staff_r" korisnici_u
```

**Kreiranje novog povezivanja (*mapiranja*) SELinux korisnika s Linux korisnicima**

Povežimo novog korisnika hrvoje sa *SELinux* korisnikom korisnici_u koji ima i administracijska i ne administracijska prava jer on pripada ulozi staff_r.

```
semanage login -a -s korisnici_u hrvoje
```

Nakon nekoliko trenutaka imat ćemo novo stanje.

**Kreiranje novog Linux korisnika uz dodjeljivanje poveznice sa SELinux korisnikom.**

Kreirat ćemo novog Linux korisnika imena pero i povezati ga sa *SELinux* korisnikom: user_u.

```
useradd -Z user_u pero
```

Korištenjem prekidača (-Z) uz naredbu useradd smo postigli navedeno povezivanje sa *SELinux* korisničkim računom.

Sada uđimo u korisnički direktorij našeg novo kreiranog korisnika `pero` te ispišimo njegov sadržaj i ***SELinux*** postavke:

```
cd /home/pero/
ls -alZ

drwx------. 2 pero pero user_u:object_r:user_home_dir_t:s0   83 Apr 23 13:03 .
drwxr-xr-x. 4 root root system_u:object_r:home_root_t:s0     32 Apr 23 13:03 ..
-rw-------. 1 pero pero unconfined_u:object_r:user_home_t:s0 11 Apr 23 13:03 .bash_history
-rw-r--r--. 1 pero pero user_u:object_r:user_home_t:s0       18 Nov  8 17:21 .bash_logout
-rw-r--r--. 1 pero pero user_u:object_r:user_home_t:s0      141 Nov  8 17:21 .bash_profile
-rw-r--r--. 1 pero pero user_u:object_r:user_home_t:s0      312 Nov  8 17:21 .bashrc
```

U trećem (označenom) stupcu sada vidimo i *SELinux* oznake.

Sada se iz korisnika `root` prebacimo u korisnika `pero` te kao on ovdje kreirajmo jednu datoteku imena `test.txt`:

```
su – pero
cd /home/pero/
echo 1234 > test.txt
ls -alZ

unconfined_u:object_r:user_home_t:s0 test.txt
```

Vidimo da je novo kreirana datoteka sa *SELinux* ovlastima: `unconfined_u:object_r:user_home_t:s0`. To znači kako je *SELinux* korisnik: `unconfined_u` iako smo za Linux korisnika `pero` rekli da pripada korisniku *SELinux* sustava: `user_u`. *SELinux* uloga mu je za ovu datoteku: `object_r`, a kontekst odnosno tip je: `user_home_t` koji se pak odnosi na korisničke direktorije (Engl. *Home*). Na kraju vidimo i: `s0` što označava ***SELinux MLS***. To je zato što su tijekom kreiranja *home* direktorija kreirane i datoteke koje smo vidjeli kod prvog ispisa *home* direktorija, a potom, kada smo postali korisnik `pero` te kreirali novu datoteku, za nju se nasljeđuju samo *SELinux* uloga i kontekst (`object_r` i `user_home_t`).

> ***SELinux** domena odnosno tip se nazivaju i **kontekstima** koji u grubo mogu biti za: datoteke (u ovom slučaju) ili procese.*

Ako naknadno želimo promijeniti *SELinux* ovlasti za ovu datoteku, to možemo napraviti na sljedeći način:

```
semanage fcontext -a -t user_home_t -s user_u /home/pero/test.txt
```

Sve ove promjene se zapisuju, za svaku datoteku zasebno, u *SELinux* konfiguracijsku datoteku: `/etc/selinux/targeted/contexts/files/file_contexts.local` koja će sada imati ovaj novi unos:

```
/home/pero/test.txt    user_u:object_r:user_home_t:s0
```



*U slučaju kada smo promijenili SELinux ovlasti za neku datoteku i kasnije ih želimo vratiti u izvorno stanje (za one datoteke za koje SELinux ima definirane ovlasti), to možemo napraviti s naredbom:* `restorecon`.

Listu svih *SELinux* ovlasti definiranih za sve datoteke na sustavu, možemo dobiti s naredbom `semanage`:

```
semanage fcontext -l
```

*SELinux* konteksta za datoteke imamo definiran veliki broj, svaki prema svojoj namjeni. Sve ih možemo vidjeti s naredbom:

```
seinfo -t
```

Dobit ćemo izlistanje njih obično oko 5.000 ili više. Nadalje moguće je i nasljeđivanje konteksta, koje se normalno i događa, kako smo vidjeli kod kreiranja novih datoteka, jer se konkretno naslijedio kontekst `user_home_t` što je uobičajeno.

Međutim moguće je da proces u jednoj domeni prelazi na drugu domenu izvršavanjem aplikacije koja ima posebnu vrstu ulazne točke (***entrypoint type***) [(901)] za drugu domenu. Dopuštenje ulazne točke koristi se u *SELinux* pravilima i kontrolira koje se aplikacije mogu koristiti za ulazak u koju domenu.

Sljedeći primjer prikazuje prelazak iz jedne domene u drugu. Svaki puta kada bilo koji korisnik želi promijeniti lozinku, on za to mora pozvati naredbu `passwd`. Pogledajmo naredbu `passwd` kako ona (kao datoteka) izgleda što se tiče *SELinux* postavki, ne gledajući na Linux ovlasti koje prethode *SELinux* ovlastima, a u ovom slučaju su postavljene na *SET UID*:

```
ls -Z /usr/bin/passwd
system_u:object_r:passwd_exec_t:s0 /usr/bin/passwd
```

Naime kada pozivamo naredbu `passwd` (tj. datoteku `/usr/bin/passwd`) i mijenjamo svoju lozinku, ona mora mijenjati unos u datoteci u koju se spremaju lozinke, a to je datoteka: `/etc/shadow`. Pogledajmo sâmo *SELinux* ovlasti ove datoteke:

```
ls -Z /etc/shadow
system_u:object_r:shadow_t:s0 /etc/shadow
```

Ovdje vidimo sljedeće:

- Naredba passwd odnosno datoteka: /usr/bin/passwd ima *SELinux* ovlasti na razini *SELinux* domene odnosno konteksta: passwd_exec_t.
- Datoteka u koju se upisuju lozinke: /etc/shadow ima *SELinux* ovlasti na razini *SELinux* domene odnosno konteksta: shadow_t.

Ako želimo vidjeti koje sve datoteke imaju ova dva *SELinux* konteksta, to možemo vidjeti s naredbom semanage:

```
semanage fcontext -l | grep -i -e passwd_exec_t -e shadow_t
```

Konkretno nas zanimaju sljedeća dva unosa (iz ispisa naredbe od gore), koja to potvrđuju:

```
/etc/shadow.*          regular file       system_u:object_r:shadow_t:s0
/usr/bin/passwd        regular file       system_u:object_r:passwd_exec_t:s0
```

Nadalje za domenu/kontekst passwd_exec_t možemo vidjeti s naredbom sesearch (skratili smo ispis):

```
sesearch -T -t passwd_exec_t

type_transition accountsd_t passwd_exec_t:process passwd_t;
type_transition guest_t passwd_exec_t:process passwd_t;
type_transition staff_t passwd_exec_t:process passwd_t;
type_transition sysadm_t passwd_exec_t:process passwd_t;
```

Dakle passwd_exec_t kontekst se veže sa kontekstom: passwd_t, koji je očito poveznica.

Pronađimo vezu s njim preko *SELinux* pravila:

```
sesearch --allow | grep -i -e shadow_t -e passwd_exec_t |grep "^allow passwd_t"

allow passwd_t passwd_exec_t:file { entrypoint execute getattr ioctl lock map open
read }; allow passwd_t shadow_t:file { append create getattr ioctl link lock map open
read relabelfrom relabelto rename setattr unlink write };
```

U ovom primjeru je vidljivo kako program odnosno naredba passwd koja ima *SELinux* kontekst: passwd_exec_t ima poveznicu (entrypoint) preko passwd_t konteksta/domene.
U drugom retku vidimo *SELinux* pravilo koje dozvoljava za drugi kontekst/domenu shadow_t, koju koristi datoteka /etc/shadow, da koristi passwd_t kontekst/domenu.
Na ovaj način je napravljena veza između *SELinux* domena preko jedne zajedničke domene/konteksta: passwd_t.

## *SELinux* i mrežni portovi

Osim navedenog praćenja programa i datoteka *SELinux* nam nudi i mogućnost praćenja mrežnih portova, prema ***TCP***, ***UDP*** ili ***SCTP*** transportnim protokolima. Sa sljedećom naredbom možemo vidjeti koje sve transportne protokole i na kojim portovima ih *SELinux* sustav prati. Skratili smo ispis, ali vidljivo je da je s lijeve strane identifikator (***Tip***), a desno protokol i broj porta:

```
semanage port -l

SELinux Port Type              Proto    Port Number
ssh_port_t                     tcp      22
dns_port_t                     tcp      53, 853
dns_port_t                     udp      53, 853
dhcpd_port_t                   udp      67, 547, 548, 647, 847
dhcpc_port_t                   tcp      68, 546, 5546
dhcpc_port_t                   udp      68, 546, 5546
tftp_port_t                    udp      69
http_port_t                    tcp      80, 81, 443, 488, 8008, 8009, 8443, 9000
snmp_port_t                    tcp      199, 1161, 161-162
. . .
```

Svaki port u konačnici se identificira sa (od lijevo na desno u prikazu):

- SELinux Port Type je vrsta porta odnosno identifikator protokola. Primjerice ssh_port_t označava ***SSH*** servis.
- Proto je vrsta transportnog protokola u upotrebi. Primjerice tcp označava ***TCP*** transportni protokol.
- Port Number je broj porta, protokola u upotrebi.

Naime *SELinux* sustav prema definiranoj listi za svaki mrežni servis prije nego se servis (*daemon*) pokuša pokrenuti, prvo provjerava njegova prava korištenja i otvaranja portova na određenom transportnom protokolu, a koja se definiraju ovdje.
Ako mrežni servis želi koristiti transportni protokol i/ili port koji nije ovdje definiran, to će mu biti **onemogućeno**.
Poveznica *SELinux* ***tipa*** odnosno domene/konteksta i u konačnici servisa (*daemona*) je vidljiva s naredbom: sesearch.
Pogledajmo primjerice naš pokrenuti ***SSH*** (sshd) servis (skratili smo ispis):

```
ps -efZ |grep sshd

system_u:system_r:sshd_t:s0-s0:c0.c1023 root 873    1  0 08:53 ?        00:00:00
/usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-
poly1305@openssh.com,aes256-ctr,aes256-cbc,aes128-gcm@openssh.com,. . .,ssh-rsa
```

Vidimo kako je njegova domena (kontekst): sshd_t koja je zapravo povezana sa SSH servisom ssh_port_t u *SELinux* sustavu. Sada nećemo detaljno pronalaziti ovu poveznicu, ali važno je znati kako ona naravno postoji.
Prema imenu mrežnog *SELinux* konteksta u svakom slučaju možemo zaključiti o kojem se protokolu odnosno servisu radi.

Dakle u našem slučaju za ***SSH*** servis je to kontekst imena: `ssh_port_t` za koji smo iz ispisa sa stranice prije vidjeli kako se veže na TCP port broj 22.

**Dodavanje novog porta postojećem servisu**
Što ako recimo za ***SSH*** servis želimo kreirati još jedan port, recimo konkretno za ***SSH*** servis, još jedan TCP port broj 22222.

Ovaj novi port ćemo u *SELinux* sustav dodati na sljedeći način:
```
semanage port -a -t ssh_port_t -p tcp 22222
```

I sada provjerimo ovu promjenu:
```
semanage port -l | grep ssh
ssh_port_t                     tcp      22222, 22
```
Vidimo da je novi port `22222` dodan, i tek sada ga možemo konfigurirati unutar ***SSH*** servisa.
Stoga otvorimo datoteku: `/etc/ssh/sshd_config` i pronađimo redak koji izgleda ovako:
```
#Port 22
```

Te ga promijenimo u:
```
Port 22
Port 22222
```
Port 22 je podrazumijevani te se on ne mora inače navoditi pa je stoga i zakomentiran sa znakom #, iako ga mi dodajemo.

Snimimo promjenu na našoj konfiguracijskoj datoteci te restartajmo SSH servis:
```
systemctl restart sshd
```
Pogledajmo jesu li se sada otvorila oba TCP porta za SSH servis: port 22 i port 22222:
```
netstat -tunap | grep sshd
tcp        0      0 0.0.0.0:22222     0.0.0.0:*         LISTEN      5430/sshd
tcp        0      0 0.0.0.0:22        0.0.0.0:*         LISTEN      5430/sshd
```
Vidimo da jesu (`0.0.0.0:22` i `0.0.0.0:22222`), što znači kako je *SELinux* sustav to dozvolio.

Ako u nekom trenutku sumnjate da se određeni portovi u SELinux pravilima preklapaju, to možete jednostavno provjeriti. Provjerimo koja sva pravila uključuju port 22:
```
sepolicy network -p 22

22: tcp ssh_port_t 22
22: udp reserved_port_t 1-511
22: tcp reserved_port_t 1-511
22: sctp reserved_port_t 1-511
```
Dakle u prvom retku ispisa vidimo kako se TCP port 22 koristi za: `ssh_port_t` što indirektno pokazuje na naš SSH servis, a u sljedećim redovima vidimo kako je port 22 unutar opsega rezerviranih portova od broja 1 do 511 (`reserved_port_t`) što je normalno jer su u tom opsegu portova sistemski servisi.
Nadalje vidimo i kako je taj opseg rezerviranih portova za transportne protokole ***UDP***, ***TCP*** i ***SCTP***.

Ako bilo kada želimo maknuti novo dodati TCP port 22222 iz liste portova koji su dozvoljeni za SSH servis, to možemo napraviti na sljedeći način:
```
semanage port -d -t ssh_port_t -p tcp 22222
```
Nakon toga ne zaboravite napraviti promjene u konfiguracijskoj datoteci SSH servisa (`/etc/ssh/sshd_config`).



***Ako koristite vatrozid (Engl. Firewall) sve promjene morate postaviti i na njemu.***



*Naučili smo kako **SELinux** sustav uvodi dodatnu razinu sigurnosti s posebnim ovlastima na datoteke i programe. S njime se dobiva znatno finija granulacija nadzora i kontrole, kako nad izvršne ili druge datoteke, tako i na programe odnosno servise, ali i mrežne protokole.*



Druge distribucije Linuxa poput ***SUSE*** (i ***Open SUSE***) te ***Ubuntu*** Linuxa koriste drugačiju, ali sličnu tehnologiju ***SELinuxu*** koja se zove ***AppArmor***. Dodatno postoje i druge ovakve tehnologije poput: Smack i Tomoyo Linux.

**Izvori informacija: (895),(896),(897),(898),(899),(900),(901),(902),** **`man sestatus`**, **`man setenforce`**, **`man semanage`**, **`man semodule`**, **`man getsebool`**, **`man setsebool`**, **`man seinfo`**, **`man restorecon`**, **`man 5 selinux_config`**, **`man 5 semanage.conf`**, **`man fixfiles`**, **`man selinux`**.

# 28.3. Sigurnosne preporuke

Svakodnevnom upotrebom Linux računala za osobne potrebe, a posebno kao poslužitelja, moramo biti svjesni koliko naše dnevne aktivnosti na računalu ili korisnika na poslužitelju, mogu uzrokovati razne probleme. Osim toga svaki poslužitelj u upotrebi je potencijalno ranjiv, što zbog činjenice da se na njemu nalaze razni servisi koji se neredovito zakrpavaju sigurnosnim i drugim zakrpama, a što zbog toga što su nepravilno konfigurirani. Zbog svega navedenog, a poglavito zbog sigurnosti, osim sigurnosnih nadogradnji sustava koje je obično jednostavno napraviti, posebnu pažnju treba posvetiti konfiguraciji servisa, ali i njihovog odabira i primjene, o čemu ćemo govoriti u ovoj cjelini.



Sigurnosne i druge nadogradnje svih programskih paketa instaliranih s aktivnih repozitorija na ***RedHat/CentOS*** baziranim distribucijama Linuxa u pravilu se odrađuju pozivanje naredbe: `yum -y update`.

**Za detalje pogledajte poglavlje: 7.2.2.1. Rad s *YUM-om*.**



**Sve navedene primjere prvo testirajte u svom testnom laboratoriju prije primjene na živom sustavu!.**

**1. Postavke BIOS-a računala/poslužitelja.**
Krenuvši od BIOS-a računala, ako želimo podići razinu sigurnosti, preporučuje se definirati administratorsku lozinku za ***BIOS***, kako bi se onemogućile maliciozne promijene u *BIOSu* računala.
Dodatno, ako vaš poslužitelj ima ugrađenu takozvanu menadžment mrežnu karticu (***IPMI/ILOM/iLO/iDRAC/iRAC***) potrebno ju je staviti u izoliranu mrežu ili **VLAN** mrežu. Nadalje za nju treba definirati snažniju lozinku te uključiti snimanje poruka o spajanju na taj sustav. Važno je i redovito raditi sigurnosne (i druge) nadogradnje *BIOS*-a.

**2. Zaštitite GRUB2 boot loader.**
Prema potrebi zaštitite ***GRUB2*** *boot loader* (menadžer) jer u slučajevima kada se sustav podiže u **init 1** način rada (*recovery*), osoba koja se nalazi ispred računala/poslužitelja ima prava raditi sve promjene na sustavu. Kao i bilo tko, tko se nalazi ispred računala tijekom pokretanja sustava, što u nekim slučajevima nije dozvoljeno.
Ako želimo ovu višu razinu sigurnosti, pokrenite naredbu koja će snimiti lozinku za ***grub2*** *boot loader* (menadžer):

```
grub2-setpassword
```

Potom upišite lozinku i potvrdite ju. Sustav će tada kreirati kriptiranu lozinku i snimiti ju u datoteku: `/boot/grub2/user.cfg`. ***GRUB2*** je sada zaštićen od promjena, odnosno za njih će tražiti lozinku.

**3. Definirajte više particija na (sistemskom) disku.**
Tijekom instalacije operativnog sustava treba voditi računa o tome da se tijekom particioniranja diska kreiraju minimalno odvojene particije za:

- *Root* (vršni) odnosno sistemski datotečni sustav: `/`
- Korisničke direktorije (mape) unutar: `/home` kako korisnici ne bi mogli prepuniti vršni datotečni sustav.
- Te po mogućnosti za: `/usr`, `/var`, `/tmp` i `/opt` kako korisnički (ne sistemski) i privremeni programi ne bi mogli prepuniti vršni odnosno sistemski datotečni sustav.

**4. Pravilan odabir servisa i njihov rad.**
Prvo pravilo za pokretanje mrežnih servisa (pr. *Web* servis) je da sâm servis ne bi smio biti pokrenuti s ovlastima *root* korisnika.



Pogledajte poglavlja:
**4.4.3. To nije sve: mogućnosti izvršavanja datoteka i programa (*capabilities*).**
**7.3.2.1. Pokretanje servisa** sa **`USER=` i `GROUP=`**.

Pravilan odabir servisa koje koristimo se odnosi na činjenicu kako neki servisi i protokoli nisu dobri za upotrebu što se sigurnosti tiče. Pa tako umjesto:

- Protokola za udaljeno spajanje i kopiranje poput: `telnet`, `rsh`, `rlogin`, `rcp` ili `ftp` koristimo modernije i sigurnije varijante poput: `ssh`, `scp` i `sctp`.
- Web poslužiteljskih servisa koji koriste ***HTTP*** protokol zamijenite s ***HTTPS*** protokolom, gdje god je to moguće.
- Koristite li ***VPN*** za udaljeno spajanje na mrežu (tvrtke i sl.) ne koristite više zastarjele i nesigurne protokole poput ***PPTP*** ili ***L2TP***, već novije i sigurnije varijante poput: ***IPSec***, ***OpenVPN*** ili ***WireGuard***.
- Ako šaljete važne dokumente drugoj strani, koristite kriptografsko potpisivanje i kriptiranje podataka, upotrebom GNU PG (***PGP***) ili sličnih alata.

**5. Deinstaliranje tehnološki (sigurnosno) zastarjelih servisa.**
Deinstalirajte zastarjele servise poput: `telnet`, `tftp`, `rsh`, `rlogin`, `rcp`, ako ih nužno ne koristite, a trebali bi ih ionako zamijeniti s novijim i sigurnijim poput ***SSH***. Obratite pažnju na to da se `tftp` smatra nesigurnim, pa ga barem zaštitite za lokalnu mrežu, ako se baš mora koristiti. Deinstalaciju svih navedenih programa možete postići sa sljedećom naredbom:

```
yum erase xinetd ypserv tftp-server telnet-server rsh-server
```

**6. Provjera aktivnih i deinstalacija nepotrebnih servisa (i programa).**
Provjerite koje sve aktivne servise imate odnosno koji su sve pokrenuti te deinstalirajte one nepotrebne. Prvo provjerite koje sve mrežne servise imate aktivno pokrenute (u stanju *LISTEN*). U krajnjem desnom stupcu ispisa će biti vidljivo ime servisa:

```
netstat -tunalp
```

Odnosno isto možete provjeriti i sa sljedećom naredbom:

```
ss -tunlp
```

Provjerite i koji se sve servisi (*daemoni*) pokreću na računalu (oni nisu nužno svi mrežni) [za ***RedHat/CentOS 7.+***]:

```
systemctl --type=service --state=active
```

One za koje ste potpuno sigurni da ih ne želite pokretati automatski, isključite ih.

Dodatno (za napredne korisnike) ispišite sve instalirane programske pakete i provjerite jesu li vam svi oni potrebni (oprez):

```
yum list installed
```

**7. Radite redovitu nadogradnju (ažuriranje) svih programa.**
Kao što smo već spomenu, ako ste programe instalirati s dostupnih repozitorija, redovito radite njihove nadogradnje (što se odnosi i na cijeli sustav). To možete postići sa sljedećom naredbom:

```
yum -y update
```

Za sve programe koje ste sami kompilirali ili kopirali, morate uraditi ručne nadogradnje/zakrpe.

**8. Radite redovite nadogradnje *BIOS*-a i *Firmware*-a svôg hardvera:** RAID i disk kontrolera, mrežnih kartica i slično.

**9. Omogućite automatsko ažuriranje sustava gdje je to moguće**.

**10. Koristite SELinux ako je to moguće (o njemu detaljnije u poglavlju: 28.2. SELinux sustav).**
Po mogućnosti uključite ***SELinuxu*** sustav. Pošto smo o njemu već govoriti, ovdje ga nećemo detaljnije objašnjavati.

**11. Ovlasti tijekom kreiranja novih datoteka.**
Važno je provjeriti standardnu masku za ovlasti (*umask*) tijekom kreiranja novih datoteka jer, ako je ona primjerice **000** to znači da se nove datoteke kreiraju s pravima zapisivanja za sve korisnike. Osigurajmo da je maska postavljena barem na **022**.
Otvorite datoteku: `/etc/init.d/functions` te pronađite unos: `umask` te postavite vrijednost na **`022`** (ovo je važeća datoteka za servise).
Druga datoteka je: `/etc/profile` i ona se primjenjuje za korisnike kada se logiraju, pa u njoj pronađite postavljanje *umask* vrijednosti za normalne korisnike (***UID*** > 199), na **022 (**ako to već nije postavljeno**)**.

**12. Ovlasti *home* direktorija (mape) te novih korisnika.**
Provjerite *home* direktorije (mape) korisnika i njihove ovlasti. Svaki *home* direktorij mora imati za vlasnika, svog vlasnika, a minimalne ovlasti moraju biti **755** (`rwxr-xr-x`) ili znatno restriktivnije.
Kod kreiranja novog korisnika i njegovog *home* direktorija, čitaju se ovlasti definirane u datoteci: `/etc/login.defs`, pod `umask` unosom, koji je standardno postavljen na **022** (ako nije definiran), a inače je obično postavljen na **077** što je restriktivnije odnosno još sigurnije. *Umask* i efektivna vrijednost za datoteke i direktorije koje ona postavlja, vidljiva je u poglavlju: **4.4.1. Ovlasti kod kreiranja nove datoteke ili direktorija i naredba *umask***. Dodatno unutar *home* direktorija korisnika provjerite ima li nekih ovlasti koje eventualno ne bi trebale biti postavljene; tako primjerice skriveni direktorij `.ssh` unutar *home* direktorija korisnika mora imati maksimalno ovlasti ***700*** (`rwx------`) jer se u njemu čuvaju SSH ključevi za udaljeno spajanje na druge sustave na mreži, za konkretnog korisnika.
Nadalje konfiguracijske datoteke koje se učitavaju kod inicijalizacije ljuske (*shella*), poput: `.bash_profile`, `.bashrc` i drugih moraju imati postavljene ovlasti najbolje na **644** (rw-r--r--).

**13. Ovlasti za sistemske konfiguracijske te druge datoteke i direktorije (mape).**
Provjerite ovlasti za sistemske konfiguracijske datoteke unutar direktorija `/etc/` čiji vlasnik u pravilu treba biti *root* korisnik, te prava pisanja u pravilu također trebaju biti za *root* korisnika.
Direktorij `/tmp` u pravilu ima ovlasti s kojima svi korisnici imaju prava pisanja i čitanja. Stoga bilo koji korisnik može stvoriti datoteke u ovom direktoriju (mapi). Zbog istih prava korisnik može stvoriti simboličke veze (*soft link*) u toj mapi s bilo kojim nazivom, usmjeravajući ih na sistemske datoteke. Ako aplikacije s administrativnim pravima u ovoj mapi pišu u privremene datoteke, a imena datoteka su poznata ili ih je lako pogoditi, tada aplikacija može prebrisati sistemske datoteke putem takve veze. Pri tome modifikacija sistemskih datoteka može stvoriti ogromne probleme u sustavu. Na sljedeći način možete pronaći sve datoteke unutar direktorija `/tmp` koje imaju poveznice te imaju administratorska prava:

```
find /tmp -type l -and -lname '/*' -or -lname '*..*'
```

Nadalje datoteke koje imaju postavljene bîtove (ovlasti): ***Set UID*** i ***Set GID*** mogu predstavljati sigurnosni problem, ako nisu regularne. Stoga pratite sve datoteke na sustavu koje imaju postavljene ove bîtove odnosno pojavljivanje novih datoteka s njima.
Vezano za ***Set UID*** i ***Set GID*** pogledajte poglavlje: **4.3.1. Što nam govore ovlasti**.

Pogledajmo kako pronaći sve datoteke koje imaju postavljene ove ovlasti (bîtove), na sljedeći način:

```
find / -type f \( -perm -2000 -o -perm -4000 \) -print
```

Najbolje je to napraviti nakon instalacije svih programa, a prije puštanja u rad poslužitelja, te svako malo uspoređivati listu.

**14. Druge nedozvoljene ovlasti nad datotekama.**
Provjerimo imamo li na sustavu direktorije na koje svi (Engl. *All/Others*) imaju prava pisanja, a nije postavljen takozvani ***sticky bît*** (opisan u poglavlju: **4.3. Ovlasti (*permissions & modes*)**). To bi značilo da datoteke unutar tog direktorija može mijenjati bilo tko, što pogotovo za *Web* i druge poslužitelje znači veliki sigurnosni problem.

Provjerimo ovakve ovlasti za cijeli vršni direktorij `/webroot` (za *Web* poslužitelj):

```
find / -xdev -type d \( -perm -0002 -a ! -perm -1000 \) -print
```

Svaki pronađeni od njih dobro proučite i promijenite mu ovlasti pravilno.

Provjerimo imamo li neke datoteke kojima ne postoji definirani vlasnik ili grupa:

```
find / -xdev \( -nouser -o -nogroup \) -print
```

Ovakvih datoteka ne bi smjeli imati.

**15. Ograničite pristup `/proc` datotekama kako biste spriječili curenje informacija.**
To možemo postići pronalaskom **`/proc`** unosa u datoteci `/etc/fstab`. Poput:

```
proc /proc proc defaults,hidepid=2 0 0
```

Pri tome kod varijable `hidepid` vrijednost 1 skriva samo podatke o procesima drugih korisnika, ali i dalje omogućuje vidljivost PID-ova, dok vrijednost 2 sakriva informacije o svim procesima osim od samih korisnika vlasnika i *root* korisnika.

## Povećajte razinu sigurnosti vezanu za korisničke račune!

**16. Povećajte razinu sigurnosti vezanu za korisničke račune Linuxa.**
Prvenstveno za korisnike, ali po potrebi i za administratora (*root* korisnik) možete smanjiti vrijeme trajanja lozinke odnosno vijeka trajanja lozinki za korisnika(e). To možete raditi za jednog po jednog ili više korisnika. Ako za recimo korisnika ***pero*** želite postaviti pravilo da lozinku mora mijenjati svakih 30 dana te da upozorenje dobije sedam dana prije isteka, to možete napraviti sa sljedećom naredbom (ili ručnom promjenom u datoteci: `/etc/shadow`):

```
chage -M 30 -W 7 pero
```

**17.** Nadalje onemogućite upotrebu već korištenih (starih) lozinki. To možemo postići na sljedeći način, upotrebom ***PAM*** servisa/funkcionalnosti. Otvorite dvije *PAM* konfiguracijske datoteke: `/etc/pam.d/password-auth` i `/etc/pam.d/system-auth` te u obje nakon linije odnosno retka:

```
password    requisite                   pam_pwquality.so try_first_pass local_users_only
```

Dakle, nakon gornjeg retka, dodajte sljedeći redak:

```
password    requisite                   pam_pwhistory.so remember=5 use_authtok
```

S ovime ćemo ograničiti korisnika da ne može koristiti stare lozinke jer će ih sustav pratiti do zadnjih pet (5).

**18. Ograničite trajanje lozinki**: postavite rok trajanja lozinki u `/etc/login.defs` (npr. `PASS_MAX_DAYS`).

**19. Koristite ograničenja resursa za korisnike**: Uvedite ograničenja (limite) koristeći: `ulimit` ili `cgroups`.

**20. Automatsko zaključavanje korisničkog računa nakon neaktivnosti**: konfigurirajte zaključavanje sesija putem sistemske (Shell) varijable `TMOUT`.

**21. Zaključajte korisničke račune koji se trenutno ne koriste na sustavu.**
Korisnički računi koji se trenutno ne koriste na sustavu, a mogli bi se koristiti ponovno, zaključajte.

Ako se radi o korisniku *pero*, njegov korisnički računa kao *root* korisnik možemo zaključati na sljedeći način:

```
passwd -l pero
```

Trebamo li kasnije otključati isti korisnički račun, to možemo napraviti sa:

```
passwd -u pero
```

Provjerite imate li na sustavu korisnika koji ima praznu lozinku; što ćete moći pronaći sa sljedećom naredbom:

```
cat /etc/shadow | awk -F: '($2==""){print $1}'
```

Postoji li takav korisnik, biti će vam ispisan.

## SUDO

**22.** Onemogućite logiranje odnosno spajanje na sustav direktno kao *root* (Administratorski) korisnik, ako je to primjenjivo.
Za ovu potrebu ćemo kreirati korisnika ***hrvoje*** kojem ćemo preko `sudo` naredbe dati mogućnost da dobije *root* ovlasti.
Kreirajmo novog korisnika koji će imati administratorske ovlasti:

```
adduser hrvoje
```

Potom omogućimo našem novom korisniku `sudo` prava, dodavanjem slijedećeg retka u datoteku: `/etc/sudoers` i to na sljedeći način:

```
echo 'hrvoje ALL=(ALL) ALL' >> /etc/sudoers
```

**Redovito provjeravajte koji su sve korisnici navedeni u datoteci `/etc/sudoers`.**

Nakon ovog koraka možemo pomoću naredbe `sudo` pokretati programe sa *root* ovlastima, poput:

```
sudo ifconfig
```

**23.** Provjerite ima li neki drugi korisnik osim korisnika *root* ***UID*** broj nula (0), što bi predstavljalo veliki sigurnosni problem. To možemo provjeriti s naredbom koja bi nam trebala ispisati samo *root* korisnika:

```
awk -F: '($3 == "0") {print}' /etc/passwd
```

## SSH Servis

**24.** Sada možemo za **SSH** servis onemogućiti direktno logiranje kao korisnik *root*. Otvorite datoteku: `/etc/ssh/sshd_config` te pronađite sljedeći redak u kojem se omogućava logiranje kao *root* korisnik na SSH servis:

```
PermitRootLogin yes
```

Promijenite ga u:

```
PermitRootLogin no
```

Eventualno razmislite o promijeni standardnog SSH TCP porta 22 u neki drugi (primjerice 22222).
Dakle ako imate konfigurirano sljedeće:

```
Port 22
```

ili

```
#Port 22
```

To promijenite u neki drugi TCP port (ako vam je *SELlinux* uključen, morate napraviti modifikacije i u njemu):

```
Port 22222
```

**25.** Bilo bi dobro za ***SSH*** servis onemogućiti korisničke račune koji imaju praznu lozinku (odnosno nemaju ju).
Dakle otvorimo standardnu konfiguraciju datoteku *SSH* servisa (`/etc/ssh/sshd_config`) te postavimo sljedeće:

```
PermitEmptyPasswords no
```

**26.** Nadalje, možemo za ***SSH*** servis ograničavati ili odobravati korisničke račune ili korisničke grupe kojima ili odobravamo ili branimo upotrebu *SSH* servisa. Prvo pogledajmo kako eksplicitno navoditi korisničke račune kojima odobravamo spajanje na *SSH* servis; primjerice eksplicitno ćemo odobriti pristup za korisnički račun *hrvoje (u datoteci* `/etc/ssh/sshd_config`*)*:

```
AllowUsers hrvoje
```

Isto tako možemo i zabraniti pristup za određenog korisnika (za primjer napravimo to za istog korisnika):

```
DenyUsers hrvoje
```

Nadalje, možemo odobriti pristup za korisničku grupu; primjerice za grupu *devel*, koju smo prethodno kreirali i stavili nekoliko korisnika u nju:

```
AllowGroups devel
```

Odnosno možemo i zabraniti pristup određenoj korisničkoj grupi; primjerice ovako:

```
DenyGroups devel
```

**I za grupe i korisnike, nakon navedene ključne riječi (**`AllowUsers`**,** `DenyUsers`**,** `AllowGroups`**,** `DenyGroups`**) možemo nîzati imena korisnika ili grupa, odvojenih razmakom (*space*).**

**27.** Dodatno eventualno možemo za **SSH** servis onemogućiti upotrebu/upisivanje lozinke, već sâmo upotrebu kriptografskih ključeva. U već navedenoj konfiguracijskoj datoteci (`/etc/ssh/sshd_config`) pronađite redak:

```
PasswordAuthentication yes
```

Promijenite ga u:

```
PasswordAuthentication no
```

Te snimite sve promjene. Sada kreirajmo nove ključeve, za korisnika `hrvoje` (ako smo prethodno **root**), na sljedeći način:

```
su - hrvoje
ssh-keygen -t rsa
```

Dobit ćemo nekoliko pitanja koje samo potvrdite stiskanjem tipke *ENTER*:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hrvoje/.ssh/id_rsa):
Created directory '/home/hrvoje/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hrvoje/.ssh/id_rsa.
Your public key has been saved in /home/hrvoje/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:I0NVZbM/zpT03Mwpqh+JH/uR4leWFxiWRnNZ9rVKY3Q hrvoje@localhost.localdomain
The key's randomart image is:
+---[RSA 3072]----+
|         ....=+.E*|
+----[SHA256]-----+
```

Sada će nam unutar direktorija `/home/hrvoje/.ssh/` biti kreirani privatni i javni ključevi za ovog korisnika (`hrvoje`).
Dakle ovdje ćemo dobiti javni ključ: `id_rsa.pub` i privatni ključ: `id_rsa`. Javni ključ si kopirajte (sadržaj datoteke) i ubacite ga u svoj SSH klijent, a na svom SSH klijentu/računalu si kreirajte isto ovakav par ključeva od kojih svoj javni ključ ubacite ovdje na poslužitelj, u datoteku `/home/hrvoje/.ssh/authorized_keys` (koju trebate kreirati).

Ograničite i broj krivih pokušaja spajanja na SSH servis (na 5) uz dodatno vrijeme čekanja od 2.minute.
Stoga dodajmo i sljedeće retke u datoteku `/etc/ssh/sshd_config`:

```
LoginGraceTime 2m
MaxAuthTries 5
```

Potom restartajte SSH servis (za *RedHat/CentOS 7.x+*):

```
systemctl restart sshd
```

Zatim se probajte logirati pomoću SSH ključeva.

**28. Ograničite broj istodobnih SSH veza koristeći `MaxStartups` parametar u `/etc/ssh/sshd_config`.**

## Razni servisi

**29. Koristite centralizirane servise za rad s korisničkim računima i grupama**

Koristite centralizirane servise za rad s korisničkim računima i grupama poput LDAP ili nekog od **AAA** servisa. Poglavito ako Linux koristite na mrežnim uređajima poput usmjerivača, vatrozida i sličnih uređaja. Naime tako se sve radnje vezane za kreiranje i rad s korisnicima i korisničkim grupama, ali i druge aktivnosti vezane za njih mogu odrađivati na udaljenom i sigurnom (centraliziranom) mjestu, na kojem se mogu i pratiti.

**30. Ako je to moguće ili potrebno, koristite dodatne servise za praćenje rada sustava**

U slučajevima kada osim oslanjanja na *SELinux* sustav, želite biti u stanju i pratiti tko je pristupao ili mijenjao određene datoteke na sustavu, to možete postići upotrebom servisa poput `auditd` koji vam daje detaljne mogućnosti:

- Praćenja datuma, vremena i vrste događaja koji se *dogodio* uz označavanja istih u više razina.
- Povezivanja događaja s korisnikom koji ga je izazvao.
- Pohranu svih načina spajanja na sustav, uz korisnike koji se spajaju.
- Pristup, promijene ili čitanje određenih datoteka i komponenti sustava, kao i druge detalje.

**31. Ograničenja upotrebe *Cron* servisa**

Prema potrebi možete ograničiti upotrebu *Cron* servisa tako da sâmo određenim korisnicima onemogućite njegovu upotrebu, što se postiže navođenjem korisnika u datoteku: `/etc/cron.deny`. Pa bi tako zabrana upotrebe *Cron* servisa za korisnike `pero` i `hrvoje` bila dodavanjem sljedećih redaka u navedenu konfiguracijsku datoteku:

```
pero
hrvoje
```

Moguće je napraviti i obratno, samo definirati korisnike kojima je dozvoljena upotreba *Cron* servisa, dodavanjem korisnika u drugu posebnu datoteku, koju je potrebno kreirati: `/etc/cron.allow`.
Pri tome datoteka `/etc/cron.deny` može biti i prazna.

## Mreža i mrežni servisi

**32. Isključite IPv6 protokol, ako ga ne koristite.**

Isključivanje IPv6 protokola možete napraviti na više načina, a mi ćemo spomenuti sâmo jedan.
Prvi način je dodavanjem sljedeća dva retka u datoteku: `/etc/sysctl.conf`.

```
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
```

Potom je potrebno aktivirati ove promjene, što ne zahtjeva restart cijelog poslužitelja/računala, osim eventualno *servisa*.

```
sysctl -p
```

**33. Optimirajte mrežne i druge parametre sustava.**

Optimizacijom raznih sigurnosnih mehanizama sustava podižete sveukupnu razinu zaštite. Naime vrlo jednostavno možete uključiti (za početak) nekoliko korisnih zaštitnih mehanizama dodavanjem sljedećih redaka u datoteku: `/etc/sysctl.conf` (Komentari koje smo dodavali, počinju sa znakom `#`).

```
# Uključimo ASLR zaštitu
kernel.randomize_va_space = 2

# Uključimo zaštitu: IP spoofing
net.ipv4.conf.all.rp_filter = 1

# Isključimo IP source routing
net.ipv4.conf.all.accept_source_route = 0

# Ignorirajmo ICMP (IPv4) broadcast zahtjeve
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_messages = 1

# Omogućimo logiranje mrežnih paketa iz opsega rezerviranih IP adresa
net.ipv4.conf.all.log_martians = 1

#  Postavite SYN cookies kako biste spriječili SYN flood napade:
net.ipv4.tcp_syncookies = 1
```

Dodatno proučite i druge opcije i parametre koje ovdje možete koristiti: **https://www.kernel.org/doc/Documentation/sysctl/**.

**34. Koristite *Bond/Agregaciju/Team* tehnologiju gdje god je moguće.**
Koristite *Bond/Agregaciju/Team* tehnologiju gdje god je to moguće. Na sustavima na kojim imate više od jedne fizičke mrežne kartice. S time se osiguravate da ispadom jedne fizičke mrežne kartice nećete potpuno izgubiti pristup mreži.



Pogledajte poglavlje:
**20.6.6. Redundancija i bonding (*Agregacija/Etherchannel/Teaming*).**

**35. Osigurajte SNMP poslužitelj.**
***SNMP*** (*Simple Network Management Protocol*) protokol se koristi za slanje raznih alarma odnosno poruka s mrežnih komponenti (preklopnici, usmjerivači, vatrozidi), ali i računala kao i poslužitelja, u svrhu obavješćivanja o promjenama u radu. U konačnici ovaj protokol se koristi za praćenje i analizu rada mreže i raznih mrežnih uređaja i elemenata na mreži.
Postoje inačice *SNMP1*, *SNMP2* i *SNMP3,* pri čemu tek zadnja inačica nudi kriptografske mogućnosti, pa ju se preporučuje koristiti gdje god je to moguće. U suprotnom koristite starije inačice, ali ih dobro zaštitite. Primjerice za takozvani „*SNMP community string*" je postavljena standardna lozinka „***public***" koju obavezno treba promijeniti u nešto kompleksnije.

Ako koristite `snmpd` servis, tada u konfiguracijskoj datoteci: `/etc/snmp/snmpd.conf`, redak koji izgleda ovako:
```
com2sec notConfigUser  default       public
```

Promijenite ga tako da stavite složeniju lozinku, poput recimo:
```
com2sec notConfigUser  default       L!3u-4C5/16*i7c+ABvEt51
```

Ne zaboravite proučiti i druge opcije i parametre koje treba optimizirati te restartajte ovaj servis.

**36. Osigurajte druge mrežne servise.**
Dobro proučite sigurnosne preporuke za sve mrežne servise koje koristite i dobro ih osigurajte. To su primjerice:

- Imenični servisi (***DNS***, ***LDAP***, i slični).
- AAA Servisi (***RADIUS***, ***DIAMETER***, ***TACACS+***).
- Mrežni servisi poput: ***DHCP***/***BOOTP***, ***TFTP*** i drugih.
- Drugi aplikacijski servisi: ***HTTP***/***HTTPS*** (Web poslužitelji), ***FTP*** i drugi.
- Servisi elektroničke pošte: ***IMAP***/***POP3***, ***SMTP*** i drugi.
- Sustavi za dijeljenje datoteka/podataka na mreži: ***SMB***/***CIFS*** (Samba), ***NFS*** (pogl.0), ***iSCSI*** i drugi.

Pročitajte i sljedeće smjernice*:

- Za ***RedHat/CentOS 7.x***: https://nvd.nist.gov/ncp/checklist/811 (905)
- Za ***RedHat/CentOS 8.x***: https://nvd.nist.gov/ncp/checklist/909 (906)

**37. Uključite vatrozid (engl. *Firewall*).**
Po mogućnosti uključite vatrozid (Engl. *Firewall*) te dozvolite sâmo one mrežne servise prema internetu i drugim lokalnim mrežama koje stvarno i nužno trebate. Prvenstveno pripazite i na to da maksimalno ograničite pristup servisima s interneta prema svojoj lokalnoj mreži. Najsigurnije pravilo je prvo sve zabraniti, a onda malo po malo dozvoljavati.
Vatrozid kombinirajte sa ***SELinux*** sustavom.



Pogledajte poglavlje:
**26.7. Vatrozid (*Firewall*).**

**38. Zabranite prijave s IP adresa koje nisu iz vaše zemlje koristeći GeoIP blokiranje u vatrozidu.**

**39. Onemogućite UPnP na mrežnim uređajima kako biste spriječili nenamjerni vanjski pristup.**

**40. Koristite `nmap` i druge alate za redovito skeniranje otvorenih portova i identificirajte potencijalne ranjivosti.**

**41. Postavite *TCP Keepalive* vrijednost kako biste automatski prekidali neaktivne veze. Primjerice:**
```
sysctl net.ipv4.tcp_keepalive_time=600
```

**42. Konfigurirajte DMZ zonu za dodatnu mrežnu sigurnost ako koristite pristup javnim servisima.**

**43. Koristite tcpwrappers servise za filtriranje mrežnog prometa i omogućavanje pristupa samo određenim IP adresama.**

**44. Onemogućite nepotrebne mrežne protokole poput DCCP i SCTP.**

**45. Implementirajte Snort ili Suricata kao IDS/IPS sustave.**

**46. Uskladite satove na svim poslužiteljima i mrežnoj opremi.**
S obzirom na činjenicu kako unutarnji satovi (*RTC*/*BIOS*) u računalima i svoj opremi imaju poprilična odstupanja, potrebno je uskladiti točno vrijeme na svoj opremi: uređajima na mreži i poslužiteljima. Stoga koristite ***NTP*** protokol za sinkronizaciju.



Za konfiguraciju ***NTP*** protokola pogledajte poglavlje:
**26.6. NTP.**

**47. Pratite aktivnosti korisnika i pregledavajte log datoteke.**
Pratite aktivnosti korisnika ovisno o aplikacijama te pregledavajte log datoteke aplikacija, ali i log datoteke sustava.
Vezano za log datoteke sustava koje se tiču sigurnosti, redovito pregledavajte:

- `/var/log/auth.log` - ovdje se zapisuju svi pokušaji logiranja na sustav.
- `/var/log/boot.log` - ovdje se zapisuju poruke pri pokretanju računala (*boot* proces).
- `/var/log/cron.log` - ovdje se zapisuju podaci o pokretanju ***Cron*** poslova.
- `/var/log/kern.log` - ovdje se zapisuju log poruke kernela.
- `/var/log/maillog` - ovdje se zapisuju podaci od poslužitelja elektroničke pošte.
- `/var/log/message` - ovdje se zapisuju sve sistemske poruke.
- `/var/log/mysqld.log` - ovdje se zapisuju sve sistemske poruke *MySQL* baze podataka (ako ju imate).
- `/var/log/secure` - ovdje se zapisuju sve sistemske sigurnosne poruke (od svih aplikacija).
- `/var/log/yum.log` ili `/var/log/dnf.*` - ovdje se zapisuju poruke ***YUM*** ili ***DNF*** menadžera za softverske pakete (primjerice o instalaciji, deinstalaciji softverskih paketa i slično).



Pogledajte poglavlje:
**17. Log datoteke.**

Razne log poruke možete pregledavati sa sljedećim naredbama (za *RedHat/CentOS 7.x+*):

```
journalctl
journalctl -u network.service
journalctl -u ssh.service
journalctl -k
```

Ako je moguće koristite udaljene log poslužitelje (**Rsyslog**) na koje će se dodatno zapisivati sve poruke sustava (i aplikacija).

**48. Fine optimizacije sustava.**
Razmislite o finoj optimizaciji cijelog sustava. Mi ćemo spomenuti sâmo neke od njih.

- Razmislite o ograničenju broja opisnika datoteka (*File deskriptora*).



Pogledajte **poglavlje: 4.5.6.1. Ograničavanje File deskriptora.**

- Razmislite o povećanju broja dostupnih mrežnih portova.



Pogledajte poglavlje: **19.8.1. TCP/UDP Portovi.**

- Razmislite o praćenju mrežnih veza (moduli `ip_conntrack` i `xt_conntrack`).



Pogledajte poglavlje: **24.2.16.1. Timeri i stanja veze.**

- Optimizirajte ***TCP*** parametre veze.



Pogledajte cijelo poglavlje: **24. Transportni protokoli (OSI sloj 4)**.

- Pročitajte i druge preporuke za *sysctl* varijable, na izvoru informacija: **(909)**.



Pogledajte poglavlje: **4.5.8. Naredba *sysctl*.**

**49. Izrada sigurnosnih kopija podataka.**
Redovito radite sigurnosne kopije podataka (Engl. *Backup*), odnosno kako podataka, tako i po mogućnosti cijelog sustava, pogotovo, ako koristite virtualno računalo ili Linux kontejner.



Za više detalja pročitajte detaljne smjernice **NIST** instituta za standardne i tehnologiju*:
- Za ***Red Hat 7.x***: **https://nvd.nist.gov/ncp/checklist/811**(905)
- Za ***Red Hat 8.x***: **https://nvd.nist.gov/ncp/checklist/909**(906)
- **Za *Red Hat 9.x.*: https://ncp.nist.gov/checklist/1072**

Pratite i druge smjernice, kao i otkrivene ranjivosti sustava i preporuke kako se od njih zaštititi, koje se stalno osvježavaju i katalogiziraju na servisima poput ***CVE***. Naime ***CVE*** (*Common Vulnerabilities and Exposures*), daje popis svih javno otkrivenih nedostataka (propusta ili ranjivosti) u računalnoj sigurnosti. Kad se netko referencira na ***CVE***, to obično znači da vam ukazuje na točno određeni ***CVE*** broj (*ID*) koji je dodijeljen sigurnosnom propustu.

Adresa *CVE* centra je: https://cve.mitre.org/data/downloads/index.html te s nje možete kopirati listu svih propusta: primjerice prema godini ili prema točno određenom *ID* broju.

Ako recimo želimo pronaći listu svih propusta za Linux (prema godinama) to možemo pronaći odlaskom na sljedeću poveznicu: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=linux.

*Važno je znati kako CVE oznake imaju sljedeći format:* ***CVE****-GODINA-RedniBroj.*
*Primjerice: CVE-2019-3882 znači kako je ovaj propust otkriven 2019 godine i nosi redni broj 3882unutar 2019 godine.*

Pogledajmo konkretan izvještaj (za navedeni ***CVE*** broj odnosno oznaku):
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-3882.
Na navedenoj stranici vidimo poveznice na druge dokumente (i distribucije Linuxa), što je čest slučaj, ako se radi o propustu koji se tiče kernela ili programa koji se nalaze u većini distribucija Linuxa. Tako za ovaj konkretan slučaj možemo vidjeti kako je ***RedHat*** objavio svojih nekoliko povezanih slučajeva, poput:
https://access.redhat.com/errata/RHSA-2019:2029, https://access.redhat.com/errata/RHSA-2019:2043 i drugih, a u kojima se vidi objašnjenje problema te kako ga riješiti. Tako je primjerice za: ***RHSA-2019:2043*** vidljivo sljedeće:
- „*An update for kernel-rt is now available for Red Hat Enterprise Linux 7. Red Hat Product Security has rated this update as having a security impact of Important. A Common Vulnerability Scoring System (CVSS) base score, which gives a detailed severity rating, is available for each vulnerability from the CVE link(s) in the References section.*“
- Slijedi cijeli nîz detalja, te kako je problem riješen u novoj inačici kernela (**Updated Packages**), za x86_64 sustave je to RPM paket: „`kernel-rt-3.10.0-1062.rt56.1022.el7.x86_64.rpm`“, kao i drugi navedeni paketi.

***Red Hat*** kao i druge distribucije Linuxa na svojim stranicama imaju ili ***CVE*** oznake ili svoje oznake za isti propust, poput ovih ***Red Hat-ovih (RHSA-2019:2029, RHSA-2019:2043, …).***

**Izvori informacija: (878),(879),(880),(903),(904)(905),(906),(909).**

# 29. Dodaci

## 29.1. ASCII tablica

**ASCII** (engl. *American Standard Code for Information Interchange*) je 7-bitni kôd za karaktere (alfabete) u kojem svaki bît predstavlja pojedini alfabet. ASCII je poznati i kao **ANSI X3.4**. Pogledajmo ASCII tablicu razlomljenu u dvije logičke cjeline. Prva cjelina su kôdovi od 0 do 31 koji predstavljaju kontrolne kôdove odnosno kontrolne znakove:

| DEC | OCT | HEX | Simbol | Opis | DEC | OCT | HEX | Simbol | Opis |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 000 | 00 | NUL | Prazan znak (karakter) | 20 | 024 | 14 | DC4 | Device Control 4 |
| 1 | 001 | 01 | SOH | Početak zaglavlja | 21 | 025 | 15 | NAK | Negativna potvrda |
| 2 | 002 | 02 | STX | Početak teksta | 22 | 026 | 16 | SYN | Synchronous Idle |
| 3 | 003 | 03 | ETX | Kraj teksta | 23 | 027 | 17 | ETB | Kraj bloka transmisije |
| 4 | 004 | 04 | EOT | Kraj prijenosa | 24 | 030 | 18 | CAN | Prekid (*Cancel*) |
| 5 | 005 | 05 | ENQ | Enquiry | 25 | 031 | 19 | EM | Kraj medija |
| 6 | 006 | 06 | ACK | Potvrda | 26 | 032 | 1A | SUB | Zamjena (*Substitute*) |
| 7 | 007 | 07 | BEL | Zvono (Bell) | 27 | 033 | 1B | ESC | Izlaz (*Escape*) |
| 8 | 010 | 08 | BS | Back Space | 28 | 034 | 1C | FS | Separator datoteke |
| 9 | 011 | 09 | HT | Horizontalni *Tab* | 29 | 035 | 1D | GS | Separator grupe |
| 10 | 012 | 0A | LF | Line Feed | 30 | 036 | 1E | RS | Separator zapisa |
| 11 | 013 | 0B | VT | Vertikalni *Tab* | 31 | 037 | 1F | US | Separator jedinice (*Unit*) |
| 12 | 014 | 0C | FF | Form Feed | | | | | |
| 13 | 015 | 0D | CR | Carriage Return (*ENTER*) | | | | | |
| 14 | 016 | 0E | SO | Shift Out / X-On | | | | | |
| 15 | 017 | 0F | SI | Shift In / X-Off | | | | | |
| 16 | 020 | 10 | DLE | Data Line Escape | | | | | |
| 17 | 021 | 11 | DC1 | Device Control 1 (obično **XON**) | | | | | |
| 18 | 022 | 12 | DC2 | Device Control 2 | | | | | |
| 19 | 023 | 13 | DC3 | Device Control 3 (obično **XOFF**) | | | | | |

Drugi dio tablice su kôdovi od 32 do 127 u kojemu su sva slova abecede, brojevi, posebni znakovi i razni simboli (na sljedećoj stranici). Za 8-bitni ASCII, nakon pozicije 127, kreću posebni znakovi za iscrtavanje rubova, kao i drugi znakovi te slova za druge jezike (ovisno o kodnoj stranici).

| DEC | OCT | HEX | Simbol | Opis *(Description)* | DEC | OCT | HEX | Simbol | Opis *(Description)* |
|---|---|---|---|---|---|---|---|---|---|
| 32 | 040 | 20 | | Razmak (*Space*) | 56 | 070 | 38 | **8** | Osam |
| 33 | 041 | 21 | **!** | Uskličnik | 57 | 071 | 39 | **9** | Devet |
| 34 | 042 | 22 | **”** | Dvostruki navodnik | 58 | 072 | 3A | **:** | Dvotočka |
| 35 | 043 | 23 | # | Ljestvice | 59 | 073 | 3B | **;** | Točka zarez (točka sa zarezom) |
| 36 | 044 | 24 | **$** | Dolar | 60 | 074 | 3C | < | Manje od |
| 37 | 045 | 25 | **%** | Postotak | 61 | 075 | 3D | = | Jednako |
| 38 | 046 | 26 | **&** | Znak *I* (*AND*) | 62 | 076 | 3E | > | Veće od |
| 39 | 047 | 27 | ' | Jednostruki navodnik/apostrof | 63 | 077 | 3F | **?** | Znak upitnik |
| 40 | 050 | 28 | **(** | Otvorena obla zagrada | 64 | 100 | 40 | **@** | Vitičnik/*Et* simbol *(At symbol/Monkey)* |
| 41 | 051 | 29 | **)** | Zatvorena obla zagrada | 65 | 101 | 41 | **A** | Veliko slovo A |
| 42 | 052 | 2A | ***** | Zvjezdica *(Asterisk)* | 66 | 102 | 42 | **B** | Veliko slovo B |
| 43 | 053 | 2B | + | Plus | 67 | 103 | 43 | **C** | Veliko slovo C |
| 44 | 054 | 2C | **,** | Zarez (*Comma*) | 68 | 104 | 44 | **D** | Veliko slovo D |
| 45 | 055 | 2D | - | Minus *(Hyphen)* | 69 | 105 | 45 | **E** | Veliko slovo E |
| 46 | 056 | 2E | **.** | Točka | 70 | 106 | 46 | **F** | Veliko slovo F |
| 47 | 057 | 2F | */* | Kosa crta ili krôz | 71 | 107 | 47 | **G** | Veliko slovo G |
| 48 | 060 | 30 | **0** | Nula | 72 | 110 | 48 | **H** | Veliko slovo H |
| 49 | 061 | 31 | **1** | Jedan | 73 | 111 | 49 | **I** | Veliko slovo I |
| 50 | 062 | 32 | **2** | Dva | 74 | 112 | 4A | **J** | Veliko slovo J |
| 51 | 063 | 33 | **3** | Tri | 75 | 113 | 4B | **K** | Veliko slovo K |
| 52 | 064 | 34 | **4** | Četiri | 76 | 114 | 4C | **L** | Veliko slovo L |
| 53 | 065 | 35 | **5** | Pet | 77 | 115 | 4D | **M** | Veliko slovo M |
| 54 | 066 | 36 | **6** | Šest | 78 | 116 | 4E | **N** | Veliko slovo N |
| 55 | 067 | 37 | **7** | Sedam | 79 | 117 | 4F | **O** | Veliko slovo O |
| 80 | 120 | 50 | P | Veliko slovo P | 104 | 150 | 68 | h | Malo slovo h |
| 81 | 121 | 51 | Q | Veliko slovo Q | 105 | 151 | 69 | i | Malo slovo i |
| 82 | 122 | 52 | R | Veliko slovo R | 106 | 152 | 6A | j | Malo slovo j |
| 83 | 123 | 53 | S | Veliko slovo S | 107 | 153 | 6B | k | Malo slovo k |
| 84 | 124 | 54 | T | Veliko slovo T | 108 | 154 | 6C | l | Malo slovo l |
| 85 | 125 | 55 | U | Veliko slovo U | 109 | 155 | 6D | m | Malo slovo m |
| 86 | 126 | 56 | V | Veliko slovo V | 110 | 156 | 6E | n | Malo slovo n |
| 87 | 127 | 57 | W | Veliko slovo W | 111 | 157 | 6F | o | Malo slovo o |
| 88 | 130 | 58 | X | Veliko slovo X | 112 | 160 | 70 | p | Malo slovo p |
| 89 | 131 | 59 | Y | Veliko slovo Y | 113 | 161 | 71 | q | Malo slovo q |
| 90 | 132 | 5A | Z | Veliko slovo Z | 114 | 162 | 72 | r | Malo slovo r |
| 91 | 133 | 5B | [ | Otvorena uglata zagrada | 115 | 163 | 73 | s | Malo slovo s |
| 92 | 134 | 5C | \ | Obrnuta kosa crta | 116 | 164 | 74 | t | Malo slovo t |
| 93 | 135 | 5D | ] | Zatvorena uglata zagrada | 117 | 165 | 75 | u | Malo slovo u |
| 94 | 136 | 5E | ^ | (*Caret – circumflex*) | 118 | 166 | 76 | v | Malo slovo v |
| 95 | 137 | 5F | _ | Podcrta/Donja crta | 119 | 167 | 77 | w | Malo slovo w |
| 96 | 140 | 60 | ` | Lijevi jednostruki navodnik | 120 | 170 | 78 | x | Malo slovo x |
| 97 | 141 | 61 | a | Malo slovo a | 121 | 171 | 79 | y | Malo slovo y |
| 98 | 142 | 62 | b | Malo slovo b | 122 | 172 | 7A | z | Malo slovo z |
| 99 | 143 | 63 | c | Malo slovo c | 123 | 173 | 7B | { | Otvorena vitičasta zagrada |
| 100 | 144 | 64 | d | Malo slovo d | 124 | 174 | 7C | \| | Vertikalna/Okomita crta (*pipe*) |
| 101 | 145 | 65 | e | Malo slovo e | 125 | 175 | 7D | } | Zatvorena vitičasta zagrada |
| 102 | 146 | 66 | f | Malo slovo f | 126 | 176 | 7E | ~ | Tilda *(Equivalency sign – tilde)* |
| 103 | 147 | 67 | g | Malo slovo g | 127 | 177 | 7F | | Brisanje *(Delete)* |

# 29.2. Popis i kratki opis Linux naredbi

U ovom poglavlju nabrojat ćemo razne Linux naredbe koje smo koristili, kao i one koje bi vam mogle dobro doći, uz njihov kratki opis.

`abipkgdiff` - program za usporedbu ***ABI*** (***KMI***) varijabli i funkcija između dvije inačice kernela.
`agetty` - program za rad s fizičkim ili virtualnim terminalima.
`alias` - skraćeno ime (alias) za neku naredbu ili nîz naredbi.
`anacron` - naredba za izvršavanje željenih naredbi (programa ili skripti) periodički.
`apropos` - naredba koja prikazuje gdje se sve u uputama za sve naredbe nalazi traženi pojam ili riječ.
`apt` - *Advanced Package Tool*, upravitelj paketa više razine za Debian distribucije Linuxa.
`at` - pokreće neku naredbu ili skriptu u željeno vrijeme.
`arp` - naredba za rad s *ARP* tablicom (za mrežu).
`atq` - izlistava sve *at* (`at`) zadatke/poslove.
`atrm` - za brisanje određenog *at* (`at`) zadatka/posla.
`authconfig` - naredba za rad odnosno prikaz resursa sustava.
`authselect` - novija naredba (od `authconfig`) za rad/prikaz resursa sustava. Na nekim sustavima se ne koristi.
`arping` - naredba za slanje ARP poruka na mrežu.
`atop` - prikazuje sve pokrenute programe/procese u realnom vremenu, te osvježava prikaz uz napredne mogućnosti. Pogledajte i naredbe: `top` i `htop`.
`auditd` - servis za praćenje aktivnosti na sustavu.
`awk` - programski jezik, ali i naredba za filtriranje te baratanje s uzorcima teksta (pr. prema stupcima, poljima i slično).
`base64` - naredba za (en)kodiranje i dekodiranje pomoću *base64* algoritma.
`bash` - Unix i Linux naredbeni redak odnosno ljuska za rad (*shell*).
`bg` - ispisuje ili ubacuje programe u pozadinsko izvršavanje ili pokreće one koji su privremeno zaustavljeni.
`blkparse` - program za obradu prikupljenih statistika rada diskovnog (pod)sustava.
`blkid` - program koji nam prikazuje particije diskova i njihove jedinstvene identifikacijske oznake (*UUID*)
`buildah` – program za rad s diskovnim slikama *Podman (OCI)* kontejnera.
`break` - naredba za terminiranje izvršavanja unutar petlje (`for`, `while` ili `until`).
`brctl` - naredba za rad s mrežnim mostom (*bridgem*).
`btt` - naredba za analizu ispisa statistika rada diskovnog podsustava, koji je generirala naredba `blkparse`.
`bzip2` - program za komprimiranje i dekomprimiranje podataka (datoteka) korištenjem *BZIP2* algoritma.
`bunzip2` - program za dekomprimiranje podataka (datoteka) korištenjem *BZIP2* algoritma.
`bzcat` - program za ispis (na ekran) tekstualnih datoteka komprimiranih pomoću *BZIP2* algoritma.
`cal` - ispisuje odnosno prikazuje kalendar s datumima za trenutni mjesec ili godinu.
`case` - naredba s kojom se kondicionalno izvršava cijeli blok naredbi.
`cat` - ispisuje sadržaj (tekstualne) datoteke na standardni izlaz (ekran/*shell*) (Engl. *Concatenate*).
`capsh` - naredba za rad s takozvanim mogućnostima (engl. *capabilities*) procesa i/ili datoteka.
`cd` - naredba za promjenu trenutnog radnog direktorija (mape) u kojem se trenutno nalazimo (Engl. C*hange directory*).
`cfdisk` - tekstualno grafički program za particioniranje diskova, za *MBR* diskove.
`cgdisk` - tekstualno grafički program za particioniranje diskova, za *GPT* i *MBR* diskove.
`chage` - naredba za rad s korisničkim računima (za promjenu vremena valjanosti/aktivnosti korisničkog računa).
`chgrp` - naredba za promjenu pripadnost grupi, za određenu datoteku ili direktorij (mapu) (Engl. *Change group ownership*).
`chattr` - za promijene naprednih atributa datoteka ili direktorija (Engl. *Change file attributes*).
`chmod` - za promjene ovlasti datoteka ili direktorija s novim ovlastima (*Read/Write/eXecute*: za korisnika, grupu ili ostale).
`chkconfig` - program za konfiguriranje servisa koji se trebaju pokrenuti tijekom pokretanja računala, za *CentOS/RedHat 6.x*.
`chown` - za promjenu vlasnika (ili grupe) nad datotekama ili direktorijima (mapama) (Engl. *Change file owner*).
`chronyc` – sučelje za konfiguraciju i rad s ***chronyd*** servisom i NTP poslužiteljima.
`chronyd` – NTP poslužitelj i NTP klijent.
`chroot` – program za kreiranje izoliranog korjenskog direktorija.
`chrt` - promjeni „*real-time*" atribute izvršavanja procesa (programa) (Engl. *Change real time attributes*).
`chpasswd` - naredba za promjenu lozinki za cijeli nîz korisnika (skriptno).
`cksum` - program za izračun *CRC* vrijednosti odnosno provjernog zbroja željene datoteke (Engl. *Checksum*).
`cmp` - program za usporedbu dvije datoteke, binarnom usporedbom (*byte-by-byte*) (Engl. *Compare*).
`column` - program za prepoznavanje teksta i prikazivanje istog u stupcima (Engl. *Columnate lists*).
`comm` - program za usporedbu sadržaja tekstualnih datoteka (Engl. *Compare*).
`compgen` - program za ispis svih naredbi i aliasa dostupnih unutar trenutne sesije naredbenog retka (unutar *shella*).
`continue` - ugrađena naredba (*bash*) kojom unutar petlje (`for`, `while` ili `until`) preskačemo trenutnu iteraciju.
`conntrack` – naredba za provjeru stanja i rad sa savim aktivnim mrežnim konekcijama na sustavu.
`cp` - za kopiranje datoteka ili direktorija (Engl. *Copy*).
`cpio` - kopiranje ili ekstrahiranje podataka u ili iz *CPIO* arhive.
`crontab` - program za upisivanje zadataka koji se trebaju pokrenuti o željeno vrijeme (pr. svaki dan u 20:00.h.) u *cron* servis.
`cut` - služi za izbacivanje ili izrezivanje dijelova teksta/stupca iz datoteka (Engl. *Cut*).
`curl` - služi za dohvaćanje ili postavljanje datoteka na udaljeni poslužitelj (HTTP/HTTPS/FTP/LDAP/POP3/SFTP/SCP,…).
`date` - ispisuje trenutno vrijeme (sat i datum) ili ga može postaviti za cijeli sustav odnosno računalo (Engl. *Date*).

`dd` - za konvertiranje ili kopiranje podataka, prema principu: *ulazni podaci→ izlazni uređaj/datoteka/direktorij* (Engl. *Disk dump*).
`debugfs` - program za debugiranje Linux „***ext***“ datotečnih sustava (*ext2/ext3/ext4*).
`depmod` - program koji radi s datotekom (*modules.dep*) koja opisuje vezu između kernel simbola i kernel modula.
`df` - program koji ispisuje zauzeće particija diskova (Engl. *Disk free*).
`dhclient` - program za ručnu konfiguraciju DHCP klijenta na željenoj mrežnoj kartici (sučelju) (Engl. *DHCP client*).
`diff` - program za uspoređivanje sadržaja dvije tekstualne datoteke, redak po redak.
`dig` - program za slanje raznih DNS upita domenu koju istražujemo/testiramo (pr. *opensource-osijek.org*).
`dkms` – programa za automatiziranu izradu kernel modula pomoću *DKMS* komponente.
`dmidecode` - program za izlistanje hardverskih informacija o komponentama računala (MB/BIOS/RAM/CPU/…).
`dmsetup` - program za podešavanje i konfiguraciju logičkih disk uređaja koji koriste tzv. *Device mapper*.
`dmesg` - program koji nam ispisuje sve poruke linux kernela u trenutku pokretanja sustava.
`dnf` - menadžer softverskih paketa za rad s RPM paketima. On je nasljednik programa `yum` na *RedHatCentOS* 8.x.
`dnsmasq` – servis za DHCP, BOOTP, TFTP i DNS protokole.
`dpkg` - *Debian Package*, upravitelj paketa niže razine za Debian distribucije Linuxa.
`dracut` - program za rad s *initramfs image-em* (boot ram disk *image*). Pogledajte i `lsinitrd`.
`dstat` - program za praćenje rada sustava, koji kombinira sve što i naredbe poput: `vmstat`, `iostat` i neke druge.
`du` - ispisuje koliko zauzima određeni direktorij s datotekama (Engl. *Disk usage*).
`dumpe2fs` - program za prikaz statistika rada ***ext*** datotečnih sustava (***ext2/ext3/ext4***).
`dumpkeys` - program za rad s translacijskom tablicom tipkovnice.
`e2fsck` - program za provjeru integriteta datotečnog sustava (Engl. *Extended 2 file system check*).
`echo` - ispiše željeni tekst, vrijednost varijable ili drugo, u naredbeni redak (*shell*/ljusku).
`edac-util` - prikazuje ***EDAC*** (*Error Detection and Correction*) status RAM memorije (ako je ECC memorija podržana).
`edquota` - program za rad s disk kvotama.
`egrep` - varijanta program `grep`, koja podržava proširene regularne izraze. *Služi za prikaz redova teksta koji sadrže ključnu riječ.*
`else` - funkcija unutar uvjeta (`if` - `else`) koja se poziva, ako uvjet nije zadovoljen.
`env` - ispisuje sistemske (*environment*) varijable.
`ethtool` - program za konfiguraciju i rad s mrežnim sučeljima na niskoj razini mreže Linuxa i/ili mrežnih sučelja (kartica).
`exec` - izvršava novu naredbu, koja mijenja trenutni proces naredbenog retka (*shella*).
`exit` - ovom naredbom se izlazi iz *shell* skripte ili se odlogira sa sustava.
`expand` - pretvara TAB razmake u razmake (*space*).
`export` - izvozi (eksportira) željenu varijablu u sve pod ljuske (*subshells)* koje ćete pokrenuti vi ili vaša skripta.
`expect` - za automatizaciju izvršavanja raznih operacija: pošalji naredbu, a ovisno o njenom odgovoru pokreće neku drugu.
`fallocate` - naredba za iznimno brzo kreiranje velikih (*sparse*) datoteka.
`fdisk` - program za rad s particijama (samo sa ***MBR*** vrstom particija).
`find` - koristi se za traženje datoteka prema željenom kriteriju (ime, datum, veličina, ovlasti, …).
`findmnt` - program koji nam prikazuje sve montirane sustave (i particije) uz detaljan prikaz opcija s kojim su montirani.
`fg` - prebacuje trenutno pokrenuti program, koji se izvršava u pozadini u izvršavanje u vidljivoj ljusci (*shellu).*
`fgrep` - varijanta programa `grep`, koja ne podržava regularne izraze i optimizirana je za vrlo brzo pretraživanje.
`finger` - korištenjem *finger* servisa, ispisuje informacije o traženom korisniku.
`file` - prikazuje informacije o vrsti željene datoteke: tekstualna, binarna, načinu kako je kodirana, …
`fixfiles` – *SELinux* naredba za osvježavanje sigurnosnih konteksta datoteka na datotečnom sustavu.
`for` - ugrađena naredba koja se koriste za izradu petlje u ljusci direktno ili u *shell* skripti, pogledajte i `while`.
`free` - ispisuje zauzeće RAM, SWAP, *buffer* i *cache* memorije.
`fsck` - za provjeru datotečnog sustava na željenoj particiji, od grešaka. Pogled. i: `fsck.ext3`, `fsck.ext4`, `xfs_check`.
`fstrim` - za čišćenje blokova koji su obrisani na SSD i NVMe diskovima (tzv. ***TRIM/Discard***).
`fuser` - program za identifikaciju procesa prema datotekama ili *socketima.*
`gdisk` - program za particioniranje diska (rad s particijama), koji podržava i ***GPT*** vrstu particija.
`getcap` - program za rad sa naprednim mogućnostima datoteka i procesa.
`getconf` - program za rad s POSIX konfiguracijskim parametrima i varijablama sustava.
`getent` - program za rad s unosima iz NSS (*Name Service Switch*) sustava.
`getfacl` – program za postavljanje ACL lista u datotečnom sustavu (za datoteke i direktorije).
`getsebool` - program za uključivanje vrijednosti i postavki *SELinux* sustava.
`git` – naredba za rad s GIT sustavom za verzioniranje programskog kôda.
`gpasswd` – za administraciju korisničkih grupa te kreiranje lozinke za korisničke grupe (prema potrebi).
`gpg` – (*GNU Privacy Guard*) softver za kriptografiju.
`grep` - za filtriranje odnosno pretraživanje ispisa ili datoteke (teksta), prema traženom pojmu ili ključnoj riječi.
`groupadd` - program za kreiranje novih korisničkih grupa (Engl. *Group add*).
`groupdel` - program za brisanje korisničkih grupa (Engl. *Group delete*).
`groupmod` - program za mijenjanje korisničkih grupa (Engl. *Group modify*).
`groups` - ispisuje kojim korisničkim grupama pripada trenutni korisnik.
`growpart` - proširuje datotečni sustav na particiji diska, do kraja diska (Engl. *Grow partition*).
`gzip` - program za komprimiranje ili dekomprimiranje datoteka, korištenjem *GZIP* algoritma.
`gunzip` - program za dekomprimiranje *GZIP* datoteka.
`hdparm` – naredba za postavljanje ili čitanje parametara rada diska (ATA, SATA ili SAS).
`head` - ispisuje sadržaj prvih nekoliko (standardno 10) redova datoteke.
`hexdump` - program za prikaz datoteka u raznim formatima; primjerice: heksadecimalnom, binarnom, oktalnom …

hostname - ispisuje ili postavlja ime računala (*hostname*).
host - program koji radi *DNS upit (DNS lookup)* imena računala/domene ili reverzni DNS upit.
htop - prikazuje sve pokrenute programe/procese u realnom vremenu te osvježava prikaz. Pogledajte i: top i atop.
hwclock - program za rad sa hardverskim satom (*RTC*) računala (čitanje ili postavljanje vremena/sata).
i2cdetect - program za detektiranje senzora, preko **I2C** ili **SMBus** sabirnice.
id - ispisuje tko je trenutni korisnik (ime i ***UID***) te kojim korisničkim grupama pripada (***GID***).
if - ugrađena naredba (unutar *bash ljuske*) za uvjete unutar skripti (if-then-else): ovo je logički uvjet *"AKO"*.
ifconfig - koristi se za konfiguraciju ili prikaz mrežnih sučelja i/ili njihovih parametara rada (*IP adresa, netmask, GW,…*).
ifenslave – naredba za kreiranje i rad s ***bond*** (team) mrežnim sučeljima.
ifup - koristi se za podizanje (uključivanje) mrežnog sučelja.
ifdown - koristi se za spuštanje (isključivanje) mrežnog sučelja.
insmod - program s kojim možemo učitati željeni kernel modul. Pogledajte i modprobe, lsmod, modinfo, kao i rmmod.
init - program za promjenu načina pokretanja operacijskog sustava, odnosno promjenu *init moda* (0,1,2,3,4,5,6).
intel-microcode2ucode - program za prikaz svi *Intel CPU* mikrokod datoteka.
ionice - naredba za promjenu ulazno/izlaznog (I/O) prioriteta procesa.
iostat - naredba za prikaz diskovnih (I/O) statistika sustava.
ip - naredba koja se koristi za konfiguraciju mrežnih sučelja i/ili parametara*: ip adrese*, *rute*, *multicast postavki* i slično.
ipcs - naredba koja nam prikazuje statistike ***IPC*** (*Inter Process Communication*) sustava.
iperf3 - naredba koja se koristi za testiranje performansi mreže.
ipmitool - naredba za rad s ***IPMI*** uređajima.
ipset - naredba za rad s velikim skupovima pravila vatrozida.
iptables - naredba/alat za konfiguraciju vatrozida (filtriranje mrežnih paketa i NAT).
ipvsadm - naredba/alat za konfiguraciju *IP virtual server* funkcionalnost balansiranja opterećenja (Engl. *Load Balancing*).
ipvsadm-save - naredba/alat za snimanje konfiguracije *IP virtual server* funkcionalnosti balansiranja opterećenja.
iscsiadm – naredba za spajanje na ISCSI odredišta (*targete*).
iucode_tool - program za rad s CPU mikrokod datotekama.
join - prikazuje redove iz više datoteka, koji imaju zajedničke nazive ili polja.
journalctl - ispisuje sve sistemske poruke, koje se logiraju na sustav (rad programa/servisa/sistemskih sustava, …).
kexec i kdump - naredbe za „*kernel dump*/Crash dump" funkcionalnost.
keepalived - naredba/servis za upotrebu ***VRRP*** protokola i osnovno raspodjeljivanje prometa (*Load Balancing*).
kill - naredba za slanje signala procesima (pr. zaustavljanje ili ubijanje procesa/programa) prema *process ID* broju (***PID***).
killall - za zaustavljanje procesa prema imenu procesa.
last - ispisuje statistike o logiranju korisnika na sustav.
ldconfig - naredba za prikaz poveznica s bibliotekama sustava (Engl. *System library's*) te za konfiguraciju biblioteka.
ldd - naredba koja nam daje podatak o tome koje sve biblioteke određeni program treba za rad.
less - interaktivno ispisuje sadržaj datoteke ili ispisa druge naredbe, ima ugrađeno pretraživanje i listanje (slično kao more).
links - web preglednik u terminalu.
lldpcli – sučelje ili naredba za rad s ***LLDP*** protokolom.
ln - naredba za kreiranje simboličkih poveznica (engl. *Symbolic link*) ili tvrdih poveznica na postojeću datoteku.
lnstat - naredba za prikaz mrežnih statistika.
localectl - naredba za konfiguraciju regionalnih postavki sustava te regionalnih postavki tipkovnice.
loadkeys - program za učitavanje postavki konfiguracije tipkovnice.
locale - program za regionalne postavke (kodna stranica, jezik, valuta).
locate - pronalazi sve instance datoteke željenog imena, na disku, pomoću baze koja se kreira s naredbom updatedb.
login - naredba zadužena za prijavljivanje korisnika na sustav (logiranje).
loginctl - naredba za konfiguraciju podsustava za logiranje korisnika.
logger - program s kojim možemo poslati neku sistemsku poruku koja će biti logirana (spremljena u log) od strane sustava.
lsattr - naredba za izlistanje naprednih atributa datoteka ili direktorija (pogledajte i chattr).
lsinitrd - program koji može ispisati sadržaj *boot RAM imagea* (*initramfs*a). Pogledajte i dracut
lsblk - program za ispis statusa blok uređaja: diskova i njihovih particija te točki montiranja.
lslogins - naredba koja nam daje podatke o korisnicima sustava (Engl. *Users*).
lsmod - program za izlistavanje svih učitanih kernel modula (*drivera*). Pogledajte i modinfo, rmmod, modprobe
lsns - program za rad sa izoliranim prostorima Linuxa (Engl. *Linux namespace*).
lsof - program koji nam daje ispis svih otvorenih klasičnih i posebnih datoteka *(pr. Unix socket*, mrežni *socket, file deskriptor …*)
lpr - naredba za ispis na štampač (*printer*).
lpq - prikaži sve datoteka koje se nalaze na čekanju u nîzu za ispis na štampanje.
lprm - obriši datoteku koja je u nizu čekanja za ispis na štampanje
ls - ispiše popis svih datoteka u trenutnom ili navedenom direktoriju (Engl. *List directory contents*).
lspci - program koji nam daje podatke o *PCI* i *PCI Express* uređajima spojenim na računalo/matičnu ploču.
lsscsi - program koji nam daje podatke o SCSI i SATA uređajima spojenim na računalo.
lstopo - program koji nam daje informacije o topologiji i arhitekturi našeg centralnog procesora (CPU).
lsusb - program koji nam daje informacije o spojenim USB uređajima na USB sabirnicu računala.
ltrace - program s kojim možemo pratiti koje biblioteke koristi željeni program, kao i pratiti sistemske pozive programa. Pogledajte i strace.

`lvcreate` - program za LVM polje diskova s kojim kreiramo LVM2 logički volumen.
`lvextend` - program za LVM polje diskova s kojim proširujemo kreiramo LVM2 logički volumen.
`lvremove` - program za LVM polje diskova s kojim brišemo LVM2 logički volumen.
`lvs` - program za LVM polje diskova s kojim ispisujemo informacije o LVM2 logičkim volumenima.
`lvscan` - program za LVM polje diskova s kojim ispisujemo sve logičke volumene u svim grupama (ako ih ima).
`lxc-attach` - za spajanje na konzolu LXC Linux kontejnera ili izvršavanje naredbi unutar njega.
`lxc-checkconfig` - za provjeru kompatibilnosti računala za podršu za LXC Linux kontejnere.
`lxc-create` - za kreiranje LXC Linux kontejnera iz predloška za određenu distribuciju linuxa.
`lxc-destroy` - za nepovratno brisanje LXC Linux kontejnera s lokalnog računala.
`lxc-info` - za informacije o pokrenutom LXC Linux kontejneru.
`lxc-ls` - za ispis svih dostupnih LXC Linux kontejnera (na lokalnom računalu).
`lxc-stop` - za zaustavljanje LXC Linux kontejnera.
`lxc-start` - za pokretanje LXC Linux kontejnera.
`lxc-top` - za ispis statistika rada LXC Linux kontejnera (CPU/RAM/Disk...)m poput Linux naredbe `top`.
`mail` - klijent elektroničke pošte u terminalu (za čitanje ili slanje elektroničke pošte).
`mailx` - klijent elektroničke pošte u terminalu (za čitanje ili slanje elektroničke pošte).
`man` - naredba s kojom dobivamo upute o korištenju drugih naredbi ili programa.
`make` - program za kompiliranje i povezivanje te izradu izvršnih programa iz njihovog izvornog programskog kôda.
`md5sum` - za izradu *MD5 provjernog zbroja (checksum)* datoteke.
`merge` - program za povezivanje više datoteka u jednu (Engl. *Merge*).
`mc` - program za vizualno baratanje s datotekama i direktorijima (Engl. *Midnight Commander*)
`mcelog` - program i servis za očitanje *machine check events* (**MCE**) stanja/događaja.
`mdadm` – naredba za rad s *Metadata device* (***MD***) logičkim volumenima.
`mkdir` - naredba za izradu novog direktorija (mape) (Engl. *Make directory*).
`mkfs` - program s kojim formatiramo željenu diskovnu particiju. Za svaki datotečni sustav, postoji zaseban program, poput: `mkfs.ext3` (za ***ext3***), `mkfs.ext4` (za ***ext4***), `mkfs.xfs` (za ***XFS***), `mkfs.btrfs` (za ***BTRFS***), …
`mkfifo` - naredba s kojom izrađujemo FIFO datoteku odnosno tzv. *named pipe* datoteku (Engl. *Make Fifo*).
`mktemp` - naredba s kojom izrađujemo privremenu (engl. *temporary*) datoteku ili direktorij.
`mknod` - naredba s kojom izrađujemo posebne blok ili karakter uređaje (u `/dev/` direktoriju) (Engl. *Make node*).
`mkswap` – naredba za formatiranje particije (ili posebne datoteke) sa *swap* datotečnim sustavom.
`modinfo` - daje nam informacije o željenom kernel modulu. Pogledajte i: `lsmod`, `rmmod`, `modprobe`, `insmod`
`modprobe` - za učitavanje željenih kernel modula. Pogledajte i `lsmod`, `rmmod`, `modinfo` i `insmod`.
`more` - ispisuje sadržaj datoteke na standardni izlaz. Ima ugrađeno i listanje po stranicama (stranica po stranica).
`mount` - za *montiranje* formatirane particije u neki direktorij ili prikaz svih onih koje su trenutno *montirane (mountane).*
`mpstat` - za praćenje opterećenja CPU-a te prikaz korištenje *soft i hard interrupta* u stvarnom vremenu.
`mt` - naredba za rad s tračnim uređajem za pohranu podataka na trake (*backup*) (Engl. *Magnetic Tape operation*).
`mv` - za preimenovanje ili prebacivanje datoteka na drugu lokaciju, isto je moguće i s direktorijima (mapama) (Engl. *Move*).
`nl` - za ispis datoteka, a pri tome svaki novi redak označuje s brojem (Engl. *Number lines*).
`nc` i `ncat` - program za rad s TCP/UDP ili *Unix socketima* (Engl. *Net cat*)
`netstat` - daje nam razne statistike o mreži i mrežnim servisima (Engl. *Network statistics*).
`newgrp` – omogućava privremenu promjenu primarne korisničke grupe.
`nice` - za promjenu prioriteta izvršavanja novog programa (pogledajte i `renice`).
`nft` - naredba za rad s vatrozidom (*firewall*) nove generacije (Engl. *nftables*)
`nfsiostat` - naredba koja nam daje statistike rada odnosno mjeri performanse rada NFS sustava.
`nfsstat` - još jedna naredba koja nam daje statistike rada odnosno mjeri performanse rada NFS sustava.
`nfsiostat-sysstat` – i još jedna naredba koja nam daje statistike rada odnosno mjeri performanse rada NFS sustava.
`nohup` - pokretanje programa koji će se ostati izvršavati, iako se veza prekine ili se odlogiramo (Engl. *No Hang up*).
`nslookup` - daje nam informacije o domeni ili *Hostname-to-IP* razlučivanju (Engl. *Name service lookup*).
`nstat` - daje nam statistike mrežnih paketa (Engl. *Network statistics*) .
`ntpdate` - za ručno povezivanje s udaljenim NTP poslužiteljem i usklađivanje lokalnog sata prema NTP izvoru.
`ntpq` - program za ispis statusa NTP klijenta i veze prema NTP poslužiteljima (Engl. *NTP query*).
`numactl` - program za prikaz *NUMA* arhitekture te podešavanje NUMA CPU afiniteta programa (Engl. *NUMA control*).
`numastat` - program za prikaz *NUMA* arhitekture i statistike rada (Engl. *NUMA statistics*).
`nvme` - program za rad s **NVMe** diskovima.
`od` - program za prikaz datoteka o binarnom, oktalnom i drugim formatima.
`openssl` – program za rad s kriptografskim komponentama sustava (obično za TLS/SSL).
`ovs-vsctl` - program za konfiguraciju *Open vSwitch* sustava.
`perf` - program za praćenje performansi programa ili dijelova sustava (Engl. *Performance*).
`partprobe` - program koji informira OS o promjenama na particijama diskova.
`passwd` - naredba za promjenu (svoje) lozinke, ili nečije druge, ako smo *root* (administrator) korisnik (Engl. *Password*).
`paste` - prikazuje povezan sadržaj dvije datoteke: jedne uz drugu.
`pine` - program (u terminalu) za elektroničku poštu i *news* grupe.
`pidof` - program koji nam na osnovu imena pokrenutog programa/procesa, može dati njegov *PID* broj (Engl. *PID of*).
`pidstat` - program koji nam daje statistike o linux procesima (pokrenutim programima) (Engl. *PID statistics*).

`ping` - šalje *ping* poruku udaljenom računalu te prikazuje dostupnost i vrijeme odziva druge strane u komunikaciji.
`pldd` - saznaje koje su dijeljene biblioteke potrebne za rada već pokrenutog programa, pogledajte i `ldd`.
`pmap` - prikazuje memorijsku kartu (mapu) pokrenutog programa (procesa) (Engl. *Process [memory] map*)
`podman` – program za rad s *Podman* (*OCI*) kontejnerima [pokretanje, zaustavljanje, …].
`printenv` - prikazuje sve varijable sustava (Engl. *Print environment*).
`prlimit` - naredba za ograničavanje resursa sustava koje može dobiti određeni program (proces) (Engl. *Print limits*).
`ps` - prikazuje sve pokrenute programe i njihove parametre rada te zauzeće resursa sustava.
`pvcreate` - za rad s LVM poljem diskova. S njim kreiramo LVM2 fizički volumen (***PV***) (Engl. *Physical Volume*).
`pvs` - program za LVM polje diskova s kojim ispisujemo LVM2 fizičke volumene (***PV***) (Engl. *Physical Volume statistics*).
`pwd` - ispisuje trenutni radni direktorij (mapu) u kojem se nalazimo (Engl. *Print working directory*).
`python` - *interpreterski* interaktivni objektno orijentirani programski jezik široke namjene.
`rasdaemon` - servis za očitavanje i pohranu *machine check events* (**MCE**) stanja/događaja.
`ras-mc-ctl` - program za očitavanje *machine check events* (**MCE**) stanja/događaja.
`read` - ugrađena naredba (u ***bash*** *shell*) koja čita liniju po liniju (red po red) teksta sa standardnog ulaza.
`readelf` - naredba koja nam da informacije o izvršnim linux datotekama: takozvanim *ELF (Engl. Read ELF*).
`rear` – naredba (i mehanizam) za izradu sigurnosnih kopija cijelog Linux sustava te za povrat podataka nakon havarije.
`renice` - za promjenu prioriteta izvršavanja već pokrenutom programu (pogledajte i naredbu `nice`).
`resize2fs` - naredba za povećavanje ili smanjivanje datotečnog sustava vrste Linux ***EXT*** (*ext2*, *ext3* ili *ext4*).
`readlink` - program koji ispisuje poveznicu simboličkog linka (*symlinka*) odnosno odredišne datoteke na koju on pokazuje.
`readonly` - s njom se varijable ili funkcije označavaju kao nepromjenjive (ne mogu se više mijenjati).
`reboot` - naredba za restart ili gašenje računala.
`reset` - naredba za ponovno pokretanje (*reinicijalizaciju*) trenutno pokrenutoga terminala (u kojem radimo).
`restorecon` - naredba vraćanje standardnih *SELinux* postavki datoteka.
`route` - naredba za rad s tablicama usmjeravanja i rutama (pr. za dodavanje, brisanje ili ispis).
`rsync` - program za sinkronizaciju (napredno kopiranje) datoteka s jednog mjesta na drugo, lokalno ili preko mreže.
`rm` - naredba za brisanje datoteka ili direktorija (mapa) (Engl. *Remove*).
`rmdir` - naredba za brisanje direktorija (mapa) (Engl. *Remove directory*).
`rmmod` – za isključivanje učitanih kernel modula. Pogledajte naredbe: `lsmod`, `modprobe`, `modinfo`, `insmod`.
`rpm` - program za rad sa sfotverskim ***RPM*** paketima: instalacija, deinstalacija, pronalaženje, nadogradnja i slično.
`sar` - program za prikupljanje raznih statistika sustava: CPU, RAM, DISK, NET, … (Engl. *System analyzing and reporting*).
`screen` - program za korištenje ili povezivanje više programa kroz jedno sučelje. On osigurava izvršavanje programa čak i u slučaju ispada veze prema drugom udaljenom računalu. Pogledajte i naredbu `nohup`.
`sed` - program za baratanje s tekstom, poput mijenjanja, dodavanja ili brisanja dijelova teksta (Engl. *Stream line editor*).
`seinfo` - naredba koja nam daje informacije o *SELinux* postavkama Linuxa.
`seekwatcher` - program za kreiranje grafa snimljene statistike rada diskovnog podsustava.
`semanage` - naredba za rad sa *SELinux* postavkama Linuxa.
`semodule` - naredba za rad sa *SELinux* modulima Linuxa.
`sensors` - naredba za ispis očitanja **I2C** ili **SMBus** senzora.
`sensors-detect` - naredba za pronalazak **I2C** ili **SMBus** senzora.
`sestatus` - naredba koja nam prikazuje status rada *SELinux* sustava
`setcap` - program za rad s naprednim mogućnostima (Engl. *Capabilities*) datoteka i procesa (pokrenutih programa).
`setenforce` - naredba za promjenu načina rada *SELinux* sustava.
`setfacl` – program za čitanje ACL lista u datotečnom sustavu (za datoteke i direktorije).
`setsebool` - naredba za postavljenje opcija unutae *SELinux* sustava.
`seq` - program za kreiranje željenog nîza brojeva (Engl. *Sequence of numbers*).
`service` - rad sa Linux servisima/*daemonima* (pokretanje, stopiranje, restart) na *RedHat*/*CentOS* 6.x, ali i podržan na 7.x/8.x.
`sestatus` - naredba s kojom dobivamo ispis statusa rada *SELinux* sustava.
`set` - ugrađena naredba (u ***bash*** *ljusku*) s kojom se mogu definirati vrijednosti, opcije ili parametri pozicije, unutar ljuske.
`setenforce` - naredba s kojom mijenjamo model rada *SELinux* sustava.
`setpci` - program za konfiguriranje uređaja na *PCI/PCI express* sabirnici.
`shutdown` - naredba s kojom se može ugasiti ili restartati računalo.
`sha1sum` - za izradu *SHA-1 provjernog zbroja (Engl. checksum)* datoteke.
`sha256sum` - za izradu *SHA-256 provjernog zbroja (Engl. checksum)* datoteke.
`sha512sum` - za izradu *SHA-512 provjernog zbroja (Engl. checksum)* datoteke.
`shopt` - naredba ugrađena u *bash* ljusku s kojom možemo uključivati napredne opcije ljuske.
`showmount` - za prikaz informacije u točkama montiranja (*mount*) za NFS dijeljeni mrežni datotečni sustav.
`skopeo` - za rad s diskovnim slikama *Podman* kontejnera i kontejnerskim registrima.
`slabtop` - prikazuje informacije o takozvanoj ***slab*** međumemoriji kernela u stvarnom vremenu (osvježava se).
`sleep` - odgađa izvršavanje programa za željeno vrijeme (sekundi).
`sntp` - naredba (klijent) za dohvaćanje točnog vremena s NTP ili SNTP poslužitelja (pogledajte i `ntpdate`)
`sort` - naredba za sortiranje teksta prema raznim kriterijima. Omogućava i sortiranje prema stupcima.
`source` - naredba za učitavanje funkcijskih i drugih skripti, njihovih funkcija i varijabli (unutar skripte ili ljuske).
`split` - razlama veću datoteku u više manjih, prema željenoj veličini.

`ss` - naredba s kojom dobivamo razne statistike mrežnih paketa, poput naredbe `netstat`, ali detaljnije (Engl. *Socket statistic*).
`stat` - prikazuje informacije o datotekama: veličina, vrijeme pristupa, vrijeme kreiranja i vrijeme modificiranja (Engl. *Status*).
`stty` - naredba za konfiguraciju ili ispis postavki terminala.
`su` - naredba s kojom možemo postati neki drugi korisnik odnosno prijaviti se kao drugi korisnik (Engl. *Substitue user*).
`sudo` - naredba s kojom možemo izvršavati programe kako neki drugi korisnik, samo ako imamo pravo na to (Engl. `su` *do*).
`sos` - naredba za izradu *sos izvještaja* (stanja rada sustava).
`ssh` - ***ssh*** klijent za spajanje na udaljeno računalo odnosno na *ssh* poslužitelj (Engl. *Secure Shell*).
`ssh-keygen` - alat za rad s SSH ključevima; primjerice za kreiranje SSH ključeva.
`ssh-copy-id` - naredba za kopiranje SSH ključeva na udaljeni SSH poslužitelj.
`strace` - program s kojim možemo pratiti koje sistemske signale i pozive koristi željeni program koji pratimo.
`sync` - za sinkronizaciju odnosno snimanje svih podataka iz privremene (*RAM cache*) memorije na površinu diska.
`swapon` - naredba za uključivanje *SWAP* particije, ali i ispis upotrebe SWAP particija.
`swapoff` - naredba za isključivanje *SWAP* particije.
`sysctl` - za mijenjanje parametara i opcija kernela, a samim time optimizaciju sustava.
`systemctl` - za konfiguraciju (***Systemd***) servisa odnosno za njihovo pokretanje/zaustavljanje (Engl. *System control*).
`systool` - daje nam informacije o uređajima na računalu (hardveru) prema sabirnici, klasi i topologiji.
`tail` - ispisuje sadržaj željene datoteke, počevši od kraja. Moguć je i prikaz kako se datoteka popunjava (dopunjava).
`tar` - program za kreiranje (*TAR*) arhive, koja čuva i ovlasti svih datoteka i direktorija. Moguće je uključiti i komprimiranje.
`targetcli` – naredba za rad sa SAN storage sustavima.
`taskset` - program za modifikaciju afiniteta programa/procesa za izvršavanje na pojedinoj CPU jezgri ili jezgrama.
`tc` - za postavljanje ograničenja na mrežni promet (brzinu, gubitak paketa i slično) (Engl. *Traffic control*).
`tftp` - naredba za dohvaćanje datoteka pomoću ***TFTP*** protokola. Dakle ovo je ***TFTP*** klijent.
`timedatectl` - naredba za čitanje ili promjenu sistemskog vremena/sata (Engl. *Time/Date control*).
`toe` - naredba za ispis podržanih vrsta terminala; primjerice: ***sun/vt100/vt102/vt220/…*** (Engl. *Table of (terminfo) entry's*).
`top` - prikazuje sve pokrenute programe/procese u realnom vremenu te osvježava prikaz. Pogledajte i `htop` i `atop`
`tput` - naredba za konfiguraciju terminala.
`tr` - za pretvaranje ili brisanje željenih karaktera (slova/znakova), pr. pretvorbu iz velikih u mala slova (Engl. *Translate*).
`tracepath` - za testiranje vremena odziva do svakog usmjerivača u nîzu do odredišnog računala, poput slijedeće naredbe.
`traceroute` - za testiranje mreže odnosno vremena odziva do svakog usmjerivača u nîzu do odredišnog računala.
`trap` - naredba za rad i testiranje slanja i dohvaćanja sistemskih signala.
`truncate` – naredba za kreiranje datoteke određene veličine ili brisanje njihovog sadržaja.
`tty` - naredba za prikaz informacija o terminalu (pr. koji trenutno terminal koristimo).
`touch` - kreiranje datoteke željenog imena (prazne) ili za promjenu vremenskih oznaka datoteke (Engl. *Touch*).
`toe` – naredba za ispis vrsta podržanih terminala.
`tsort` - naredba za topološko sortiranje teksta.
`type` - naredba s kojom možemo provjeriti pripadnost neke naredbe: dolazi li unutar ljuske ili kao zasebna naredba/datoteka.
`udevadm` - menadžment alat za uređaje (*udev*) odnosno hardver na našem računalu te za konfiguraciju rada hardvera (uređaja).
`ulimit` - naredba za postavljanje ograničenja na sustav: prema korisnicima, grupama ili drugim resursima sustava.
`umount` - naredba s kojom brišemo točku montiranja (*mount point*) particije u odredišni direktorij (mapu).
`unalias` - naredba s kojom brišemo određeni postavljeni `alias`.
`uname` - ispisuje informacije o inačici linux kernela i sustava.
`uniq` - naredba za pronalaženje (prikazivanje ili ne prikazivanje) linija/redova teksta koji se ponavljaju (Engl. *Unique*).
`unset` - poništava postavljene određene sistemske varijable.
`unshare` – naredba za rad s izoliranim prostorima Linuxa (*Mount, UTS, Network, IPC*, …).
`unsquashfs` – naredba za rad sa *SQUASHFS* datotečnim sustavom ili diskovnom slikom.
`until` - ugrađena naredba (u *bash shell*) za petlju, koja se izvršava sve dok nije zadovoljen neki uvjet.
`unzip` - program za dekomprimiranje datoteka upotrebom ZIP algoritma. Pogledajte i `zip`.
`unzstd` - program za komprimiranje i dekomprimiranje datoteka upotrebom **ZSTD** algoritma. Pogledajte i `zstd`.
`unxz` - program za dekomprimiranje datoteka upotrebom ***xz*** (**LZMA**) algoritma. Pogledajte i `xz` .
`updatedb` - naredba s kojom osvježavamo bazu (*mlocate*) s popisom datoteka, koja se koristi za naredbu `locate`.
`update-pciids` - naredba s kojom osvježavamo PCI ID bazu PCI identifikatora uređaja (s interneta).
`uptime` - ispisuje koliko vremena je računalo aktivno odnosno od kada je uključeno.
`useradd` - program za dodavanje novih korisnika na sustav ili za promjenu podataka o korisničkom računu.
`userdel` - program za brisanje korisnika (korisničkog računa) iz sustava.
`usermod` - program za modifikacije korisnika odnosno korisničkih računa.
`uuidgen` - program za kreiranje jedinstvenog identifikator (tzv. ***UUID*** *oznake*).
`vgcreate` - naredba za rad s *LVM2* logičkim diskovima; za kreiranje *volume grupe (**VG**).*
`vgdisplay` - naredba za rad s *LVM2* logičkim diskovima; za ispis konfiguriranih *volume grupa (**VG**).*
`vgextend` - naredba za rad s *LVM2* logičkim diskovima; za proširenje *volume grupe (**VG**).*
`vgreduce` - naredba za rad s *LVM2* logičkim diskovima; za smanjivanje *volume grupe (**VG**).*
`vgremove` - naredba za rad s *LVM2* logičkim diskovima; za brisanje *volume grupe (**VG**).*
`vgs` - naredba za rad s *LVM2* logičkim diskovima; za ispis *volume grupa (**VG**).*
`vi`- interaktivni uređivač teksta (Engl. *Visual Editor*).
`vim` - napredni `vi` uređivač teksta. On je danas standard i obično se pozivanjem `vi` pokreće `vim` (Engl. *Vi Improved*).

`vimdiff` - uspoređuje dvije ili tri datoteke i prikazuje razlike među njima u `vim` uređivaču teksta (obojano).
`vmstat` - program za prikaz statistike upotrebe sustava virtualne memorije, ali i CPU-a te ulazno-izlaznog (I/O) sustava.
`virt-sysprep` – program za čišćenje konfiguracije Linuxa kao virtualnog računala (za pripremu kao predloška OS-a).
`visudo` - naredba (uređivač) za rad s konfiguracijskom datotekom za *sudo* sustav.
`vtysh` – naredbena ljuska ***FRR*** sustava koja nam nudi način rada kao da se nalazimo na usmjerivaču (routeru).
`w` - ispisuje tko je sve spojen (prijavljen/logiran) na ovo računalo i što trenutno radi (pogledajte i naredbu `who`).
`wall` - naredba za slanje (tekstualnih) poruka svim prijavljenim korisnicima sustava (računala) na terminal.
`watch` - pokreće željenu naredbu ili skriptu svakih dvije (konfigurabilno) ili više sekundi.
`wc` - program za prebrojavanje riječi/slova (Engl. *Word count*).
`wget` - program za kopiranje (*download*) datoteka, cijele web stranice ili željenih dijelove preko HTTP/S ili FTP protokola.
`whatis` - program koji pretražuje bazu s naredbama na sustavu i njihovim pripadajućim kratim opisom, za željenu naredbu.
`whereis` - ispiše sve lokacije tražene datoteke na disku, iz kreirane lokalne *whereis* baze podataka.
`which` - ispiše gdje se nalazi željeni program, uz ispis njegove točne putanje. Primjerice: `/usr/sbin/`
`while` - ugrađena naredbe koja se koriste za izradu petlje u *shellu* direktno ili u *shell* skripti, pogledajte i `for`
`whoami` - ispisuje s kojim korisničkim računom (kao koji korisnik) smo trenutno logirani na sustav.
`whois` - saznajmo više informacija o traženoj domeni, preko *whois* servisa.
`who` - daje nam informacije o tome tko je sve logiran na sustav (pogledajte i `w`).
`write` - šalje željenu poruku točno određenom korisniku koji je prijavljen (logiran) na sustav.
`xargs` – za preuzimanje argumenata prethodnog programa sa standardnog ulaza i prosljeđivanje nekom drugom programu.
`xz` - program za komprimiranje i dekomprimiranje datoteka upotrebom ***xz*** (**LZMA**) algoritma. Pogledajte i `unxz`.
`xfs_admin` – program za promjenu parametara **XFS** datotečnog sustava.
`xfs_db` - program za ručno popravljanje grešaka/nekonzistentnosti na ***XFS*** datotečnom sustavu.
`xfs_growfs` - program za proširenje kapaciteta **XFS** datotečnog sustava.
`xfs_repair` - program za automatsko popravljanje grešaka/nekonzistentnosti na ***XFS*** datotečnom sustavu.
`yum` - program za rad s *RedHat*/*CentOS*/*Fedora* Linux softverskim paketima, koji se oslanja na program `rpm`.
`zcat` - program s kojim možemo ispisati (na ekran) sadržaj tekstualne datoteke koja je komprimirana sa ***GZIP*** algoritmom.
`zstd` - program za komprimiranje i dekomprimiranje datoteka upotrebom **ZSTD** algoritma. Pogledajte i `unzstd`.
`zip` - program za komprimiranje i dekomprimiranje datoteka upotrebom **ZIP** algoritma. Pogledajte i `unzip`.
`zless` - program s kojim možemo ispisati (na ekran) sadržaj tekstualne datoteke koja je komprimirana sa ***GZIP*** algoritmom.
`quota` - provjerava ograničenja diskovnog prostora za korisnika/e, samo ako je postavljena disk kvota.
`quotacheck` - provjerava stanje disk kvote (ako su postavljene).
`quotaoff` - isključuje disk kvotu i `quotaon` - uključuje disk kvotu (ako je prethodno definirana).
`qemu-img` - program za rad s virtualnim diskovima za virtualna računala: kreiranje, proširivanje i konverzija između formata (***RAW, QCOW2, VMDK, VDI, VPCS, VHD, …***) i slično.

**Izvori informacija: `man` stranice navedenih naredbi.**

# 29.3. Popis kratica i pojmova

**U ovoj cjelini navest ćemo sve kratice, pojmove i tehnologije koje smo spominjali u knjizi, ali i one koje nismo spomenuli, a smatramo ih važnima za vaš profesionalni razvoj.**

## 0-9

**10B2** - *10 Base2,* varijanta 10Mbps *etherneta* koja je koristila koaksijalni kabel.
**10BASE5** - *10 Base 5,* varijanta 10Mbps *etherneta* koja je prva koristila **UTP** kabel.
**386, 486, 586, 686** – Intel procesori redom: **80386**, **80486**, **Pentium** te **686** za: Pentium Pro, Pentium II/III i Pentium 4.
**802.1 –** ***LAN/MAN mreže (OSI sloj dva [OSI 2]):***

- **802.1ab** - *Link Layer Discovery Protocol (**LLDP**).*
- **802.1ad** - ***QinQ*** *(ugniježđeni VLAN).*
- **802.1ak** - *Multiple Registration Protocol (**MRP**) i Generic Attribute Registration Protocol (**GARP**).*
- **802.1AX-2008 –** LACP. Pogledajte **802.3ad**.
- **802.1d** - *Ethernet MAC bridges (Bridging, Spanning Tree Protocol).*
- **802.1p –** *Quality of Service* **(QoS).**
- **802.1x** - *Port-based Network Access Control (**PNAC**) Security.*
- **802.1Q –** *Virtual Local Area Network* **(*VLAN*).** Pogledajte **VLAN.**

**802.2 -** ***Logical Link Control (LLC) i Subnetwork Access Protocol (SNAP) mrežni okviri.***
**802.3 - Ethernet mrežni okviri:**

- **802.3ab** - *1000Mbps (1Gbps) Ethernet [1000BASE-T].*
- **802.3ac** - *Odnosi se na* **802.1Q.**
- **802.3ad** - *Link Aggregation Control Protocol* **(LACP)**.
- **802.3an** - *10Gbps Ethernet [10GBASE-T].*
- **802.3ba-2010, 802.3bg-2011,802.3bj-2014,802.3bm-2015,802.3cd-2018 -** *100Gbps Ethernet.*
- **802.3bq-2016** - *25Gbps Ethernet [25GBASE-T].*
- **802.3bq-2016** - *40Gbps Ethernet [40GBASE-T].*
- **802.3bs** - *200Gbps Ethernet.*
- **802.3bz-2016** - *2.5Gbps Ethernet [2.5GBASE-T].*
- **802.3bz-2016** - *5Gbps Ethernet [5GBASE-T].*
- **802.3i** - *10Mbps Ethernet [10BASE-T].*
- **802.3u** - 100Mbps Ethernet [100BASE-TX].
- **802.3x** - *Flow control, kontrola protoka*.
- **802.1Q –** Mrežni standard u kojem je definirana podrška za virtualne lokalne mreže. Pogledajte **VLAN**.

**802.11** – **Bežične (*wireless*) mreže:**

- **802.11a –** Bežična mreža propusnosti 54Mbps.
- **802.11b -** Bežična mreža propusnosti 11Mbps.
- **802.11g -** Bežična mreža propusnosti 54Mbps.
- **802.11n -** Bežična mreža propusnosti 600Mbps.
- **802.11ac -** Bežična mreža propusnosti 6.8–6.93 Gbit/s.
- **802.11ad -** Bežična mreža propusnosti 7.14–7.2 Gbit/s.
- **802.11ax -** Bežična mreža propusnosti 11 Gbit/s.

**10G-PON** - noviji **FTTH** - **PON** standard koji omogućuje propusnosti do 10Gbps. Pogledajte **PON** i **GPON**.

## A

**AAA** - *Authentication Authorization and Accounting,* mehanizmi za kontrolu i nadzor pristupa na mrežu ili mrežne elemente.
**ABI –** *Application Binary Interface,* sučelje između dva binarna programa. Pogledajte i **KMI**.
**Access port** - označava mrežno sučelje preklopnika koje pripada samo i isključivo jednoj **VLAN** mreži. Pogledajte **VLAN**.
**ACID –** *Atomicity, Consistency, Isolation, Durability*, svojstva uobičajena za SQL baze podataka i transakcijske sustave.
**ACK** — *ACKnowledgement*, potvrdna poruka.
**ACL** — *Access Control List,* mehanizmi zaštite pristupa na mrežne resurse (pr. prema IP adresi, portu i slično).
**ACPI** — *Advanced Configuration and Power Interface*, sučelje za konfiguraciju i postavke potrošnje (napajanja).
**ACPI Power management timer** — jedan od izvora vremena. Pogledajte: **APIC**, **HPET**, **RTC**, **PIC** i **TSC**.
**ADSL** — *Asymmetric Digital Subscriber Line*, varijanta asinkronog **DSL**-a (tehnologije za spajanje na Internet).
**AES** — *Advanced Encryption Standard,* kriptografski algoritam za simetrično kriptiranje (šifriranje) i dekriptiranje podataka.
**AGP** — *Accelerated Graphics Port*, sučelje za spajanje grafičkih kartica (stariji standard).
**Agregacija** — tehnologija za povezivanje više fizičkih mrežnih kartica u jedno logičko sučelje. Pogledajte **LACP** protokol..
**AHCI** — *Advanced Host Controller Interface*, standard koji definira mogućnosti rada uređaja spojenih preko **SATA** sučelja.
**ALSA** — *Advanced Linux Sound Architecture*, softverska arhitektura zvučnog podsustava Linuxa.
**ALU** — *Arithmetic and Logical Unit*, aritmetičko logička jedinica centralnog procesora (CPUa).
**AMD** — *Advanced Micro Devices*, tvrtka **AMD**.
**AMD-v** – Hardverska podrška za virtualizaciju ugrađena u procesore tvrtke **AMD**.
**ANSI** — *American National Standards Institute*, Američki institut za standarde.

**Anycast** — metoda komunikacije u kojoj do jedne odredišne (**IP**) adrese postoji više putanja odnosno puteva.
**AND** — obično se odnosi na logičku operaciju **I** (**Booleova algebra**).
**AP** — *Access Point*, pristupna točka, obično označava bežični uređaj za povezivanje na mrežu.
**API** — *Application Programming Interface*, programsko sučelje, obično između programa i sustava.
**APIC** — *Advanced Programmable Interrupt Controller*, sklop za rad sa signalima prekida (**IRQ**). Pogledajte i **LAPIC**.
**APIC** - *kao izvor vremena (brojač vremena).* Pogledajte: **ACPI PM**, **APIC**, **HPET**, **RTC**, **PIC** i **TSC**.
**apt** - *Advanced Package Tool,* upravitelj paketa više razine za Debian distribucije Linuxa.
**ARM** — *Advanced RISC Machines*, tvrtka ARM ili **ARM** arhitektura procesora (nekompatibilna sa: **x86**, **SPARC**, …).
**ARP** — *Address Resolution Protocol*, mrežni komunikacijski protokol.
**ARPA** — Advanced Research Projects Agency (pogledajte i **DARPA**), agencija za razvoj naprednih projekata.
**ARPANET** — *Advanced Research Projects Agency Network*, računalna mreža koju je razvila **ARPA** agencija (preteča Interneta).
**ASCII** — *American Standard Code for Information Interchange*, standard za kôdiranje i dekôdiranje alfabeta (slova,brojeva,znakova).
**ASIC** — *Application Specific Integrated Circuit*, posebno učinkovit elektronički sklop specijaliziran za posebnu namjenu.
**Asinkrono** — vrsta komunikacije u kojoj se podaci mogu slati izvan nîza, te različitom brzinom (pr. slanje/primanje).
**Asimetrično kriptiranje –** mehanizmi u kriptografiji koji koriste različite ključeve za kriptiranje (šifriranje) i dekriptiranje.
**ASLR** - *Address Space Layout Randomization,* mehanizam zaštite memorije.
**ASN.1** - *Abstract Syntax Notation One,* standard koji definira strukturu podataka i načine **serijalizacije**/**deserijalizacije**.
**AT** — *Advanced Technology,* napredna tehnologija, označava PC format računala, ali i format matičnih ploča računala.
**ATA** — *Advanced Technology Attachment*, napredna tehnologija za spajanje tvrdih diskova na računalo.
**ATM** — *Asynchronous Transfer Mode*, mrežna tehnologija za širokopojasni Internet (*razvijena 1980-ih*), radi na prva tri ***OSI*** sloja.
**Atomski sat** — najtočniji izvor točnog vremena, s maksimalnom pogreškom od jedne sekunde u 30 milijuna godina.
**Availability** – dostupnost sustava: količina vremena (u postotku) koju je uređaj (računalo) u stanju neprekidno raditi bez greške. Iskazuje se u odnosu na broj radnih sati u jednoj godini (24 x 365=8.760 sati), primjerice 99.99% (od navedenog broja sati).

# B

**Backup** — označava izradu sigurnosnih kopija podataka ili sigurnosnu kopiju.
**Bandwidth** — (**BW**) Širina komunikacijskog pojasa (pogledajte i **Throughput**).
**Bash** — *Bourne-again shell*, BASH ljuska (naredbeni redak).
**BASIC** — *Beginner's All-Purpose Symbolic Instruction Code*, početnički simbolički računalni programski jezik.
**BAR –** *Bese Address Register,* registri/adrese koje se koriste za komunikaciju između računala i uređaja na PCI/PCIe sabirnici.
**Baza podataka (DB)** — organizirani skup zbirki podataka, koje je na strukturiran način moguće pretraživati/zapisivati.
**BBU** — *Battery Backup Unit,* odnosi se na posebnu RAM memoriju s baterijom (obično se koristi za **RAID** kontrolere).
**BeeGFS** — *Bee File System,* klasterski dijeljeni datotečni sustav.
**BER** - *Basic Encoding Rules,* jedan od **ASN.1** načina kôdiranja podataka (pr. za pohranu strukture podataka u datoteku).
**Beta** — označava nedovršenu (testnu) inačicu programa ili sustava.
**BFD** - *Bidirectional Forwarding Detection,* protokol za brzu detekciju kvarova u vezama između mrežnih uređaja.
**BGP** — *Border Gateway Protocol*, napredni protokol za usmjeravanje (najčešće u primjeni od strane telekoma).
**Big-endian (BE)**— način pohrane podataka u memoriju za određene arhitekture procesora. Pogledajte **Little-endian**.
**Big TCP** — mehanizam mrežnog stoga Linuxa koji može drastično povećati **GSO** i **GRO** među memoriju i propusnost mreže.
**Binarno** – brojčani sustav koji koristi samo nule (0) i jedinice (1).
**BIND** — *Berkeley Internet Name Daemon,* **DNS** servis odnosno poslužiteljski softver (pogledajte **DNS**).
**BIOS** — *Basic Input Output System*, osnovni ulazno/izlazni sustav računala.
**bit** — binarna znamenka (nula [*0*] ili jedan [*1*]).
**Booleova algebra** — simbolički sustav matematičke logike kojim se prikazuju postupci deduktivnog zaključivanja.
**BOOTP** — *Bootstrap Protocol*, mrežni protokol (preteča **DHCP** protokola).
**Bond(ing)** — tehnologija za povezivanje više fizičkih mrežnih sučelja (kartica) u jedno logičko sučelje. Pogledajte **LACP**.
**Boot** — označava inicijalni proces pokretanja računala.
**bps** — *bits per second,* bitova u sekundi.
**BSD** — *Berkeley Software Distribution,* varijanta operativnog sustava **UNIX**.
**Bridge** — uređaj (preteča preklopnika) ili tehnologija za povezivanje više mrežnih sučelja na OSI sloju dva (**OSI 2**).
**Broadcast** — metoda komunikacije u kojoj se poruka emitira na sva računala/uređaje na (konkretnoj) lokalnoj mreži.
**Buffer** — predmemorija odnosno brza memorija sustava ili uređaja (pr. mrežne kartice, diska i slično).
**Bug** – označava grešku u programu ili sustavu.
**Bus** – označava sabirnicu; pr. na matičnoj ploči računala za ugradnju komponenti računala [*mrežna, grafička i druge kartice*].
**Bzip(2)** —format kompresije podataka (datoteka) ***BZIP2***.

# C

**CA** – *Certificate Authority,* certifikacijsko tijelo zaduženo za izdavanje, pohranu i potpisivanja certifikata za kriptografiju.
**Cache** – priručna odnosno brza predmemorija koja se koristi za pohranu (i kasnije dohvaćanje) podataka koji se često koriste.
**CAM** - *Content Addressable Memory*, najčešće je to **ARP** tablica u **SRAM** memoriji usmjerivača.
**CAMM(2)** - *Compression Attached Memory Modul*, vrsta novije RAM memorije. Koristi **LGA** konektore.
**CARP** – *Common Address Redundancy Protocol*, mrežni protokol za redundanciju na OSI sloju tri. Pogledajte i **VRRP**.
**CD** — *Compact Disc,* kompaktni optički disk.
**CD-ROM** — *CD Read-Only Memory*, CD disk (memorija) s koje se može sâmo čitati, ali ne i zapisivati.
**CEPH** — objektni sustav za pohranu podataka u klasteru; skalabilan, visoko dostupan i ekstremno proširiv.
**CEPH FS** — datotečni sustav za pohranu podataka u klasteru, koji se oslanja na **CEPH** sustav u pozadini.

**CERT** — *Computer Emergency Response Team*, obično označava organizaciju koja se bavi objavom sigurnosnih propusta i načina kako ih ispraviti.
**CET** — *Central European Time,* vremenska zona koja je jedan sat u plusu u odnosu na **UTC** (centralnu vremensku zonu).
**CFI** — *Control Flow Integrity,* mehanizam zaštite memorije.
**CFQ** - *Completely fair queueing*, mehanizam *task/process scedulera* za raspodjelu izvršavanja procesa. Pogl. **EEVDF**.
**CGNAT**— *Carrier-grade NAT*, vrsta NAT translacije IPv4 adresa koju koriste telekomi (ISP-ovi). Pogl. **NAT** i **ISP**.
**Cgroups** — *control groups,* mehanizam za izolaciju i ograničavanje resursa računala (Linux).
**Chip** — označava elektronički sklop koji se sastoji od međusobno povezanih poluvodičkih elemenata (pogledajte **IC**).
**Chipset** — označava poseban elektronički sklop (ili sklopove) na matičnoj ploči računala, koji su nužni za rad računala.
**Character set** — standardiziran skup znakova specifičan za određenu regiju ili govorno područje. Primjerice za **ASCII** znakove iznad 127 pozicije, nakon: slova, brojeva te standardnih i posebnih znakova, slijede regionalni znakovi (pr. ***ŠĐČĆŽ***).
**CIFS** — *Common Internet Filesystem*, mrežni dijeljeni datotečni sustav. Pogledajte i **NFS**.
**CISC** procesor – *Complex Instruction Set Computer*, računalna arhitektura kod koje se koristi što je moguće više naredbi na mikrorazini odnosno tzv. mikro instrukcija procesora.
**CLI** – *Command-line Interface*, tekstualno korisničko sučelje (naredbeni redak).
**Client** – računalo/uređaj ili program koji koristi ili traži pristup servisu (uslugu) od poslužitelja.
**Cluster** – označava skup istih elemenata; primjerice: skup računala koja se korisniku prikazuju kao jedan (zajednički) resurs.
Označava i skup fizičkih **sektora** na disku, koji čine najmanju jedinicu za zapisivanje podataka u datotečnom sustavu.
**CMOS** — *Complementary Metal-Oxide Semiconductor*, vrsta poluvodiča
**CN** — *Canonical Name, Common Name,* **DNS** unos koji pokazuje na drugu domenu ili se radi o polju unutar certifikata (**TLS**).
**CoDec** – *Coder/Decoder,* sustav koji odrađuje kodiranje/dekodiranje prema određenom algoritmu (pr. za video je to **HEVC**).
**Compression** — proces smanjivanja (sažimanja) podataka, kako bi zauzimali manje prostora za pohranu. Pogl. **Komprimiranje**.
**Context switch** — metoda (mehanizam) prebacivanja na obradu između više programa (procesa) unutar jezgre oper. sustava.
**cps** — *characters per second*, broj znakova u sekundi.
**CPU** — *Central Processing Unit,* centralni procesor.
**CPU Affinity** — mehanizam vezivanja točno definirane CPU jezgre za određeni proces (program). Pogl. **CPU Pinning**.
**CPU *Pinning*** — mehanizam vezivanja točno definirane CPU jezgre za određeno virtualno računalo. Pogl. **CPU Affinity**.
**CR** — *Carriage Return*, znak (oznaka) za novi redak teksta (ENTER).
**CRC** — *Cyclic Redundancy Check,* algoritam za izradu provjernog zbroja (*checksuma*).
**CRLF** — *Carriage Return Line Feed*, oznaka za novi redak teksta prema **Windows** načinu označavanja.
**CRT –** *Chinese Remainder Theorem,* Kineski teorem o ostacima je matematički teorem koji omogućuje dekriptiranje poruka.
**Cryptography** - *Kriptografija*, odnosi se na metodu šifriranja i slanja poruka koje može pročitati samo onaj kome su one i namijenjene. Postoje mnoge metode rada i algoritmi u kriptografiji.
**CSMA/CD** – *Carrier Sense Multiple Access with Collision Detection*, protokol za detekciju višestrukog pristupa mrežnom mediju s otkrivanjem kolizije. Ovaj protokol je važan u **Half Duplex** načinu rada te u mrežama koje su koristile **HUB** uređaje.
**CSS** — *Cascading Style Sheets*, koristi se za prezentaciju dokumenta u jezicima poput **HTML**.
**CSV** — *Comma-Separated Values*, tekstualni format datoteke koji koristi posebni znak (delimiter) za odvajanje stupaca teksta.
**CWDM –** *Coarse Wavelength Division Multiplexing,* tehnologija modulacije i multipleksiranja više optičkih signala kroz jedno optičko vlakno.
**CXL** — *Compute Express Link,* napredna sabirnica koja kao fizičko sučelje koristi **PCIe** sabirnicu za direktan pristup RAM memoriji među uređajima na **PCIe** sabirnici, bez posredovanja CPU-a (pr. *LAN RAM ←→ GPU RAM, DPU RAM ←→ GPU RAM*).

# D

**DARPA** — *Defense Advanced Research Projects Agency*, vojna agencija SAD-a za istraživanje i razvoj novih tehnologija.
**DAC** — *Digital to Analog Converter*, konverter (elektronički sklop) digitalnog u analogni signal.
**DAC** — *Direct Attach Cable*, pasivni ili aktivni kabel s **SFP** vrstom konektora za velike brzine prijenosa podataka (10+Gbps)
**DAT** — *Digital Audio Tape*, oblik (format) tračnog uređaja odnosno trake (vrpce).
**Datotečni sustav** – definira metode i strukturu podataka koju operativni sustav koristi za pohranu i dohvaćanje podataka (datoteka i direktorija [*mapa*]) s diskovnog medija (tvrdi disk, USB disk, SSD disk i drugo). Pogledajte **Filesystem**.
**Daemon** — poslužiteljski program koji radi u pozadini, kao servis.
**DB** — *Database*, baza podataka. Pogledajte **Baza podataka**.
**DCF77** — radijski (kratkovalni [77.5kHz]) izvor točnog vremena, spojen na atomski sat.
**DCL –** *Data Control Language,* logička kategorija SQL naredbi za upravljanje pristupom podacima.
**DDL -** *Data Definition Language,* logička kategorija SQL naredbi za definiranje strukture baze podataka.
**DDoS** — *Distributed Denial of Service*, distribuirani napad koji uzrokuje uskraćivanje mrežnih resursa (servisa).
**DDR** — *Double Data Rate*, vrsta RAM memorije kod koje se prenosi dvostruko više podataka u svakom ciklusu.
**Debugging** — postupak otkrivanja i ispravljanja grešaka (pr. u programskom kôdu).
**Debugfs** — pseudo datotečni sustav koji se montira u `/sys/kernel/debug`. Pogledajte: **sysfs, procfs, tmpfs** i **Filesystem**.
**Dekriptiranje** – kriptografska metoda kojom kriptirani tekst vraćamo u izvorni (čisti) nekriptirani (nešifrirani) oblik.
**Dekodiranje** – metoda prevođenja kodiranih podataka u nekodirani odnosno izvorni oblik. Pogledajte **(en)kôdiranje**.
**DEP -** *Data Execution Prevention,* mehanizam zaštite memorije.
**DES** — *Data Encryption Standard*, kriptografski algoritam za simetrično kriptiranje (šifriranje) i dekriptiranje podataka.
**DKMS** — *Dynamic Kernel Module Support*, mehanizam za automatsko kompiliranje i instalaciju kernel modula.
**DML –** *Data Manipulation Language*, logička kategorija SQL naredbi za manipuliranje podacima.
**Deserijalizacija** – proces prevođenja tôka podataka u objekte. Za suprotan proces, pogledajte **serijalizacija**.
**Destination Address** - adresa mrežnog uređaja koji prima podatke (odnosi se na adresu odredišnog uređaja).
**dev** — *device*, odnosi na neki fizički (hardverski) ili logički uređaj.
**Devtmpfs** — pseudo datotečni sustav koji se montira u `/dev/` direktorij (mapu). Pogledajte: **sysfs, procfs, tmpfs** i **Filesystem.**

**DER** - *Distinguished Encoding Rules,* jedan od **ASN.1** načina kôdiranja podataka (pr. za pohranu strukture podataka u datoteku).
**DFB** - *Distributed Feedback Laser*, vrsta lasera za modulaciju snopa svjetlosti kroz optičko vlakno.
**DH**— *Diffie–Hellman*, metoda (algoritam) za sigurnu razmjenu kriptografskih ključeva.
**DHCP** – *Dynamic Host Configuration Protocol*, mrežni protokol za dodjeljivanje IP adresa i mrežnih postavki.
**Diameter** - protokol koji osigurava pristup mrežnim resursima (u kategoriji **AAA** protokola, a nasljednik je **RADIUS** protokola**)**.
**DiffServ** – *Differentiated services,* arhitektura koja definira ***DSCP*** polje unutar zaglavlja IP paketa. Pogledajte **DSCP**.
**DIMM** — *Dual Inline Memory Module*, vrsta RAM memorije.
**Directory** — *folder,* [*mapa],* kataloška struktura unutar datotečnog sustava koja može sadržavati datoteke ili druge mape.
**DLL** — *Dynamic Link Library*, posebna (datoteka) koja je prema namjeni biblioteka (sadrži razne funkcije).
**DMA** – *Direct Memory Access*, komponenta sustava koja omogućuje direktan pristup glavnoj memoriji računala, bez posredovanja procesora (CPU).
**DMI** – *Direct Media Interface*, specijalizirana ekstremno brza sabirnica između procesora i čipseta.
**DMI** – *Desktop Management Interface*, sučelje za dohvaćanje informacija o računalu. Noviji naziv za ovo sučelje je **SMBIOS**.
**DML** - *Directly Modulated Laser*, vrsta lasera za modulaciju snopa svjetlosti kroz optičko vlakno.
**DMTF** – *Distributed Management Task Force*, organizacija za definiranje standarda (definirala je **SMBIOS**).
**DNS** – *Domain Name Server*, distribuirani hijerarhijski sustav poslužitelja s tablicama domenskih naziva.
**Docker** – jedna od tehnologija Linux kontejnera. Pogledajte **mikro kontejnere**, **LXC, Podman** i **OpenVZ**.
**Domain name** – odnosi se na ime domene (**DNS** sustava).
**DOS** – *Disk Operating System*, odnosi se na takozvani diskovni operativni sustav (tvrtke **Microsoft**).
**DoS** – *Denial of Service*, napad koji uzrokuje uskraćivanje mrežnih resursa (servisa).
**Download** — prebacivanje (kopiranje) podataka (datoteka) s udaljenog računala, na naše računalo.
**DPDK** - *Data Plane Development Kit*, skup programskih biblioteka (i sustav) zaduženih za brzo dohvaćanje mrežnih paketa.
**dpkg** - *Debian Package,* upravitelj paketa niže razine za Debian distribucije Linuxa.
**DPU** - *data processing unit*, specijalizirani sklop za visokoučinkovitu (paralelnu) obradu podataka.
**DRAM** – *Dynamic Random-access Memory,* vrsta radne (**RAM**) memorije.
**Driver** – upravljački program za određeni hardver (pr. mrežnu, zvučnu. grafičku ili neku drugu karticu/uređaj).
**DSA** - *Digital Signature Algorithm, kriptografski* algoritam za potpisivanje i provjeru vjerodostojnosti.
**DSCP** - *Differentiated Services Code Point,* dio zaglavlja IP paketa zadužen za kontrolu kvalitete usluge. Pogledajte **QoS**.
**DSL** — *Digital Subscriber Line,* mrežna tehnologija za pristup internetu (od telekoma do domaćinstva).
**DTE** — *Data Terminal Equipment*, terminalna oprema (obično se odnosi na opremu telekoma [pr. usmjerivač]).
**DTR** — *Data Terminal Ready*, u RS-232 serijskoj komunikaciji, signal da je terminal spreman za rad.
**Duplex** — metoda komunikacije kroz jedan komunikacijski kanal: ili jednosmjerno (**Half**) ili dvosmjerno (**Full**).
**DVD** — *Digital Video Disc,* digitalni video disk ili format.
**DVD-ROM** — *DVD-Read Only Memory,* DVD disk (memorija) samo za čitanje, ali ne i zapisivanje.
**DWDM** – *Dense Wavelength Division Multiplexing*, tehnologija modulacije i multipleksiranja više optičkih signala kroz jedno optičko vlakno.

# E

**E-mail** — elektronička pošta.
**EAP** — *Extensible Authentication Protocol*, mrežni protokol za autentikaciju korisnika.
**EBCDIC** — *Extended Binary Coded Decimal Interchange Code*, način kôdiranja i dekodiranja alfabeta. Pogledajte i **ASCII**.
**ECC** — *Elliptic Curve Cryptography* ili *Error Correction Code*, kriptografski standard ili kôd za korekciju greške (*ECC*).
**ECDSA** – *Elliptic Curve Digital Signature Algorithm, kriptografski* algoritam za potpisivanje i provjeru vjerodostojnosti.
**Ed25519** – k*riptografski* algoritam za potpisivanje i provjeru vjerodostojnosti koji koristi **SHA-512** i **Curve25519**.
**EEPROM** — *Electronically-Eraseable Programmable Read-Only Memory*, programibilna memorija koja se koristi za čitanje.
**EEVDF** - *Earliest Eligible Virtual Deadline First*, mehanizam *task/process scedulera* za raspodjelu izvršavanja procesa. Pogledajte i **CFQ** i ***Task/process scheduler***.
**EFI** — *Extensible Firmware Interface*, sučelje između operativnog sustava i **Firmware**-a (pogledajte i **UEFI**).
**EGP** — *Exterior Gateway Protocol*, mrežni protokol za usmjeravanje, za usmjerivače.
**EIDE** — *Enhanced IDE*, sučelje za spajanje diskovnih uređaja. Pogledajte i druga sučelja: **IDE**, **SATA**, **SAS** i **SCSI**.
**EIGRP** — *Enhanced Interior Gateway Routing Protocol*, mrežni protokol za usmjeravanje, za usmjerivače.
**EISA** — *Extended Industry Standard Architecture,* (starija) sabirnica računala.
**Ekstenzija –** datotečni nastavak, produžetak, sufiks, ekstenzija, drugi dio imena datoteke koji označava format datoteke.
**ELF** — *Executable and Linkable Format*, binarni format za izvršne Linux datoteke.
**Emulacija** — odnosi se na imitiranje rada određenog hardvera i svôg njegovog pripadajućeg sklopovlja. Koristi se primjerice u virtualizaciji. Jednostavnije imitiranje sâmo dijela određenih funkcija hardvera s mnoštvom ograničenja je **Simulacija**.
**Embedded system** — sustav ili čak cjelokupno mikro računalo dizajnirano za specifičnu namjenu, s ograničenim hardverom.
**EML** - *Externally Modulated Laser*, vrsta lasera za modulaciju snopa svjetlosti kroz optičko vlakno.
**Enkapsulacija** — metoda ugnježđivanja podataka. U mrežama se odnosi na ugnježđivanje podataka iz jednog sloja mreže, u polja koja se nalaze na drugom sloju mreže. Primjerice aplikacijski podaci se ugnježđuju u transportni sloj koji dodaje svoje zaglavlje. Potom se to sve ugnježđuje u niži sloj (IP), koji dodaje svoje zaglavlje i šalje na još niži sloj i tako dalje.
**Enkripcija** — (kriptografska) metoda s kojom izvorni (čisti) tekst prebacujemo u kriptirani (šifrirani) ili posebno kôdirani oblik.
**(En)kodiranje** — metoda za pretvorbu podataka u kôdirani oblik, ovisno o shemi kodiranja; primjerice: ***ASCII, UTF-8, …*** .
**EOF** — *End of File*, označava kraj datoteke.
**EOL** — *End of Life* ili *End of Line*, označava kraj života (podrške) za uređaj ili kraj linije (reda) teksta.
**EOM** — *End of Message*, kraj poruke.
**EPROM** — *Eraseable Programmable Read-Only Memory*, programibilna memorija koja se primarno koristi za čitanje.
**EtherChannel** — tehnologija (*Cisco*) povezivanja više fizičkih mrežnih kartica u jednu logičku. Pogledajte **Agregacija**.

**Ethernet** — najčešće korištena tehnologija računalnih mreža, najčešće u primjeni u lokalnim (**LAN**) mrežama.
**EUI-48** — opći standard za definiranje 48 bitnih **MAC** adresa (softverskih i hardverskih). Pogledajte i **MAC-48**.
**EUI-64** — opći standard za definiranje 64 bitnih **MAC** adresa (softverskih i hardverskih). Pogledajte **MAC-48**.
**EULA** — *End User License Agreement*, pravni (obvezujući) ugovor s krajnjim korisnikom.
**EXT** — *extended file system (**ext2**, **ext3** ili **ext4**)*, datotečni sustav koji koristi Linux. Pogledajte **Datotečni sustav**.

# F

**F2FS** — *Flash-Friendly File-System*, datotečni sustav prilagođen **SSD** i **NVMe** diskovima. Pogledajte i **SSDFS**.
**FAQ** — *Frequently Asked Questions*, često postavljana pitanja.
**FAT** — *File Allocation Table*, tablica datotečnog sustava i/ili datotečni sustav (***FAT12***/***FAT16***/***FAT32***).
**FBDIMM** - *Fully Buffered Memory,* vrsta **SDRAM** memorije koja ima ugrađen međuspremnik (sklop/čip) prema kontrolnim i podatkovnim linijama na samoj memorijskoj pločici.
**FC** – *Fibre Channel*, tehnologija za brzi prijenos blokova podataka, za uređaje za pohranu podataka (u **SAN** sustavima).
**FCoE** - *Fibre Channel over Ethernet,* tehnologija koja ugnježđuje ***Fibre Channel (FC)*** protokol u ***IP (TCP)*** mrežne okvire.
**FCP - Fibre Channel protocol,** Fibre Channel protokol.**.**
**FDD** — *Floppy Disk Drive*, meki disk dimenzija 3.5 inča ili 5.25 inča.
**FDDI** — *Fiber Distributed Data Interface*, standard za prijenos podataka u **LAN** mrežama, preko optičkih vlakana.
**FEC** - *Forward error correction,* metoda detekcije i korekcije grešaka u slanju podataka računalnim mrežama.
**FHS** — *Filesystem Hierarchy Standard*, standard za hijerarhijsku strukturu direktorija (mapa) na datotečnom sustavu.
**Fibre Channel (FC)** – Pogledajte **FC i FCoE**.
**FIFO** — *First In First Out*, sustav slanja prema principu: prvi koji ulazi, prvi i izlazi.
**File** — datoteka, resurs unutar datotečnog sustav u koji se mogu snimati ili iz kojeg se mogu čitati podaci.
**File descriptor** — *opisnik datoteke*, posebna datoteka u UNIX ili Linux datotečnim sustavima.
**File system** — datotečni sustav. Pogledajte **FS**.
**Firewall** — predstavlja uređaj ili tehnologiju vatrozida (onaj koji ograničava pristup mrežnim resursima).
**Firmware** — poseban softver (u **ROM/EPROM/Flash** memoriji) koji omogućuje pristup hardveru, na najnižoj razini.
**FLAC** — *Free Free Lossless Audio Codec*, format za digitalni zvuk koji koristi oblik kompresije bez gubitka podataka.
**Format** — odnosi se na formatiranje particije diska odnosno instalaciju datotečnog sustava ili primjerice format datoteka.
**FOSS** — *Free and Open Source Software*, slobodni i softver otvorenog programskog kôda.
**FPGA** — *Field Programmable Gate Array*, programabilni visoko učinkoviti elektronički sklop.
**FPU** — *Floating Point Unit*, koprocesor za rad s pomičnim zarezom (s decimalnim brojevima).
**Fragmentacija** – podložnost datotečnog sustava neučinkovitoj uporabi podatkovnog prostora (na particiji diska).
**FQDN** - *Fully Qualified Domain Name*, puno ime računala koje uključuje i pripadajuću domenu.
**FS** — *File System*, datotečni sustav (pr. **NTFS**, **FAT, exFAT**, **XFS**, **ZFS**, **ext 2/3/4**, …). Pogledajte **Datotečni sustav**.
**FSB** — Front Side Bus, sabirnica od procesora prema *chipsetu*.
**FSF** — *Free Software Foundation*, fondacija koja pruža podršku za razvoj programa i sustava otvorenog programskog kôda.
**FTP** – *File Transfer Protocol*, internetski protokol za prijenos datoteka.
**FTTB** - *Fiber to the Building,* **FTTH** tehnologija za optička vlakna (razvod do zgrade).
**FTTC** - *Fiber to the Curb,* **FTTH** tehnologija za optička vlakna – razvod do rubova ulica ili blizu korisničkog područja.
**FTTH** - *Fiber to the Home*, tehnologija za kućanstva u kojoj se koristi optičko vlakno za pristup internetu.
**FTTP** - *Fiber to the Premises,* **FTTH** tehnologija za optička vlakna (razvod do stanova/kućanstava).
**Full Duplex** — istovremena dvosmjerna (*upload+download*) komunikacija kroz jedan komunikacijski kanal.

# G

**Gateway** — odnosi se na uređaj kojem je namjena davanje pristupa uređajima s lokalne mreže prema internetu.
**GARP** — *Generic Attribute Registration Protocol,* stariji protokol za registriranje pripadnosti VLAN-ovima (pogl. **MRP**).
**Gb** — *Gigabit*, označava milijardu bitova (1.000.000.000 bitova)
**Gbps** — *Gigabit per second, Gigabita u sekundi (milijarda bitova u sekundi) [jedinica za propusnost mreže].*
**GB** — *Gigabyte*, gigabajt, označava milijardu bajta (1.000.000.000 bajta) [*1bajt=8 bitova*].
**GCC** — *GNU Compiler Collection* (nekad se zvao *GNU C Compiler*), **GNU** kompilator.
**Geneve** - *Generic Network Virtualization Encapsulation,* protokol koji objedinjuje značajke **VxLAN** i **NVGRE**.
**GIF** — *Graphics Interchange Format*, format rasterske slike (datoteke).
**GID** — *Group ID*, identifikacijski broj korisničke grupe na UNIX/Linux sustavu.
**Git –** sustav za verzioniranje koda. Pogledajte **SVN** i **Mercurial**.
**GFS2**— *Global Filesystem v.2.,* klasterski dijeljeni datotečni sustav.
**GLBP** — *Gateway Load Balancing Protocol, Cisco* protokol za redundanciju (na OSI3) s mogućnošću *Load Balancing-a.*
**GlusterFS**— *Gluster Filesystem,* klasterski dijeljeni datotečni sustav.
**GNU** — *GNU's Not Unix*, paradigma koja označava GNU projekt koji obuhvaća tisuće programa otvorenog kôda.
**GMT**— *Greenwich Mean Time,* početna vremenska zona.
**GMRP** - *GARP Multicast Registration Protocol,* stariji protokol za registriranje **multicasta**. Pogledajte **GARP**.
**GPG** — *GNU Privacy Guard*, softver otvorenog programskog kôda za kriptografiju.
**GPL** — *General Public License*, opća javna licenca otvorenog kôda.
**GPON -** *Gigabit Passive Optical Network,* FTTH tehnologija za kućanstva**.** Pogledajte **FTTH.**
**GPS** – *Global Positioning System,* globalni pozicijski sustav (sustav za pozicioniranje).
**GPT** — *GUID Partition Table,* vrsta particijske tablice i sâmim time particioniranja koja podržava diskove veće od **2TB**.
**GPU** — *Graphics Processing Unit*, grafički procesor.

**GRE -** *Generic Routing Encapsulation*, protokol za tuneliranje koji ugnježđuje protokole na OSI sloju dva, preko IP mreže.
**GRO** — *Generic receive offload*, implementacija **LRO** mehanizma u softveru za povećanje dolazne mrežne propusnosti.
**GRUB** — *Grand Unified Boot-Loader*, tzv. *boot loader (zadužen za inicijalnu fazu učitavanja operativnog sustava)*.
**GSM** — *Global System for Mobile Communications*, telekomunikacijski standard za mobilnu telefoniju.
**GSO -** *Generic segmentation offload*, mehanizam za povećanje odlazne mrežne propusnosti. Naziva se i **LSO** ili **TSO**.
**GUI** — *Graphical User Interface*, grafičko sučelje.
**GUID** — *Globally Unique IDentifier*, globalni identifikacijski broj (oznaka).
**GVRP** — *GARP VLAN Registration Protocol*, stariji protokol za registriranje **VLAN** trunk veza. Pogledajte **GARP**.
**Gzip** — *GNU ZIP*, format kompresije podataka (datoteka).

## H

**HAL** — *Hardware Abstraction Layer*, hardverski apstrakcijski sloj.
**Half Duplex** — jednosmjerna komunikacija kroz jedan komunikacijski kanal (ili *upload* ili *download*)).
**Hard Link -** *tvrda veza*, je unos u datotečnom sustavu koji povezuje ime sa sadržajem datoteke u datotečnom sustavu.
**Hardware** — odnosi se na sklopovlje računala odnosno fizičke dijelove računala (pr. **CPU**, **RAM**, disk, mrežna kartica ,…).
**Hash** — jednosmjeran proces pretvaranja ulaznih podataka bilo koje veličine u provjerni broj fiksne veličine (pr. **MD5** ).
**HBA** — *Host Bus Adapter*, odnosi se na diskovni kontroler ili posebno sučelje; primjerice *Fibre Channel* (**FC**).
**HD** — *High Density*, visoka gustoća (zapisa).
**HDD** — *Hard Disk Drive*, diskovni pogon (disk).
**HEVC** — *High Efficiency Video Coding*, algoritam za kodiranje i dekodiranje (**CoDec**) video zapisa.
**Hypervisor –** *Hipervizor*, upravitelj (*virtualizator*) i menadžer za virtualna računala. Pogledajte **KVM** i **QEMU**.
**Horizontalno kabliranje** – dio strukturnog kabliranja koji se odnosi na razvlačenje kabela unutar etaže stambenog objekta.
**Host** – označava računalo ili uređaj na mreži.
**Hostname** – odnosi se na ime računala ili uređaja na mreži.
**HP** — Hewlett-Packard (tvrtka).
**HPC** — *High-Performance Computing*, računalstvo visokih performansi.
**HPET** *High Precision Event Timer*, izvor vremena (brojač vremena). Pogledajte: **ACPI PM**, **APIC**, **RTC**, **PIC** i **TSC**.
**HT** — *Hyper Threading*, **Intelova** tehnologija za poboljšanje performansi rada superskalarnih procesora. Pogledajte **SMT**.
**HTML** — *Hypertext Markup Language*, **HTML** jezik.
**HTTP** — *Hypertext Transport Protocol*, **HTTP** aplikacijski transportni protokol.
**HTTPS** — *Hypertext Transport Protocol Secure*, **HTTP** protokol zaštićen sa **SSL** ili **TLS** protokolima.
**Hugetlbfs** — pseudo datotečni sustav koji se montira u `/dev/hugepages`. Pogledajte: **sysfs, procfs, tmpfs** i **Filesystem**
**Hz** — Hertz, mjerna jedinica za frekvenciju u Međunarodnom sustavu (u **SI** sustavu).

## I

**I2C ili IIC**— *Inter-Integrated Circuit*, sabirnica za jednostavnu komunikaciju (uobičajeno sa senzorima). Pogl. **SMBus**.
**I/O -** *Input/Output*, ulazno/izlazni sustav računala.
**IANA** — *Internet Assigned Numbers Authority*, autoritet (organizacija) za rezervaciju/kupnju IP adresa na internetu.
**IBM** — *International Business Machines* (tvrtka).
**IC** — *Integrated Circuit*, integrirani sklop (*krug*) koji upravlja elektroničkim signalima, sastoji se od međusobno povezanih poluvodičkih elemenata (pr. tranzistori, diode, otpornici, kondenzatori, …).
**ICANN** — *Internet Corporation for Assigned Names and Numbers*, autoritet (organizacija) za rezervaciju/kupnju IP adresa.
**ICMP** − *Internet Control Message Protocol*, komunikacijski mrežni protokol.
**ICT** — *Information and Communication Technology*, odnosi se na informacijske i komunikacijske tehnologije.
**IDE** — *Integrated Development Environment*, softver (aplikacija) namijenjena za programiranje (razvoj programa).
**IDE** — *Integrated Drive Electronics*, sučelje i tehnologija za spajanje (**ATA**) diskova na računalo.
**IDS** – *Intrusion Detection Systems*, sigurnosni (pasivni) sustav za analizu napada IT sustava. Pogledajte **IPS**.
**IEEE** — *Institute of Electrical and Electronics Engineers*, standardizacijsko tijelo koje donosi standarde u elektrotehnici.
**IETF** — *Internet Engineering Task Force*, standardizacijsko tijelo koje donosi standarde vezane za Internet tehnologije.
**IGMP** — *Internet Group Management Protocol*, mrežni protokol za *multicast* komunikaciju.
**IGRP** — *Interior Gateway Routing Protocol*, mrežni protokol za usmjeravanje, za usmjerivače.
**IMAP** — *Internet Message Access Protocol*, protokol za elektroničku poštu.
**InfiniBand** – mrežni komunikacijski standard u mrežama za povezivanje *superračunala (konkurencija **Ethernetu**).*
**Integrated Circuits -** integrirani krugovi odnosno elektronički sklopovi.
**Interface** – sučelje (međusklop) računala ili logičke/fizičke: mrežne ili neke druge kartice u računalu i slično.
**Internet** – najveća globalna računalna mreža na svijetu.
**Intel** – tvrtka poznata po proizvodnji raznih kategorija poluvodiča (pr. procesora, mrežnih kartica i slično).
**IP adresa -** 32-bitna adresa koja se dodjeljuje uređajima na mreži, kao njihov identifikator (iz skupa **TCP/IP** protokola).
**IPAM –** *IP Address Management*, sustav za planiranje IP mreže i mrežnih resursa (IP, VLAN, podmreže, uređaji na mreži,…)
**IP** – *Internet protocol*, Internet protokol (protokol iz skupa **TCP/IP** protokola) [*komunikacijski protokol*]. Pogledajte **Protokol**.
**IP fragmentacija** - mehanizam ugrađen u IP protokol, koji omogućava razlamanje mrežnih paketa u manje pakete.
**IPv4** – Internet protokol inačice (verzije) četiri, najrašireniji u lokalnim (**LAN**) mrežama.
**IPv6** – Internet protokol inačice (verzije) šest, najrašireniji u globalnim (**WAN**) mrežama.
**IPC** — *Inter-Process Communication* ili *Instructions per clock*, komunikacija između procesa ili broj instrukcija po taktu.
**IPI** - *Inter Processor Interrupt*, poseban signal prekida koji jedan fizički procesor (CPU) može poslati drugom (NUMA).
**IPMI -** *Intelligent Platform Management Interface*, sučelje za udaljeni pristup hardveru računala.

**IPS** – *Intrusion Prevention Systems*, sigurnosni (aktivni) sustav za zaštitu IT sustava od napada.
**IPsec** - *Internet Protocol Security*, standard za enkripciju i dekripciju podataka te upotrebu za **VPN** veze.
**IPTV** – *Internet Protocol Television*, televizijski servis preko IP protokola (TCP/IP).
**IPX/SPX** — *Internetwork Packet Exchange/Sequenced Packet Exchange*, mrežni komunikacijski protokol (preteča **TCP/IP**).
**IRC** — *Internet Relay Chat*, mrežni protokol za komunikaciju između korisnika.
**IRQ** — *Interrupt Request*, hardverski signal prekida (obično kreiran od strane uređaja prema procesoru).
**IS** — *Information Systems*, informacijski (informatički) sustav.
**ISA** — *Industry Standard Architecture* ili *Instruction set architecture:* oznaka za sabirnicu ili arhitekturu/dizajn računala.
**ISDN** — *Integrated Services Digital Network:* mrežna tehnologija za prijenos podataka preko **PSTN** (u upotrebi od 80-ih do 90-ih).
**iSCSI** - *Internet Small Computer Systems Interface*, tehnologija koja ugnježđuje SCSI protokol u **IP** (**TCP**) mrežne okvire.
**iSNS** -*Internet Storage Name Service*, imenični servis za **iSCSI** i **Fibre Channel** protokole.
**ISO** — *International Organization for Standardization*, organizacija za standarde ili format datotečnog sustava (pr. *ISO 9660*).
**ISP** — *Internet service provider* – tvrtka (telekom) koji nam daje pristup Internetu. Pogledajte **CGNAT**.
**IT** — *Information Technology*, informacijska tehnologija.
**ITU** — *International Telecommunication Union*, organizacija pri UN-u zadužena za područje informacijskih i telekomunikacijskih tehnologija te standardizaciju.
**IQN** - *iSCSI Qualified Name*, jedinstveno ime za iSCSI ciljeve i inicijatore.

# J

**Java** – objektno orijentirani računalni programski jezik.
**JBOD** — *Just a Bunch of Disks*, slobodni/dostupni diskovi, kako ih obično vidi **RAID** kontroler.
**JIT** — *Just-In-Time*, u trenutku, *JIT compiler* se koristi za dinamičko kompiliranje i izvršavanje u letu (u datom trenutku).
**Jitter** — u mrežama se odnosi na devijaciju (odstupanje) od prosječnog kašnjenja (latencije) signala.
**JPEG** — *Joint Photographic Experts Group*, ekspertna grupa zadužena za grafički format i ime formata datoteka .
**JRE** — *Java Runtime Environment*, Java virtualna mašina koja omogućuje izvršavanje (pokretanje) Java programa.
**JS** — *JavaScript* programski jezik.
**JSON** — *JavaScript Object Notation*, format objekata za razmjenu preko mreže.
**JSP** — *JavaServer Pages*, skup tehnologija koji omogućuje izradu dinamičkih web stranica.
**JTAG** — *Joint Test Action Group* ili *JTAG connector*, ind. standard za provjeru dizajna i testiranje štampanih pločica ili konektor za spajanje na štampanu ploču (uređaj) radi debugiranja i sličnih potreba.
**Jumpstart** — metoda automatizirane mrežne instalacije operativnog sustava **Solaris**. Pogledajte **Kickstart** i **PXE**.
**JVM** — *Java Virtual Machine*, JAVA virtualna mašina koja omogućava računalu izvršavanje **JAVA** programskog kôda.

# K

**K&R** — *Kernighan and Ritchie*, odnosi se na autore **C** (i drugih) programskih jezika: *Brian Kernighan* i *Dennis Ritchie.*
**K8s** — *Kubernetes*, sustava za automatiziranje, postavljenje, pokretanje, skaliranje i upravljanje Linux kontejnerima.
**KB** — **Keyboard**, tipkovnica.
**Kb** — Kilo bit (*tisuću bitova*).
**KB** — *Kilobyte*, kilobajt ili *Knowledge Base* - baza znanja.
***KDE*** — *K Desktop Environment*, grafičko okruženje Unix/Linux sustava.
**Keepalive** (*poruke*) – poruke s kojima se provjerava dostupnost druge strane u komunikaciji.
**Kernel** — odnosi se na jezgru svakog operativnog sustava, koja sadrži osnovne mehanizme potrebne za njegov rad.
**Kickstart** — metoda automatizirane instalacije i cjelokupne konfiguracije (*RedHat*) Linuxa. Pogledajte i **Jumpstart.**
**kHz** — *Kilohertz (tisuću hertza).*
**KISS** — *Keep It Simple, Stupid*, princip dizajniranja programa ili sustava: što čišći i jednostavno funkcionalan.
**KMI** - *Kernel Module Interface* - sučelje između kernel modula i sustava. Pogledajte i **ABI.**
**Kodiranje** — metoda za pretvorbu podataka u kôdirani oblik, ovisno o shemi kodiranja; primjerice: ***ASCII, UTF-8, …*** .
**Kompresija, Komprimiranje** — proces smanjivanja (sažimanja) podataka, kako bi zauzimali manje prostora za pohranu.
**Kriptografija** – znanstvena disciplina o metodama sa sigurno i pouzdano slanje podataka preko nesigurnih medija.
**Kubernetes (K8s)** – orkestrator za menadžment linux kontejnera [pr. **Docker** ili **Podman**]. Pogledajte **LXC, Docker** i **Podman**.
**KVM** — *Keyboard, Video, Mouse*, označava tipkovnicu, video (monitor) i miš.
**KVM** — *Kernel-based Virtual Machine*, označava komponentu Linuxa zaduženu za virtualizaciju i/ili sâmu virtualizaciju odnosno *hipervizor* za virtualizaciju (otvorenog kôda). Pogledajte **Hypervisor** i **QEMU**.

## L

**L1/L2/L3 cache** – priručne ekstremno brze RAM memorije (obično **SRAM** vrste) ugrađene unutar procesora (**CPU**-a).
**LAA -** *Locally administered addresses*, poseban opseg **MAC** adresa koji definira njenu upotrebu. Pogledajte **MAC**.
**LACP -** *Link Aggregation Control Protocol*, mrežni komunikacijski protokol za redundanciju i povećanje propusnosti.
**LAN -** *Local-Area Network*, lokalna mreža koja pokriva usko zemljopisno područje: unutar tvrtke, zgrade i slično (do nekoliko stotina metara).
**LAPIC –** *Local Advanced Programmable Interrupt Controller* komponenta integrirana u procesor (**CPU**). Pogledajte **APIC**.
**Latencija** – kašnjenje u komunikaciji (između dvaju točaka koje međusobno komuniciraju).
**Layer** – označava sloj, primjerice OSI sloj mreže ili TCP/IP sloj mreže i slično. Pogledajte **OSI** i **TCP/IP**.
**LBA** — *Logical Block Addressing*, način adresiranja blokova (za diskove).
**LCD** – *Liquid Crystal Display*, zaslon s tekućim kristalima.
**LDAP -** *Lightweight Directory Access Protocol*, protokol koji nam daje pristup **X.500** imeničnom servisu.
**LED** – *Light-emitting Diode*, svjetleća dioda.
**LF** — *Line Feed*, oznaka za kraj reda (retka teksta) prema Linux metodi označavanja.
**LFS** — *Linux From Scratch*, označava pojam „*Linux od početka*".
**LGA –** *Land grid array,* fizičko sučelje za ugradnju procesora, **CAMM2** i **LPCAMM2** RAM memorije.
**LGPL** — *[GNU] Lesser General Public License*, posebna licenca otvorenog kôda.
**LIFO** — *Last In First Out*, sustav slanja prema principu: zadnji koji ulazi, prvi i izlazi (suprotno od **FIFO**).
**LILO** — *Linux Loader*, jednostavni Linux *boot loader (koji se u potpunosti instalira u **MBR** zapis).*
**LIO -** *Linux-IO,* Linux servis koji je iSCSI odredište (target) (za **iSCSI SAN** sustav).
**Linux kontejner** — *Linux container,* tehnologija virtualizacije na razini operativnog sustava. Pogledajte **Docker** i **LXC**.
**Little-endian (LE)**— način pohrane podataka u memoriju za određene arhitekture procesora. Pogledajte **Big-endian**.
**LKML** — *Linux Kernel Mailing List*, mailing lista za Linux kernel.
**LLDP** — *Link Layer Discovery Protocol*[i] *[**802.1AB**]*, mrežni protokol za otkrivanje susjednih uređaja na mreži.
**Load Balancing** - tehnologija za distribuiranje i raspodjeljivanje mrežnog prometa prema odredišnim (*unutarnjim*) poslužiteljima.
**Login** — označava proceduru prijave korisnika na sustav, uređaj ili neki (mrežni) resurs.
**LPCAMM2** — *Low-Power CAMM(2)*, vrsta DDR2 RAM memorije koja radi na vrlo niskom naponu, Pogl. **CAMM2**.
**LPDDR** — *Low-Power Double Data Rate*, vrsta DDR 2 RAM memorije koja radi na vrlo niskom naponu.
**LPI** — *Linux Professional Institute*, institut za školovanje i certificiranje, iz područja Linuxa i Unixa.
**LRDIMM** – *Load Reduced DIMM,* vrsta SDRAM memorije koja ima ugrađen međuspremnik (čip) prema kontrolnim i podatkovnim linijama (na samoj memorijskoj pločici), koji prema memorijskim čipovima ima paralelni pristup.
**LRO** — *Large receive offload*, mehanizam za povećanje dolazne mrežne propusnosti.
**LSB** — *Least Significant Bit*, najmanje važan bît (podataka).
**LSB** — *Linux Standard Base*, zajednički projekt više distribucija Linuxa za standardizaciju strukture Linux sustava (pr. **FHS**).
**LSO –** *Large Send offload*, mehanizam za povećanje odlazne mrežne propusnosti. Naziva se i **GSO** ili **TSO**.
**LSI** — *Large-Scale Integration*, visoka razina integracije (obično se odnosi na integrirane sklopove).
**LTO** – *Linear Tape Open* – standard odnosno format tračnih uređaja i sâmih traka (vrpci) za arhiviranje podataka.
**LTR** — *Left-to-Right,* lijevo na desno.
**LUG** — *Linux User Group*, korisnička Linux grupa.
**LUN** — *Logical Unit Number*, broj koji identificira logički disk, obično **SCSI** uređaja. Pogledajte **SCSI**.
**Lustre** — klasterski dijeljeni datotečni sustav.
**LV** — *Logical Volume*, logički volumen (primjerice za **LVM2** sustav).
**LVM** — *Logical Volume Management*, menadžment logičkog volumena (tzv. softverskog RAID kontrolera).
**LZW** — *Lempel-Ziv-Welch*, standard za komprimiranje odnosno algoritam za komprimiranje podataka.
**LXC** — *Linux Containers*, jedna od tehnologija Linux kontejnera. Pogledajte **Docker, Podman i OpenVZ**.
**LXD** – orkestrator odnosno menadžment za **LXC** linux kontejnere. Pogledajte **LXC** i **Docker**.

## M

**M.2** — sučelje za spajanje **SSD** i **NVMe** diskova. Ono je praktično veza na **PCI express** (**PCIe**) sabirnicu i koristi **NVM Express** (**NVMe**) kao sučelje. Nudi ekstremno velike brzine prijenosa podataka, koje ovise samo o brzini **PCIe** sabirnice.
**MAC** — *Media Access Control*, jedinstvena adresa (mrežnog sučelja) na OSI sloju dva (OSI 2) *ethernet* mreža. Pogl. **EUI-48**.
**MAC-48 -** standard za definiranje 48 bitnih **MAC** adresa od strane proizvođača hardvera. Pogledajte **MAC** i **EUI-48**.
**Maska mreže** — *netmask*: parametar uz IP adresu, koji definira mrežu i/ili pripadnost pod mrežama (tzv. *subnetima*).
**Mail server** — označava poslužitelj za elektroničku poštu.
**MAN** — *Metropolitan Area Network*, mreža koja pokriva šire zemljopisno područje: unutar grada (do nekoliko desetaka kilometara).
**MCA –** *Machine Check Architecture,* arhitektura unutar procesora koja daje mehanizme za prijavu hardverskih grešaka.
**MCE -** *machine check exception,* mehanizmi unutar MCA arhitekture za dohvaćanje poruka o hardverskim greškama.
**MCF -** *Multi-core Fiber*, vrsta optičkog vlakna koje ima više jezgara unutar jednog vlakna.
**Mb** — *Megabit, Mega bit (milijun bitova).*
**Mbps** — *Megabit per second, Megabita u sekundi (milijuna bitova u sekundi).* Jedinica za propusnost mreže.
**MB** — *Megabyte*, megabajt.
**MBR** – *Master Boot Record*, glavni (početni) zapis na prvom sektoru svakog diska. Označava i shemu adresiranja diskova.
**MD5 -** *Message Digest 5,* kriptografski algoritam za izračun provjernog zbroja [*hash*]. Pogledajte i **SHA**.
**MDA** — *Mail Delivery Agent*, agent za dostavu elektroničke pošte.
**Memory Page** — označava tzv. stranicu memorije, koja je najmanja jedinica memorije u sustavu virtualne memorije.
**Mercurial –** sustav za verzioniranje koda. Pogledajte **Git** i **SVN**.
**MHz** — *Megahertz, Mega hertz (milijun hertza) [Hz=jedinica za frekvenciju].*

**MIDI** — *Musical Instrument Digital Interface*, sučelje za povezivanje glazbenih instrumenata.
**Mikro kontejner** — *Micro container***,** tehnologija apstrakcije Linux kontejnera. Pogledajte **Docker**.
**MIME** — *Multipurpose Internet Mail Extensions,* standard koji definira format sadržaja: datoteka ili privitaka unutar elektroničke pošte.
**Mini PCI** — mini **PCI** sučelje/sabirnica (koristilo se na prijenosnim uređajima zbog manjih dimenzija od **PCI** utora).
**Mini PCI express** (**Mini PCIe**) — mini **PCI express** sučelje (koristilo se na prijenosnim uređajima zbog manjih dimenzija).
**Mini SATA** (**mSATA**) – mini **SATA** sučelje za spajanje diskova (*koristilo se na prijenosnim uređajima zbog manjih dimenzija).*
**MIMD** - *multiple instruction, multiple data,* arhitektura je zadužena za paralelizam u izvršavanju instrukcija.
**MIPS** — *Million Instructions Per Second,* milijun instrukcija u sekundi (indirektna mjera sposobnosti obrade podataka).
**MIPS** — arhitektura procesora (nekompatibilna s: **x86**, **ARM**, **SPARC**, **RISC-V**, …).
**MIT** — *Massachusetts Institute of Technology,* istraživačko sveučilište u američkom gradu *Cambridge*, država *Massachusetts.*
**MLC** - *Multi Level Cell,* vrsta *Flash* memorije, koja može pohranjivati dva bita po memorijskoj ćeliji.
**MMIO** — *Memory-Mapped I/O,* memorijski povezan (mapiran) ulazno/izlazni sustav.
**MM** - *Multimode,* odnosi se na vrstu optičkog vlakna za prijenos svjetlosnih signala.
**MMU** — *Memory Management Unit,* dio memorijskog (RAM) kontrolera, zadužen za translaciju adresa: virtualne-fizičke.
**MMRP** — *Multiple MAC Registration Protocol,* protokol za registriranje **multicasta,** zamjena za **GMRP**. Pogledajte **MRP**.
**MooseFS** — *Moose File System,* klasterski dijeljeni datotečni sustav.
**MOSFET** — *Metal-Oxide Semiconductor FET*, tehnologija poluvodiča.
**Motherboard** — označava matičnu ploču računala (višeslojna štampana ploča na kojoj se nalaze sve komponente računala).
**MPEG** — *Motion Pictures (Coding) Experts Group,* standard za kodiranje i dekodiranje (**CoDec**) video sadržaja.
**mp3** — M*PEG-1 ili MPEG2 audio sloj 3,* format za digitalni zvuk koji koristi oblik kompresije s gubitkom podataka.
**MPIO** - *Multipath IO,* Multipath tehnologija za ubrzavanje pristupa diskovnom (I/O) sustavu.
**Mpps** — *Million Packets Per Second*, milijun (mrežnih) paketa u sekundi [*jedinica za propusnu snagu mrežnog uređaja*].
**MPTCP** - *Multipath TCP,* Multipath tehnologija za ubrzavanje rada TCP protokola.
**MRP** — *Multiple Registration Protocol,* noviji protokol za registriranje pripadnosti VLAN-ovima. Pogledajte stariji **GARP**.
**MS-DOS** — *Microsoft DOS, Microsoftov* operativni sustav.
**MSB** — *Most Significant Bit,* najvažniji bît podataka
**MSI** — *Message Signalled Interrupts,* noviji način kreiranja (slanja) signala prekida (IRQ).
**MSR** - *Model specific register,* CPU registri zaduženi za **MCE** poruke sustava o greškama u radu.
**MT** — *Machine Translation*, strojni (računalni) prijevod.
**MTA** — *Mail Transfer Agent*, agent za prijenos elektroničke pošte
**MTU** — *Maximum Transmission Unit*, maksimalna veličina paketa na mreži (na OSI sloju tri).
**Multicast** - metoda komunikacije u kojoj se jedan paket kopira na više odredišta (pretplatnika *multicast* grupe).
**Multicast Address** - jedinstvena (grupna) IP adresa koja se odnosi na sve pretplatnike te grupe (grupne IP adrese).
**Multipleksiranje** je metoda s kojom se više podataka ili signala kombinira u jedan nîz podataka (ili signal).
**MVRP** - *Multiple VLAN Registration Protocol***,** noviji protokol za registriranje **VLAN** *trunk* veza. Pogledajte **GVRP**.
**MX** — *Mail exchange*, obično označava DNS unos za poslužitelj elektroničke pošte. Pogledajte **DNS**.

## N

**NACK** — *Negative ACKnowedgement,* negativna potvrda.
**NAPI** — *New API*, novi model rada mrežnih sučelja, ako ih podržava mrežna kartica i njen upravljački program.
**NAS** - **Network Access Server** ili *Network-Attached Storage*. Mrežni pristupni poslužitelj ili uređaj ili funkcionalnost mrežnog dijeljenog datotečnog sustava (*NAS*). **NAS** radi s datotekama i direktorijima dok **SAN** radi s blokovima (preko mreže).
**NAT** − *Network Address Translation,* proces ili metoda preslikavanja odnosno translacije IP adresa. Pogledajte i **PAT**.
**NCQ** — *Native Command Queuing,* mehanizam za preraspodjelu zahtjeva za zapisivanjem ili čitanjem na tvrdi disk.
**Netlink** — sučelje između programa (procesa) i linux kernela zaduženog za mrežu.
**Netfilter** — komponenta unutar mrežnog modela linuxa zadužena za filtriranje mrežnih paketa, **NAT**, **PAT** i drugo.
**Netflow** — protokol za slanje statistika mrežnog prometa s mrežnih uređaja. Pogledajte **sFlow.**
**Netmask** — maska mreže: *parametar uz IP adresu, koji definira mrežu i/ili pripadnost pod mrežama (tzv. subnetima).*
**NFS** — *Network Filesystem*, mrežni dijeljeni datotečni sustav. Pogledajte i **CIFS**.
**NFV** — *Network Functions virtualization*, označava arhitekturu kojoj je cilj vrtualizacija funkcionalnosti mrežnih elemenata.
**NG-PON2** - najnoviji *PON* (**GPON**) standard koji omogućuje propusnosti do 40Gbps. Pogledajte **PON** i **GPON**.
**NIC** — *Network Interface Card,* označava mrežnu karticu odnosno fizičko mrežno sučelje.
**NIO** — *New I/O,* novi ulazno/izlazni sustav.
**NIST** — *National Institute of Standards and Technology,* nacionalni (SAD) institut za standarde i tehnologiju.
**NMI** — *Non-Maskable Interrupt*, nemaskirajući signal prekida (**IRQ**) [*posebni signal prekida s najvišim prioritetom*].
**NNTP** — *Network News Transfer Protocol*, mrežni protokol za razmjenu poruka (uglavnom teksta) korisnika.
**NOC** — *Network Operations Center*, mrežni operativni (i nadzorni) centar.
**NOR** — može se odnositi na logičku operaciju negacije **ILI** (**Booleova algebra**).
**NOS** — *Network Operating System*, mrežni operativni sustav.
**NOT** — može se odnositi na logičku operaciju negacije **NE** (**Booleova algebra**).
**NSA** — *National Security Agency,* nacionalna (SAD) agencija za sigurnost.
**NSS** — *Name Service Switch*, servis i/ili mehanizam za imenične servise i/ili rad s korisničkim računima i grupama.
**NSS** — *Network Security Services,* skup servisa za sigurnost sustava.
**NTFS** — *NT Filesystem, Windows* datotečni sustav koji je razvila tvrtka *Microsoft.*
**NTP** - *Network Time Protocol*, mrežni protokol za sinkronizaciju točnog vremena s udaljenim **NTP** poslužiteljem.
**NVGRE** - *Network Virtualization using Generic Routing Encapsulation*, protokol za povezivanje izoliranih LAN mreža, koji se oslanja na **GRE** protokol za ugnježđivanje protokola na OSI sloju dva unutar IP protokola.

**NUMA** — *Non-Uniform Memory Access*, neuniformirani način pristupu memorije; odnosi se i na sâmu arhitekturu računala.
**NVM Express** (**NVMe**) — *Non-Volatile Memory Host Controller Interface Specification*, sučelje za spajanje SSD i drugih ekstremno brzih diskova. Ono je praktično veza na **PCI express** (**PCIe**) sabirnicu te nudi velike brzine prijenosa podataka.
**NX** – *NX (No-Execute) bit*, mehanizam zaštite memorije.

# O

**OCFS2** — *Oracle Cluster File System,* klasterski dijeljeni datotečni sustav.
**OCI** — *Open Container Initiative,* industrijski standard koji definira format i način pokretanja Linux kontejnera.
**ODBC** — *Open Database Connectivity*, mehanizam za povezivanje s bazama podataka.
**OEM** — *Original Equipment Manufacturer*, označava proizvođača (tvrtku) uređaja odnosno opreme/računala.
**Offline** — označava stanje veze ili rada u kojem sustav (računalo) nije spojeno na mrežu ili Internet.
**OLE** — *Object Linking and Embedding*, način povezivanja i ubacivanja objekata (sadržaja/podataka) između aplikacija.
**OLT -** *Optical Line Terminal,* centralna oprema za upravljanje **GPON** mrežom. Pogledajte **GPON** i **FTTH**.
**OM -** *Optical Multi Mode cable*, kategorija optičkih vlakana za *multi mode* komunikaciju (**OM-1** do **OM-5**).
**Omni-Path** - *mrežni komunikacijski standard u mrežama za povezivanje superračunala (konkurencija **Ethernetu**).*
**Online** — označava stanje veze ili rada u kojem je sustav (računalo) spojen (i dostupan) na mrežu ili Internet.
**OO model** − *Object-Oriented Model*, objektno usmjeren model programiranja.
**OpenFlow** — protokol za konfiguraciju SDN komponenti (pr. naprednih preklopnika, ***Open vSwitcha*** i slično).
**Open Source –** označava programe ili sustave koji su otvorenog programskog kôda (tzv. „*slobodan*“ softver).
**OpenVPN** — mrežni protokol koji implementira upotrebu **VPN** mogućnosti. Pogledajte: **VPN**, **IPSec**, **Wireguard**.
**OpenVZ** – jedna od tehnologija Linux kontejnera. Pogledajte **mikro kontejnere**, **LXC, Docker** i **Podman**.
**OR** — može se odnositi na logičku operaciju **ILI** (**Booleova algebra**).
**OS** – *Operating System* ili *Open Source*, operativni sustav ili pojam za sustave otvorenog programskog koda.
**OS -** *Optical Single Mode cable*, kategorija optičkih vlakana za *single mode* komunikaciju (**OS-1** ili **OS-2**).
**OSI** — *Open Systems Interconnection*, slojeviti standard mrežne komunikacije.
**OSPF** — *Open Shortest Path First*, protokol za usmjeravanje.
**OSS** — *Open Sound System*, otvoreni sustav za zvuk (obično pod Linuxom) ili *Open Source Software* – slobodan softver otvorenog programskog kôda.
**Oversubscription** – odnosi se na prekomjernu pretplatu odnosno prekomjerno korištenje ili zauzimanje resursa.

# P

**PAC** — *Proxy Auto-Configuration,* format/vrsta autokonfiguracijske datoteke za Proxy klijenta.
**Packet,** *paket* - logička cjelina koja se sastoji od podataka (informacija) i zaglavlja s kontrolnim podacima, koji su definirani određenim protokolom. Obično se odnosi na jedan logički blok podataka koji se šalje mrežom. Pogledajte **Protokol**.
**PAM -** *Pluggable authentication module,* modularni sustav za autentikaciju korisnika.
**PAP** — *Password Authentication Protocol,* protokol za autentikaciju.
**PARC** — *Palo Alto Research Center*, razvojni centar za istraživanje novih tehnologija u gradu *Palo Alto*, države *Kalifornija*.
**Partition** — *Particija,* regija (područje) diska koje se može formatirati s datotečnim sustavom. Pogledajte i **Format(iranje)**.
**Password** — odnosi se na zaporku (lozinku).
**PAT** — tehnologija preslikavanja portova, vezana za translaciju adresa odnosno **NAT**.
**Patch Cable** — kabel za povezivanje između računala i mrežne utičnice s jedne strane i **Patch Panela** s druge.
**Patch Panel** — prespojni panel koji se ugrađuje u komunikacijski ormar kao (pasivna) veza između položenih kablova i aktivne mrežne opreme. Prema logici rada on je središnja (pasivna) točka u strukturnom kabliranju.
**PATA** — *Parallel ATA*, paralelno ATA sučelje za spajanje tvrdih diskova i CD/DVD-ROM uređaja.
**PC** – *Personal Computer*, osobno računalo, odnosno takozvano **PC** kompatibilno.
**PCAP** – *Packet Capture,* označava **API** za dohvaćanje/snimanje mrežnih paketa, ali i format istoimene datoteke.
**PCB** — *Printed Circuit Bord*, štampana pločica.
**PCN -** *private cloud network*, izolirana mreža unutar sustava virtualizacije dostupna virtualnim računalima. Pogledajte **VCN**.
**PCI** — *Peripheral Component Interconnect*, vrsta (starije) sabirnice računala.
**PCIe** — *PCI Express,* novija vrsta sabirnice velike propusnosti, nasljednik **PCI** sabirnice računala.
**PCM** — *Pulse-Code Modulation*, način modulacije signala.
**PCMCIA** — *Personal Computer Memory Card International Association*, sučelje za povezivanje uređaja (posebnih kartica) s računalom.
**PD** — *Public Domain,* javna domena: označava javno dostupne podatke, obično slobodne za korištenje bez ikoje licence.
**PDF** — *Portable Document Format,* format zapisa dokumenata koji uključuje: tekst, vektorsku grafiku, rasterske slike, a može sadržavati i fontove. Ovaj format je neovisan o aplikacijskom softveru i operativnom sustavu. Baziran je na **PostScript** jeziku.
**PEM** – *Privacy Enhanced Mail format,* format datoteke koja sadrži kriptografske ključeve, certifikate i drugi kripto sadržaj.
**PERL** — *Practical Extraction and Reporting Language*, programski jezik.
**Permissions** – ovlasti odnosno prava pristupa, poput ovlasti nad datotekama i direktorijima (mapama).
**PF** - *Physical Function,* kod **SR-IOV** mrežnih kartica se odnosi na fizičku mrežnu karticu.
**PGA** — *Pin Grid Array*, vrsta pakiranja (izrade) elektroničkih sklopova.
**PGP** — *Pretty Good Privacy*, kriptografski program/sustav.
**PHP** – *Hypertext Preprocessor*, objektno orijentirani programski jezik namijenjen programiranju dinamičnih web stranica.
**PHY** – označava fizički sloj prema OSI modelu, obično se odnosi na elektroniku mrežnog sučelja, prema fizičkom mediju.
**PIC** — *Peripheral Interface Controller*, vrsta mikrokontrolera, zaduženog za rad sa signalima prekida (IRQ).
**PIC** — može biti i izvor vremena (brojač vremena). Pogledajte: **ACPI PM**, **APIC**, **HPET**, **RTC** i **TSC**.

**PID** – *process ID*, Identifikator (broj) svakog pokrenutog programa (*procesa* prema Linux terminologiji).
**PKCS** — *Public Key Cryptography Standards*, standard u kriptografiji javnog ključa.
**PKI** — *Public Key Infrastructure*, infrastruktura kriptografskog sustava bazirana na javnim i privatnim ključevima.
**PLC** – *Programmable Logic Controller*, programibilni logički upravljač (kontroler).
**PNG** — *Portable Network Graphics*, rasterski grafički format datoteka.
**PnP** — *Plug-and-Play,* tehnologija prepoznavanja novog hardvera tijekom spajanja na sustav.
**Podman** – jedna od tehnologija Linux kontejnera. Pogledajte **mikro kontejnere**, **LXC, Docker** i **OpenVZ**.
**PoE** — *Power over Ethernet,* tehnologija napajanja preko **ethernet** mreže.
**POP** – *Post Office Protocol*, protokol za elektroničku poštu.
**POP3** — *Post Office Protocol v3*, mrežni protokol za elektroničku poštu, treće inačice.
**PON -** *Passive optical network,* pasivna optička mrežna tehnologija. Pogledajte **GPON** i **FTTH**.
**Port** — prema TCP/IP protokolu (na razini TCP ili UDP) to je 16. bitni broj koji identificira aplikacijski protokol i konekciju.
**Port mirror** – metoda kopiranja cjelokupnog mrežnog prometa s jednog mrežnog sučelja na drugo. Pogledajte **SPAN**.
**POSIX** — *Portable Operating System Interface*, skup standarda i definicija načina rada sustava, u svrhu kompatibilnosti između operativnih sustava (pr. Unix ←→ Linux). POSIX se izvorno odnosi na standard **IEEE Std 1003.1-1988** iz 1988 g.
**PPP** — *Point-to-Point Protocol,* mrežni komunikacijski protokol između usmjerivača.
**PPPoA** — *PPP over ATM,* **PPP** mrežni protokol preko **ATM** mreže.
**PPPoE** — *PPP over Ethernet,* **PPP** mrežni protokol preko **Ethernet** mreže.
**PPTP** — *Point-to-Point Tunneling Protocol*, mrežni komunikacijski protokol za tuneliranje (**VPN**).
**Prime numbers –** svi prirodni brojevi veći od 1 koji su djeljivi samo s 1 i sa samim sobom. Često se koriste u kriptografiji.
**Process** — u Unix i Linux operativnim sustavim označava pokrenuti program.
**Process/task scheduler** - mehanizam zadužen za raspodjelu izvršavanja procesa. Pogledajte: **CFQ** i **EEVDF** algoritme.
**Procfs** — pseudo datotečni sustav koji se montira u `/proc` direktorij (mapu). Pogledajte: **sysfs, debugfs** i **tmpfs** te **Filesystem**.
**Protokol** - opisuje skup pravila i standarda koji definiraju način na koji se informacije mogu dijeliti između dvaju strana u komunikaciji [*ovdje govorimo o komunikacijskim protokolima*]. Pogledajte: **TCP/IP**, **TCP**, **UDP**, …
**Proxy (Server)** - program ili sustav koji je posrednik u komunikaciji između krajnjih točaka komunikacije.
**Proxy ARP** - program ili značajka sustava koji je posrednik u komunikaciji ARP protokolom.
**PS** — *PostScript*, opisni programski jezik za vektorski opis stranice (za ispis/stolno izdavaštvo i pisače).
**PS/2** — *Personal System/2*, osobna računala ili sučelje *PS/2* za spajanje miševa i tipkovnica na računalo.
**PSU** — *Power Supply Unit*, napajanje (komponenta).
**PSTN** — *Public switched telephone network*, uopćeni naziv za (javne) telefonske centrale (analogne ili digitalne).
**PV** — *Physical Volume,* element **LVM** sustava (*Logical volume management*/**LVM2**).
**PVG** — *Physical Volume Group,* element **LVM** sustava (*Logical volume management/**LVM2***).
**PXE** — *Pre Execution Environment*, mrežno okruženje (poslužitelj-klijent) koje omogućava pokretanje ili instalaciju operativnog sustava preko mreže. Pogledajte i **Jumpstart** i **Kickstart**.

# Q

**QA** — *Quality Assurance,* odnosi se na osiguranje kvalitete (softvera, hardvera ili bilo kojeg proizvoda).
**QDR** — *Quad Data Rate*, četverostruki (pro)tok podataka.
**Qemu** — emulator virtualnog hardvera za virtualizaciju (otvorenog kôda). Pogledajte i **KVM**.
**QinQ** – tehnologija dvostrukog ugnježđivanja **VLAN** (**802.1Q**) oznaka unutar mrežnih paketa. Definirano u **802.1ad**.
**QLC** — *Quad Level Cell*, vrsta **MLC** (*Flash*) memorije, koja može pohranjivati četiri bita po memorijskoj ćeliji.
**QoS** — *Quality of Service*, sustav/mehanizmi koji osiguravaju kvalitetu (propusnost, kašnjenje, gubici) prijenosa podataka.
**QOTD** — *Quote of the Day,* kratka poruka dana.
**QSFP** – *Quad Small Form-factor Pluggable,* oblik kompaktnog modularnog mrežnog sučelja velike propusnosti.
Pogledajte **SFP** i **SFP+** te **XFP**.
**Queue** – označava vrstu međumemorije sa slijednim pristupom.
**QUIC** — brzi i pouzdani protokol na transportnom sloju mreže, koji se oslanja na **UDP** protokol.

# R

**RADIUS** - *Remote Authentication Dial-In User Service,* protokol koji osigurava pristup mrežnim resursima (u kategoriji **AAA** protokola).
**RAID** – *Redundant Array of Independent Disks*, redundantno (zaliho) polje nezavisnih diskova, odnosno sustav koji se temelji na polju diskova.
**RAM** — *Random Access Memory*, memorija (RAM) s nasumičnim pristupom, odnosi se na radnu memoriju računala.
**RARP** — *Reverse Address Resolution Protocol*, mrežni protokol (reverzni **ARP**).
**RAS** - *Reliability, availability and serviceabilit,* odnosi se na sustav ili komponente koje osiguravaju pouzdan i siguran rad.
**RC** — *Release Candidate,* kandidat za objavljivanje: obično inačica programa ili sustava koja je kandidat da postane stabilna odnosno pouzdana. Obično je to inačica prije zadnje konačne produkcijske i pouzdane inačice.
**RDIMM** - *Registered (Buffered) Memory,* vrsta SDRAM memorije koja ima ugrađen međuspremnik (čip) prema kontrolnim i podatkovnim linijama (na samoj memorijskoj pločici) .
**ReaR** — *Relax-and-Recover,* mehanizam za izradu sigurnosne kopije i povrat cijelog Linux sustava nakon havarije.
**Recovery** - način na koji se sustav ili računalo oporavlja nakon nekog kvara ili softverske greške.
**Redundancy** - redundancija ili zalihost: sposobnost sustava koji se sastoji od više dupliciranih komponenti, da izdrži greške ili kvarove na jednoj od komponenti sustava, bez ispada cijelog sustava.
**Regex** — *Regular Expression*, regularni izraz.

**Repozitorij** — *Repository*, sustav za pohranu i distribuciju softvera (softverskih paketa), certifikata i slično, te pripadajućih informacija.
**Revision control –** sustav za verzioniranje koda. Pogledajte **Git**, **Mercurial** i **SVN**.
**RF** — *Radio Frequency*, radijska frekvencija.
**RFC** - *Request For Comments*, dokumenti (dokumentacija) koji opisuju neki protokol ili standard.
**RGB** — *Red, Green, Blue*, odnosi se na aditivne bolje: *crvena*, *zelena* i *plava*.
**RHEL** — *Red Hat Enterprise Linux*, distribucija ***Red Hat Linuxa***.
**RIP** — Raster Image Processor, procesor ili program za pripremu rasterskih slika (obično prije otiska).
**RIP** — *Routing Information Protocol*, protokol za usmjeravanje.
**RISC** — *Reduced Instruction Set Computer*, računalna arhitektura kod koje se koristi što je moguće manje naredbi procesora na mikrorazini, kako bi one bile što učinkovitije.
**RISC-V** — otvorena (**open source**) arhitektura **RISC** procesora (nekompatibilna s: **x86**, **ARM**, **SPARC**, **MIPS**, …).
**RMI** — *Remote Method Invocation*, način komunikacije između distribuiranih (udaljenih) objekata (programerskih).
**ROM** — *Read Only Memory*, memorija iz koje se može samo čitati, ali ne i zapisivati.
**Root Account** - privilegirani (administratorski) korisnički račun.
**Route** – *Ruta* - označava odnosno definira putanju kroz računalnu mrežu.
**Router** (*usmjerivač*) - mreži uređaj koji koristi razne metrike kako bi uvijek odabrao najbolji put (za pakete) do odredišta.
**Routing** - proces pronalaska najboljeg puta kroz mrežu, do odredišta.
**RPC** — *Remote Procedure Call*, pozivanje udaljenih metoda u distribuiranom mrežnom računalnom sustavu.
**RPM** — *RPM Package Manager*, paketni menadžment distribucija Linuxa baziranih na *Red Hat* Linuxu.
**RSA** — *Rivest Shamir Adleman*, kriptografski algoritam za kriptiranje (šifriranje) i dekriptiranje podataka.
**RSPAN** – metoda udaljenog kopiranja cjelokupnog mrežnog prometa s jednog mrežnog sučelja na drugo. Pogledajte **SPAN**.
**RTC** — *Real-Time Clock*, lokalni sat (obično ugrađen na matičnu ploču računala). Pogledajte: **ACPI PM**, **APIC**, **HPET**, **RTC** i **TSC**.
**RTO** — *Retransmission timeout*, vrijeme koje koristi **RTT** vrijednost za izračun vremena ponovnog slanja paketa (pr. za **TCP**).
**RTOS** — *Real Time Operating System*, specijalizirani operativni sustav za izvršavanje programa u stvarnom vremenu, s minimalnim kašnjenjem u izvršavanju.
**RTS** — *Ready To Send*, poruka koja označava da je sustav spreman za slanje *(pr. kod komunikacije preko serijskog porta/sučelja)*.
**RTT** — *Round Trip Time*, vrijeme prolaska mrežnog paketa od pošiljatelja do primatelja.
**RX** – *Receive*, označava primanje (pr. preko mreže) ili prijemnu stranu.

## S

**SAN** — *Storage Area Network*, dijeljeni diskovni mrežni sustav (s blok uređajima). Pogledajte **FC**, **FCoE**, **SCSI** i **iSCSI.**
**SAN** - *Subject Alternative Name* - polje unutar TLS/SSL certifikata s nabrojanim domenama, poddomenama. Pogl. **TLS**.
**SAS** - *Serial Attached SCSI*, serijsko SCSI sučelje (standard) za priključivanje perifernih jedinica (diskova) na računalni sustav.
**SATA** — *Serial ATA*, serijsko ATA sučelje (standard) za priključivanje perifernih jedinica (diskova) na računalo.
**SCI** - *System call interface*, sučelje između programa (procesa) i sustava (kernela).
**SCP** — *Secure Copy*, protokol za sigurno (kriptirano) kopiranje datoteka preko računalne mreže.
**SCST** - *SCSI Target Subsystem for Linux, Linux* servis koji je iSCSI odredište (target) za iSCSI SAN sustav. Pogledajte i **LIO**.
**Script** — *Skripta*, označava izvršnu datoteku koja sadrži programski kôd koji se interpretira prije pokretanja (bez kompiliranja).
**SCTP** — *Stream Control Transmission Protocol*, napredni sigurni transportni mrežni protokol iz skupa TCP/IP protokola.
**SCSI** – *Small Computer System Interface*, standard (sučelje) za priključivanje perifernih jedinica (diskova) na računalni sustav.
**SDK** — *Software Development Kit*, skup programa i alata za programiranje.
**SDM –** *Space Division Multiplexing*, tehnologija koja koristi više jezgri unutar jednog optičkog vlakna.
**SDN –** *Software Defined Network*, tehnologija za dinamičku konfiguraciju i upravljanje mrežnim sustavima.
**SDRAM** — *Synchronous Dynamic Random Access Memory*, vrsta RAM memorije slična **DRAM** usklađena s brzinom sabirnice.
**SDSL** — *Symmetric DSL*, simetrični **DSL**.
**Sector** — *Sektor*, dio je staze (trake) na disku i označava najmanju moguću fizičku jedinicu za pohranu podataka na disk.
**Server** – poslužitelj ili poslužiteljski program (softver) koji pruža servise (usluge) klijentima.
**Serijalizacija** – proces prevođenja podatkovne strukture u format koji se može pohraniti. Pogledajte **deserijalizacija**.
**sFlow** — protokol za slanje statistika mrežnog prometa s mrežnih uređaja. Pogledajte i **Netflow.**
**SFP, SFP+** - *Small Form-factor Pluggable*, oblik kompaktnog modularnog mrežnog sučelja.
**SFTP** — *Secure FTP*, protokol za sigurno (kriptirano) kopiranje datoteka preko računalne mreže.
**SFTP** — *SSH File Transfer Protocol*, protokol za sigurno kopiranje datoteka preko računalne mreže upotrebom *SSH* protokola.
**SHA** — *Secure Hash Algorithm*, kriptografski algoritam za izračun provjernog zbroja [*hash*]. Pogledajte i stariji **MD5**.
**SHA-2** — noviji i sigurniji (od **SHA)** kriptografski algoritam za izračun provjernog zbroja [*hash*]. Pogledajte i stariji **MD5**.
**SI** — *Système International d'Unités*, oznaka za međunarodni sustav mjernih jedinica.
**SIMM** — *Single Inline Memory Module*, vrsta RAM memorije.
**Simetrično kriptiranje –** mehanizmi u kriptografiji koji koriste isti ključ i za kriptiranje (šifriranje) i dekriptiranje.
**Simulacija** — imitiranje rada sâmo dijela nekih funkcija hardvera s mnoštvom ograničenja (*pojednostavljena* **Emulacija**).
**Sinkrono** — vrsta komunikacije u kojoj se podaci moraju slati u nîzu, te istom brzinom (pr. slanje/primanje).
**SIP** — *Session Initiation Protocol*, mrežni protokol za prijenos videa i zvuka bez kašnjenja.
**Skalarni procesor** — dizajn procesora koji može izvršiti samo jednu instrukciju u svakom taktu. Pogl. **Superskalarni**.
**SLAB** — *Slab allocation*, naziv ili mehanizam za predmemoriju unutar sustava virtualne memorije. Pogl. **SLOB** i **SLUB**.
**SLC** — *Single Level Cell*, vrsta *Flash* memorije, koja može pohranjivati jedan bit po memorijskoj ćeliji.
**SLOB** — *Simple list of blocks*, mehanizam za predmemoriju unutar sustava virtualne memorije. Pogl. **SLAB** i **SLUB**.
**SLUB** — *Unqueued slab allocator*, mehanizam za predmemoriju unutar sustava virtualne memorije. Pogl. **SLAB** i **SLOB**.
**SM** - *Single-mode*, odnosi se na vrstu optičkog vlakna za prijenos svjetlosnih signala.
**SMB** — *Server Message Block*, mrežni protokol za dijeljenje datoteka preko mreže.

**SMBIOS** — *System Management BIOS*, definira strukturu i metodu pristupa čitanja podataka iz **BIOS**a.
**SMBus** — *System Management Bus*, sabirnica za jednostavnu komunikaciju (uobičajeno sa senzorima). Pogl. **I2C**.
**SMP** — *Symmetric Multi-Processing*, arhitektura procesora s više fizičkih (i/ili logičkih) jezgri.
**SMT** — *Simultaneous Multithreading*, tehnika za poboljšanje performansi rada *superskalarnih* procesora.
**SMTP** – *Simple Mail Transfer Protocol*, protokol za prijenos elektroničke pošte.
**SNMP** – *Simple Network Management Protocol*, mrežni protokol za nadzor i upravljanje uređajima u TCP/IP mrežama.
**SNTP**– *Simple Network Time Protocol*, jednostavni mrežni protokol za dohvaćanje/postavljanje točnog vremena.
**SO-DIMM** — *Small Outline DIMM*, vrsta RAM memorije (obično za prijenosna računala).
**SOA** — *Service-Oriented Architecture*, arhitektura sustava orijentirana na servise.
**SOAP** — *Simple Object Access Protocol*, mrežni protokol za razmjenu strukturiranih podataka (obično za web servise).
**SoC** — *System-on-a-Chip*, sustav na elektroničkom sklopu. Odnosi se na integraciju većine elemenata računala u jedan sklop.
**Socket** – identifikator ostvarene veze, sastoji se od: izvorišne i odredišne IP adrese te izvorišnog i odredišnog **Porta**.
**SOCKS** – mrežni aplikacijski Proxy protokol Pogledajte: **WPAD**, **PAC** i **Proxy**.
**Soft link** – meka poveznica, simbolička poveznica na neku drugu datoteku ili direktorij u datotečnom sustavu.
**Soft IRQ** — Software *Interrupt Request*, softverski signal prekida (obično kreiran od strane programa).
**SONET** — *Synchronous optical networking*, posebno dizajnirana sinkrona optička mrežna tehnologija.
**SoM** - *system on a module*, specijalizirana (minijaturna) štampana pločica koja sadrži sve komponente računala.
**Source code –** odnosi se na izvorni programski kôd.
**Source control -** sustav za verzioniranje koda. Pogledajte **Git**, **Mercurial** i **SVN**.
**SPARC** — *Scalable Processor Architecture*, arhitektura procesora (nekompatibilna s **x86**, **ARM** i drugima).
**Sparse file** – posebna vrsta *prorijeđene* datoteke, koja sadrži veće dijelove s nulama (bez podataka).
**SPAN** – metoda kopiranja cjelokupnog mrežnog prometa s jednog mrežnog sučelja na drugo. Pogledajte **Port Mirror** i **RSPAN**.
**SQL** - *Structured Query Language*, standardizirani jezik za definiciju i pristup relacijskim bazama podataka.
**SRAM** — *Static Random Access Memory*, vrsta ekstremno brze RAM memorije koja ne zahtjeva osvježavanje za očuvanje podataka. Ovakva vrsta memorije se koristi primjerice kao **L1**/**L2** ili **L3** memorija unutar procesora (**CPU**-a) ili unutar **ASIC**-a.
**SR-IOV** - *Single-root input/output virtualization*, hardverska emulacija virtualnih mrežnih sučelja na mrežnoj kartici.
**SSD** - *Solid-State Disk/Solid-State Drive*, vrsta diskova koji koriste neku od **Flash** memorija (**SLC**/**MLC**/**QLC**/**TLS**, …)
**SSDFS** — datotečni sustav prilagođen **SSD** i **NVMe** diskovima. Pogledajte i **F2FS**.
**SSH** — *Secure Shell*, mrežni protokol za udaljeni pristup (kriptirani).
**SSL/TLS** – *Secure Sockets Layer/Transport Layer Security*, kriptografski protokoli za sigurnu komunikaciju putem interneta.
**Stack** - sučelje preklopnika velike propusnosti namijenjeno direktnom povezivanju sabirnice više preklopnika (za ulančavanje).
**STP** - *Spanning Tree Protocol*, mrežni protokol za zalihost (redundanciju) i sprječavanje petlje u mreži.
**Strukturno kabliranje** – odnosi se na strukturiranu kabelsku instalaciju mrežnog sustava.
**su** — *superuser*, super korisnik odnosno administrator UNIX i Linux sustava.
**Subnet** — označava pôd mrežu odnosno logičku podjelu IP mreža. Pogledajte **Netmask**.
**Superskalarni procesor** — dizajn procesora koji može izvršiti više instrukcija u svakom taktu. Pogl. S**kalarni procesor**.
**SUS** — *Single UNIX Specification*, odnosi se na pojedinu UNIX specifikaciju.
**SVG** — *Scalable Vector Graphics*, format datoteke s vektorskom grafikom.
**SVN - Subversion** – sustav za verzioniranje koda. Pogledajte **Git** i **Mercurial**.
**Switch** — preklopnik (uređaj) za umrežavanja računala, koji radi na OSI sloju dva (**OSI 2**).
**Switch fabric** — posebna vrlo brza sabirnica unutar preklopnika (*switcha*).
**SWDM** - *Short Wavelength Division Multiplexing*, tehnologija modulacije i multipleksiranja više optičkih signala kroz jedno optičko vlakno.
**Syscall** — sistemski poziv programa prema kernelu (i indirektno ostatku operativnog sustava i hardvera).
**Sysfs** — pseudo datotečni sustav koji se montira u `/sys` direktorij (mapu). Pogledajte: **procfs, debugfs** i **tmpfs** te **Filesystem**.
**Syslog** — servis i mrežni protokol za spremanje log poruka lokalnog i udaljenih (mrežnih) računala ili uređaja.

# T

**Tag** — oznaka (etiketa) odnosno način označavanja; primjerice mrežnih paketa (pr. **802.1Q** i slično)
**Tar** — *Tape archive*, arhiva (inicijalno na trake) i format arhiviranja podataka: datoteka i direktorija.
**TAP** - *Network TUNnel*, virtualno mrežno sučelje koje radi na OSI sloju dva (obično se povezuje s virtualnim računalima).
**Task manager** ili **Task/Process scheduler** — program ili mehanizam zadužen za životni vijek odnosno pokretanje/pauziranje/zaustavljanje programa (procesa). Pogledajte: **CFQ** i **EEVDF** algoritme.
**Tape** — odnosi se na vrpce (trake) ili tračne uređaje za izradu sigurnosnih kopija (arhivu) podataka. Pogledajte i **LTO**.
**Tb** — Terabit, bilijun bitova [*1.000.000.000.000 bitova*].
**TB** — *Terabyte*, terabajt, bilijun bajta [*1.000.000.000.000 bajta*].
**TCAM** — *Ternary Content Addressable Memory*, obično se odnosi na tablicu (*OSI 3* zapisa) u SRAM memoriji usmjerivača.
**Tcl** — *Tool Command Language*, programski jezik.
**Teaming** — tehnologija za povezivanje više fizičkih mrežnih sučelja u jedno logičko sučelje. Pogledajte **LACP**.
**Terminal** — uređaj koji je krajnja točka u komunikaciji [*obično se odnosi na: monitor+miš+tipkovnica*]. Pogledajte **TTY**.
**TCP** – *Transmission Control Protocol*, transportni protokol iz skupa TCP/IP protokola, koji se brine o sigurnoj isporuci poruka.
**TCP/IP** — *Transmission Control Protocol/Internet Protocol*, skup mrežnih komunikacijskih protokola, baziran na **OSI** modelu.
**TDMA** - *Time Division Multiple Access*, tehnika multipleksiranja bazirana na vremenskim intervalima.
**TFTP** — *Trivial File Transfer Protocol*, jednostavni protokol za prijenos datoteka.
**Throughput** — propusnost mreže (pogledajte i **Bandwidth**).
**Thread** — programska nît odnosno dretva: dio programskog kôda koji se može izvršavati paralelno unutar programa.
**TLC** — *Triple Level Cell*, vrsta **MLC** (*Flash*) memorije, koja može pohranjivati tri bita po memorijskoj ćeliji.
**TLS/SSL** – *Transport Layer Security/ Secure Sockets Layer*, kriptografski protokoli za sigurnu komunikaciju putem interneta.

**Tmpfs** — pseudo datotečni sustav. Pogledajte: **procfs**, **debugfs** i **sysfs** te **Filesystem**.
**Token Ring** — mrežna tehnologija u **LAN** mrežama (konkurencija **Ethernet** mrežama koje su danas standard).
**ToS** — *Type of service*, polje u IP zaglavlju paketa, čijom upotrebom se omogućuje klasifikacija (prioritizacija) paketa.
**TPM** - *Trusted Platform Module (ISO/IEC 11889),* kriptografski standard koji za rad koristi poseban sklop (mikrokontroler).
**Transceiver** – primopredajnik, uređaj koji radi konverziju signala (pr. električni u optički) u oba smjera komunikacije.
**TRIM** - funkcionalnost za slanje upozorenja prema SSD disku, koje stranice *flash* memorije je moguće brisati.
**Trunk port** - obično označava mrežno sučelje preklopnika koje pripada u više **VLAN** mreža istovremeno (**802.1Q**).
**Trunk** - može označavati mrežna sučelja preklopnika s posebnom namjenom (pr. **STACK** logičko sučelje).
**TSC** — *Time Stamp Counter*, izvor vremena (brojač vremena). Pogledajte: **ACPI PM**, **APIC**, **HPET**, **RTC i PIC**.
**TSO –** *TCP segmentation offload*, mehanizam za povećanje odlazne mrežne propusnosti. Naziva se i **GSO** ili **LSO**.
**TUI -** *Text User Interface*, korisničko sučelje iz tekstualnog terminala (naredbenog retka odnosno **CLI**).
**TUN** - *Network TUNnel*, virtualno mrežno sučelje koje radi na OSI sloju tri. Obično se koristi za razne tunele [pr. **VPN**].
**TTL** — *Time To Live*, vrijeme života - obično se odnosi na vrijeme života poruke odnosno mrežnih paketa.
**TTY** — *Teletype*, označava terminal u Linuxu. Pogledajte **Terminal**.
**TX** – *Transmit*, označava slanje (pr. preko mreže) ili stranu koja šalje odnosno odašilje.
**TXT** – odnosi se na tekst ili označava tekstualni format datoteke.
**TWDM -** *Time and Wavelength Division Multiplexing,* napredna tehnologija koja kombinira *Time Division Multiplexing* (**TDM**) i *Wavelength Division Multiplexing* (**WDM**) kako bi povećala kapacitet i fleksibilnost PON sustava.

## U

**UAA -** *Universally administered addresses,* opseg **MAC** adresa koje dodjeljuje proizvođač. Pogledajte **MAC** i **MAC-48**.
**UART** — *Universal Asynchronous Receiver Transmitter,* elektronički sklop koji je međuveza između računala i okoline (obično se odnosi na serijski port [**RS232** i sl.]).
**UDMA** — *Ultra DMA*, način prijenosa podataka unutar računala.
**UDP** – *User Datagram Protocol*, transportni protokol iz skupa TCP/IP protokola, koji se ne brine o sigurnoj isporuci poruka.
**UEFI** — *Unified Extensible Firmware Interface*, programsko sučelje između operativnog sustava i računalnog sklopovlja.
**UID** — *User ID*, identifikacijski broj korisnika na UNIX/Linux sustavu.
**UMA** — *Upper Memory Area*, viši memorijski prostor, između 640kB i 1024kB memorije, za **IBM PC** kompatibilna računala.
**UMB** — *Upper Memory Block*, viši memorijski blok, odnosi se na dijelove **UMA** prostora memorije.
**UNIX** – *Uniplexed Information and Computing System* → UNICS → UNIX, vrsta operativnog sustava (preteča Linuxa).
**UNICODE** – način kôdiranja (zapisivanja) alfabeta (pogledajte i **UTF**).
**Unicast** – način komunikacije: jedan-na-jedan.
**UPS** — *Uninterruptible Power Supply*, neprekidni izvor napajanja (s baterijom/akumulatorom).
**Update** — procedura ažuriranja programa (ili cijelog sustava).
**Upgrade** — procedura nadogradnje programa, sustava ili drugih *metapodataka*.
**Upload** — prebacivanje (kopiranje) podataka (datoteka) na udaljeno računalo.
**URI** — *Uniform Resource Identifier*, niz znakova koji definira određeni resurs.
**URL** — *Uniform Resource Locator*, web adresa određenog resursa na mreži (internetu)
**URN** — *Uniform Resource Name*, **URI** koji koristi **URN** shemu za pristup resursu.
**USB** – *Universal Serial Bus*, univerzalna serijska sabirnica.
**User name** – označava korisničko ime korisnika (primjerice prilikom logiranja/prijavljivanja na sustav).
**UTC** — *Coordinated Universal Time*, univerzalna referentna vremenska zona.
**UTF** — *Unicode Transformation Format*, način kôdiranja alfabeta (pr. UTF-8, UTF-16).
**UTP** — *Unshielded Twisted Pair*, neoklopljeni mrežni kabel koji se sastoji od upletenih parica (četiri upletene parice).
**UUCP** — *Unix to Unix Copy*, metoda kopiranja datoteka, ali (nekriptirano) nezaštićeno.
**UUID** — *Universally Unique Identifier*, jedinstveni identifikator; primjerice formatirane particije tvrdog diska.

## V

**VCN -** *virtual cloud network*, izolirana mreža unutar sustava virtualizacije dostupna virtualnim računalima. Pogl. **PCN**.
**VCS -** *Version Control System* - sustav za verzioniranje koda. Pogledajte **Git**, **Mercurial** i **SVN**.
**VCSEL -** *Vertical-Cavity Surface-Emitting Laser*, vrsta lasera za modulaciju snopa svjetlosti kroz optičko vlakno.
**VESA** — Video Electronics Standards Association, standardizacijsko tijelo vezano za uređaje za prikaz slike (*display*).
**VETH -** *Virtual Ethernet Device*, virtualno mrežno sučelje koje radi kao tunel između imeničnih mrežnih prostora Linuxa.
**Vertikalno kabliranje** – dio strukturnog kabliranja koji se odnosi na dovođenje kabela do svih etaža stambenog objekta.
**VF –** *Virtual Function,* hardverski emulirani PCIe mrežni uređaj unutar ***SR-IOV*** mrežne kartice.
**VFAT** — *Virtual FAT*, datotečni sustav **vFAT**.
**VFS** — *Virtual File System*, virtualni datotečni sustav odnosno vršna komponenta Linuxovog diskovnog podsustava.
**VG** — *Volume Group*, grupa diskovnih volumena (za **LVM2**).
**VGA** — *Video Graphics Array*, grafičko sučelje ili vrsta grafičke kartice odnosno standarda.
**VirtIO** — tehnologija za *paravirtualizaciju* (zaobilaženje pune **Emulacije**) u svrhu poboljšanja performansi.
**Virtualizacija** — tehnologija stvaranja virtualne inačice računalnog hardvera odnosno svih komponenti računala.
**VLAN** — *Virtual Local Area Network*, virtualna lokalna mreža (definirana u standardu **802.1Q**).
**VLB** — *Vesa Local Bus*, vrsta sabirnice (starije).
**VLSI** — Very-Large-Scale Integration, vrlo visoki stupanj integracije (odnosni se na elektroničke sklopove).
**VLSM** – *Variable Length Subnet Masking*, promjenjiva duljina maske mreže (*netmaska*). Pogledajte **Netmask**.
**VM** — *Virtual Machine*, virtualna mašina odnosno računalo (nekada se naziva i *gost*).

**VNet** - *Virtual Network*, označava virtualnu (privatnu) mrežu, obično unutar virtualizacijskog (i/ili *Cloud*) sustava.
**VNF** - *Virtual Network Functions,* obično se odnosi na hardversku značajku **SR-IOV** mrežne kartice za virtualizaciju.
**VNI** - *Virtual Network Identifier*, identifikator pripadnosti točno određenoj ***VXLAN*** mreži. Pogledajte **VXLAN**.
**vNIC** - *Virtual Netwok Interface Card*, označava virtualiziranu mrežnu karticu to jest mrežno sučelje. Pogledajte **Virtualizacija**.
**VoIP** – *Voice over Internet Protocol*, protokol za prijenos zvučne komunikacije putem TCP/IP protokola.
**VPC** – *Virtual private cloud*, sustav za virtualizaciju „u oblaku", za izolaciju mrežnih resursa i pripadajućih virtualnih računala.
**VPN** – *Virtual Private Network*, virtualna privatna mreža: zaštićeni komunikacijski tunel za sigurno povezivanje na udaljene mreže. Pogledajte: **IPsec**, **Wireguard** i **OpenVPN**.
**VRRP** – *Virtual Router Redundancy Protocol*, mrežni protokol za redundanciju na OSI sloju tri (pogledajte i **CARP**).
**VRF** – *Virtual routing and forwarding*, tehnologija virtualnog usmjeravanja koja na istom uređaju/računalu omogućuje istovremenu upotrebu više tablica usmjeravanja.
**VTEP** – *VXLAN tunnel endpoints*, označava krajnju točku u *VXLAN* komunikaciji. Pogledajte **VXLAN**.
**VTP** - *VLAN Trunking Protocol*, protokol tvrtke ***Cisco*** za propagaciju VLAN mreža.
**VT-c** – hardverska podrška za virtualizaciju, vezana za mrežu, ugrađena u poslužiteljske mrežne kartice tvrtke **Intel**.
**VT-d** – hardverska podrška za virtualizaciju, vezana za ulazno/izlazne (I/O) operacije, ugrađena u procesore tvrtke **Intel**.
**VT-x** – hardverska podrška za virtualizaciju ugrađena u procesore tvrtke **Intel**.
**VxLAN** – *Virtual Extensible LAN*, virtualno mrežno sučelje (ili tehnologija) za povezivanje izoliranih LAN mreža. Koristi se za segmentiranje mreže ugnježđivanjem unutar IP (UDP) mrežnih paketa, slično, ali naprednije od VLAN tehnologije. Pogledajte **VLAN**.

## W

**W3C** — *World Wide Web Consortium,* međunarodna organizacija za standarde vezane za **WWW**.
**WAN** – *Wide Area Network*, mreža širokog područja (globalna mreža koja se može prostirati tisućama kilometara).
**WAP** — *Wireless Access Point* ili *Wireless Application Protocol*, bežična pristupna točka ili *protokol za bežičnu komunikaciju*.
**Wayland** — komunikacijski protokol za rad s grafičkim sučeljem (potencijalna zamjena za **X Window** sustav).
**WDM** - *Wavelength Division Multiplexing,* tehnologija modulacije i multipleksiranja više optičkih signala kroz jedno optičko vlakno.
**WEP** — *Wired Equivalent Privacy*, protokol za bežičnu komunikaciju.
**Web Browser** – web preglednik, softver (preglednik) za web stranice.
**Wi-Fi** — *Wireless Fidelity*, označava bežičnu mrežu.
**Wildcard** — posebni znakovi (pr. * . $ \ ? ~ | ) koji imaju posebno značenje unutar operativnog sustava.
**Window manager** — sistemski softver za kontrolu i uporabu grafičkog sučelja (**X Window**) za Unix/Linux.
**Wireguard** — mrežni protokol koji implementira upotrebu **VPN** mogućnosti. Pogledajte **VPN**.
**WLAN** – *Wireless Local Area Network*, bežična lokalna mreža.
**WOL** — *Wake-on-LAN,* tehnologija/standard koja omogućuje uključivanje (*paljenje*) računala preko mreže.
**WOM** — *Wake-on-Modem*, tehnologija/standard koja omogućuje uključivanje (*paljenje*) računala preko modema.
**WOR** — *Wake-on-Ring,* obično se referencira na **WOM**.
**WPA** — *Wi-Fi Protected Access*, protokol za bežičnu komunikaciju
**WPAD** – *Web Proxy Auto-Discovery*, protokol/mehanizam za pronalaženje Proxy poslužitelja na mreži.
**WWW** – *World Wide Web*, svjetska mreža, multimedijski računalni sustav za objavljivanje i razmjenu informacija.
**WWID –** *World Wide ID,* jedinstveni identifikator **LUN**-a u **Fibre Channel SAN** sustavima.
**WWN –** *World Wide Name,* jedinstvena adresa u **Fibre Channel** (pr. za *Fibre Channel HBA*)**.**
**WWPN –** *World Wide Port Number,* jedinstvena adresa porta na *Fibre Channel HBA.*
**WYSIWYG** — *What You See Is What You Get*, označava značajku programa u kojem uređujemo dokument, čiji izgled će u konačnici (na ekranu ili ispisan) izgledati identično onome na kojem radimo.

## X

**X.509** — kriptografski standard koji definira format certifikata u sustavu javnih ključeva (**PKI**).
**X11** — *X Window System,* grafički sustav na Unix i Linux operativnim sustavima. Pogledajte **X Window**.
**x86** - arhitektura mikroprocesora (nije kompatibilna s drugim arhitekturama procesora poput: **ARM**, **SPARC**).
**XDM** — *X Window Display Manager*, komponenta zadužena **X Window** sustav za grafičko sučelje.
**XDMCP** — *X Display Manager Control Protocol*, kontrolni mrežni protokol **X Window** grafički sustav.
**XDP** - *eXpress Data Path,* sustav koji osigurava ekstremno brzo dohvaćanje mrežnih paketa. Pogledajte i **DPDK**.
**XG-PON1** - noviji *FTTH - PON* standard koji omogućuje asimetričnu propusnosti do 10Gbps (10/2.5 Gbps). Pogl. **PON** i **FTTH**.
**XG-PON2** - noviji *FTTH - PON* standard koji omogućuje simetričnu propusnosti do 10Gbps (10/10 Gbps). Pogl. **PON** i **FTTH**.
**XML** – *EXtensible Markup Language*, proširivi jezik za označavanja podataka i dokumenata.
**XFree86** — implementacija *X Window sustava,* koja je zamijenjena s **X.org**.
**XFP** — modul za 10Gbps mreže, poput **SFP+** modula, ali malo većih dimenzija.
**XFS** — visoko performantni Unix/Linux datotečni sustav. Pogledajte **Datotečni sustav**.
**X org, X.Org** — referentna implementacija **X Window** sustava.
**XOR** — *Exclusive OR,* odnosi se na logičku operaciju isključivo **ILI** (**Booleova algebra**).
**XSS** — *Cross Site Scripting*, vrsta sigurnosnog propusta kojem su podložne web aplikacije.
**X Window** — grafički sustav i radna okolina (**GUI**) za Unix/Linux operativne sustave.
**XZ** — format kompresije za Unix/Linux operativne sustave.

## Y

**YAML** — *YAML Ain't Markup Language*, strukturirani jezik/format, koji se obično koristi za konfiguracijske datoteke.
**YaST -** *Yet another Setup Tool,* paketni menadžer na **SUSE** distribuciji Linuxa.
**yum -** *Yellowdog Updater, Modified,* paketni menadžer na ***Red Hat*** kompatibilnim distribucijama Linuxa.

## Z

**Zip** – arhivski format (za komprimiranje datoteka i direktorija) kompresije.
**ZFS** – *Zattabyte File system*, napredni datotečni sustav i softverski **RAID** kontroler (sve u jednom).
**zstd** – *Zstandard,* format ekstremno brze kompresije/dekompresije u stvarnom (realnom) vremenu.

**Izvori informacija: (1062),(1063) i druga poglavlja knjige.**

# 29.4. Popis značajnijih datoteka Linuxa

Pogledajmo i popis važnijih konfiguracijskih (i drugih) datoteka i direktorija (mapa), koje smo spominjali u knjizi:

`/.unconfigured` - ako na sustavu u vršnom direktoriju postoji ova datoteka, pokreću se mehanizmi (re)konfiguracije.
`/boot/config-`**`X.y`** - datoteka s konfiguracijskim parametrima s kojima je kernel inačice **`X.y`** kompiliran.
`/boot/initramfs-`**`X.y`** - datoteka koja je inicijalni RAM disk inačice **`X.y`** za kernel iste inačice.
`/boot/symvers-`**`X.y`** - datoteka s listom sistemskih simbola kernela inačice **`X.y`**.
`/boot/System.map-`**`X.y`** - datoteka sa sistemskim simbolima kernela inačice **`X.y`**.
`/boot/vmlinuz-`**`X.y`** - ova datoteka je Linux kernel inačice **`X.y`**.
**`/boot/grub2/`** - direktorij s datotekama u koje se spremaju inicijalne postavke ***GRUB2*** servisa.
`/boot/grub2/grub.cfg` - konfiguracija **GRUB2** *boot loadera*.

---

`/etc/chrony.conf` - inicijalne postavke **Chrony** NTP servisa.
`/etc/conntrackd/` - direktorij s datotekama u kojima se nalazi konfiguracija **Conntrackd** servisa.
`/etc/cron.d/0hourly` - **Cron** skripta koja se izvršava svaki sat (ovdje mogu postojati i druge **Cron** skripte).
`/etc/cron.daily/` - direktorij s **Cron** skriptama koje se izvršavaju na dnevnoj bazi.
`/etc/cron.hourly/` - direktorij s **Cron** skriptama koje se izvršavaju svaki sat.
`/etc/cron.monthly/` - direktorij s **Cron** skriptama koje se izvršavaju na mjesečnoj bazi.
`/etc/cron.weekly/` - direktorij s **Cron** skriptama koje se izvršavaju na tjednoj bazi.
**`/etc/default/`** - direktorij s datotekama u koje se spremaju inicijalne postavke mnogih servisa.
`/etc/default/grub` - inicijalne postavke **GRUB** ili **GRUB2** servisa.
`/etc/default/nss` - inicijalne postavke **NSS** (*Name Service Switch*) servisa.
`/etc/default/useradd` - inicijalne postavke sustava tijekom procedure kreiranja korisničkih računa.
**`/etc/dnf/`** - direktorij s konfiguracijskim datotekama za **dnf** menadžer za softverske pakete.
**`/etc/firewalld/`** - direktorij s konfiguracijskim datotekama za **firewalld** servis (vatrozid).
`/etc/firewalld/firewalld.conf` - inicijalna konfiguracijska datotekama za **firewalld** servis (vatrozid).
**`/etc/frr/`** - vršni direktorij s konfiguracijskim datotekama **FRR** (*Free Range Routing*) mrežnih protokola/servisa.
`/etc/grub.d/` - direktorij s datotekama u kojima se nalaze predlošci **GRUB2** servisa.
`/etc/grub.d/` - direktorij s datotekama u kojima se nalaze predlošci **GRUB2** servisa.
`/etc/iproute2/` - direktorij s datotekama u kojima se nalaze *Policy based route* parametri **iproute2** sustava.
`/etc/iscsi/` - direktorij s datotekama vezanima zs iSCSI sustav.
**`/etc/keepalived/`** - direktorij s datotekama u kojima se nalazi konfiguracija **Keepalived** servisa.
`/etc/keepalived/keepalived.conf` - konfiguracija **Keepalived** servisa.
`/etc/lldpd.d/lldpd.conf` - konfiguracija **LLDP** servisa (protokola).
`/etc/ld.so.conf.d/` - direktorij s datotekama u kojima se nalaze putanje direktorija s bibliotekama (Engl. *Library*).
`/etc/logrotate.d/` - direktorij s datotekama u kojima se nalaze definicije za rad pojedinih servisa, vezano za logiranje poruka upotrebom servisa **logrotate**.
`/etc/lvm/lvm.conf` - inicijalna konfiguracijska datoteka s dodatnim opcijama za **LVM**(2) sustav.
`/etc/mce/mcelog.conf/`- konfiguracijska datoteka ***mcelog*** servisa (za **MCE** poruke o greškama).
`/etc/modprobe.d/` - direktorij s konfiguracijskim datotekama, s opcijama i parametrima, za kernel module.
`/etc/mdadm/mdadm.conf` - konfiguracija MDADM servisa.
`/etc/multipath.conf` - konfiguracija Linux ***Multipath I/O*** servisa.
`/etc/NetworkManager/NetworkManager.conf` - inicijalna konfiguracija **Network Manager** servisa.
`/etc/ntp/` - direktorij s dodatnim konfiguracijskim datotekama za **NTP** servis.
`/etc/pam.d/` - direktorij s konfiguracijskim postavkama za **PAM** sustav.
**`/etc/pki/`** - direktorij s dodatnim konfiguracijskim datotekama i certifikatima za ***PKI*** sustav (SSL/TLS i sl.).
`/etc/pki/tls/` - direktorij s certifikatima za **SSL/TLS** sustav.
`/etc/pki/rpm-gpg/` - direktorij s **GPG** javnim ključevima repozitorija.
**`/etc/qemu-kvm/`** - direktorij s konfiguracijskim datotekama za **QEMU** i **KVM** servise za virtualizaciju.
`/etc/qemu-kvm/bridge.conf` - konfiguracijska datoteka **KVM** servisa za virtualizaciju, vezano za ***Bridge*** sučelja.
**`/etc/rc.d/`** - vršni direktorij unutar kojeg se nalaze poddirektoriji prema **runlevel**-ima.
`/etc/rc.d/init.d/` - vršni direktorij unutar kojeg se nalaze sve **init** skripte.
`/etc/rc.`**`0`**`/` - direktorij unutar kojeg se nalaze **init** skripte (poveznice na `/etc/rc.d/init.d/`) koje se pokreću u **runlevel**-u **0**.
`/etc/rc.`**`1`**`/` - direktorij unutar kojeg se nalaze **init** skripte (poveznice na `/etc/rc.d/init.d/`) koje se pokreću u **runlevel**-u **1**.
`/etc/rc.`**`XY`**`/` - direktorij unutar kojeg se nalaze **init** skripte (poveznice na `/etc/rc.d/init.d/`) koje se pokreću u **runlevel**-u **XY**.
`/etc/rc.d/rc.local` - zadnja skripta koja se izvršava nakon inicijalizacije sustava.
**`/etc/rear/`** - direktorij unutar kojeg se nalaze konfiguracijske datoteke *ReaR* sustava za izradu sigurnosnih kopija sustava.
**`/etc/security/`** - vršni direktorij unutar kojeg se nalaze konfiguracijske datoteke vezane za sigurnosne postavke sustava.
`/etc/security/limits.d/` - konfiguracijske datoteke vezane za dodatne sigurnosne postavke sustava (korisničke).
`/etc/security/limits.conf` – konfiguracijska datoteka u kojoj se ograničavaju resursi sustava.
**`/etc/selinux/`** - vršni direktorij koji sadrži postavke **SELinux** sustava.

/etc/selinux/config - konfiguracijska datoteka koja sadrži osnovnu konfiguraciju **SELinux** sustava.
/etc/selinux/ - vršni direktorij koji sadrži postavke **SELinux** sustava.
/etc/skel/ - sadrži datoteke koje se kopiraju tijekom kreiranja novog korisnika (pr. .bashrc, .bash_profile, …).
**/etc/snmp/** - vršni direktorij koji sadrži postavke **SNMP** (*Simple Network Management Protocol*) servisa.
/etc/snmp/snmpd.conf - konfiguracija **SNMP** servisa.
/etc/sos/sos.conf - konfiguracija ***sos*** sustava za izradu *sos izvještaja*.
**/etc/sssd/** - vršni direktorij koji sadrži postavke **sssd** servisa.
**/etc/sssd/**sssd.conf - konfiguracijska datoteka koja sadrži konfiguraciju **SSS**(d) servisa.
**/etc/ssh/** - vršni direktorij koji sadrži postavke **SSH** servisa (klijent, poslužitelj, ključevi, *moduli* i sl).
/etc/ssh/moduli – SSH datoteka koja sadrži sve **Diffie-Hellman** *module* (*prim* brojeve i parametre).
/etc/ssh_host_**XY**_key – SSH datoteka koja za algoritam (**XY**) sadrži privatni ključ.
/etc/ssh_host_**XY**_key.pub – SSH datoteka koja za algoritam (**XY**) sadrži javni ključ.
/etc/sshd_config – konfiguracijska datoteka **SSH** poslužitelja.
/etc/ssl – simbolička poveznica na /etc/pki/tls/ to jest direktorij (mapu) s certifikatima za **SSL/TLS**.

---

**/etc/sysconfig/** - vršni direktorij koji sadrži inicijalne postavke **sustava i servisa**. **Pogledajte poglavlje: 7.4.**
/etc/sysconfig/authconfig - konfiguracija koja sadrži inicijalne postavke za autentikaciju i logiranje na sustava.
/etc/sysconfig/autofs - konfiguracija koja sadrži inicijalne postavke za sustav montiranja (*mount*).
/etc/sysconfig/clock - konfiguracija koja sadrži inicijalne postavke za postavke vremenske zone i vremena.
/etc/sysconfig/crond - konfiguracija koja sadrži inicijalne postavke **Cron** servisa.
/etc/sysconfig/dhcpd - konfiguracija koja sadrži inicijalne postavke **dhcpd** servisa (DHCP poslužitelja).
/etc/sysconfig/firewalld - konfiguracija koja sadrži inicijalne postavke **Firewalld** servisa.
/etc/sysconfig/grub - konfiguracija koja sadrži inicijalne postavke **GRUB** servisa.
/etc/sysconfig/init - konfiguracija koja sadrži inicijalne postavke **init** servisa.
/etc/sysconfig/iptables-config - konfiguracija koja sadrži inicijalne postavke **iptables** (*firewall*) servisa.
/etc/sysconfig/ipvsadm-config - konfiguracija koja sadrži inicijalne postavke **IPVS** servisa.
/etc/sysconfig/irqbalance - konfiguracija koja sadrži inicijalne postavke **irqbalance** servisa.
/etc/sysconfig/keepalived - konfiguracija koja sadrži inicijalne postavke **keepalived** servisa.
/etc/sysconfig/kernel - konfiguracija koja sadrži inicijalne postavke **kernela** sustava.
/etc/sysconfig/ksm - konfiguracija koja sadrži inicijalne postavke **ksm** servisa.
/etc/sysconfig/network - konfiguracija koja sadrži inicijalne postavke **mreže (pr. Default Gateway)**.
/etc/sysconfig/nftables.conf - konfiguracija koja sadrži inicijalne postavke **nftables** (vatrozid) servisa.
/etc/sysconfig/ntpd - konfiguracija koja sadrži inicijalne postavke **ntp** servisa.
/etc/sysconfig/sshd - konfiguracija koja sadrži inicijalne postavke **ssh(d)** servisa.
/etc/sysconfig/static-routes - konfiguracija koja sadrži statičke rute (izbjegavati ih konf. ovdje).
/etc/sysconfig/sysstat - konfiguracija koja sadrži inicijalne postavke **sysstat** servisa.
**/etc/sysconfig/network-scripts/** - vršni direktorij koji sadrži inicijalne mrežne postavke **sustava**.
/etc/sysconfig/network-scripts/ifcfg-**eth0** – konfiguracija **eth0** mrežnog sučelja.
/etc/sysconfig/network-scripts/route-**eth0** – konfiguracija ruta za **eth0** mrežno sučelje.
/etc/sysconfig/network-scripts/ifdown-eth – konfiguracija za gašenje ***ethernet*** vrste mrežnih sučelja.
/etc/sysconfig/network-scripts/ifup-eth – konfiguracija za aktivaciju ***ethernet*** vrste mrežnih sučelja.
/etc/sysconfig/network-scripts/network-functions – funkcije potrebne za sva mreža sučelja.

---

**/etc/sysctl.d/** - vršni direktorij koji sadrži dodatne **Sysctl** postavke sustava, ako ih nismo pohranili u: /etc/sysctl.conf.
**/etc/systemd/** - vršni direktorij koji sadrži postavke **Systemd** servisa.
/etc/systemd/system.conf - konfiguracijska datoteka koja sadrži inicijalne postavke **Systemd** servisa.
**/etc/udev/** - vršni direktorij koji sadrži dodatne postavke **udev**(**d**) sustava odnosno tzv. UDEV pravila.
/etc/udev/udev.conf – konfiguracijska datoteka s postavkama **udevd** servisa.
/etc/udev/rules.d/ – vršni direktorij koji sadrži dodatna **UDEV** pravila (ako ih ima).
/etc/xinetd.d/ – vršni direktorij koji sadrži konfiguraciju **xinetd** servisa (pr. za *TFTP* servis).
**/etc/yum/** - vršni direktorij koji sadrži konfiguraciju paketnog menadžera **YUM**.
/etc/yum.repos.d/ - vršni direktorij koji sadrži datoteke s definicijom softverskih repozitorija.

---

Slijede konfiguracijske datoteke u direktoriju (mapi) /etc/
/etc/aliases - datoteka s definicijom aliasa za **sendmail** servis (ne pseudonima naredbi).
/etc/anacrontab – konfiguracija **anacron** servisa.
/etc/asound.conf - **ALSA** (*Advanced Linux Sound Architecture*) konfiguracija.
/etc/bashrc - konfiguracija i postavke za **BASH** ljusku.
/etc/containers/ - **Podman** konfiguracija (kontejnera).
/etc/cron.deny - konfiguracija ograničenja upotrebe **cron** servisa (prema korisnicima).
/etc/crontab – konfiguracijska datoteka **cron** servisa.
/etc/chrony.conf – konfiguracijska datoteka **chronyd** servisa (novog **NTP** servisa za točno vrijeme).
/etc/cups – konfiguracijska datoteka **CUPS** servisa (servisa za ispis/printanje pod Linuxom).
/etc/dracut.conf – konfiguracijska datoteka **dracut** servisa (za rad s *initramfs* sustavom).
/etc/dnsmasq.conf – konfiguracijska datoteka **dnsmasq** servisa (***DHCP+BOOTP***, ***TFTP*** i ***DNS*** servis).
/etc/e2fsck.conf – konfiguracijska datoteka servisa za provjeru stanja diska (*e2 fsck*).

`/etc/ethers` ili `/etc/ethertypes` – definicija vrsta i kategorija *Ethernet* mreža (*IPv4, IPv6, 802.1Q, PPP, ARP, …*)
`/etc/exports` – konfiguracijska datoteka **NFS** servisa za eksportiranje mrežnih diskova.
`/etc/filesystems` – konfiguracijska datoteka s popisom trenutno podržanih datotečnih sustava.
`/etc/fstab` – konfiguracijska datoteka s popisom datotečnih sustava i njihovih točki montiranja (*mountpoints*).
`/etc/group` – definicija korisničkih grupa s pripadajućim **GID** brojevima te pripadnosti korisnika tim grupama.
`/etc/grub2.cfg` – konfiguracijska datoteka **GRUB2** sustava (*boot loadera*).
`/etc/gshadow` – sadrži *hash* vrijednost lozinke za svaku korisničku grupu, ako je lozinka za grupu uopće definirana.
`/etc/host.conf` – definicija ponašanja sustava, kako se uparuju imena računala s njihovom pripadajućom IP adresom.
`/etc/hostname` – ova datoteka sadrži ime (ovog) računala.
`/etc/hosts` – sadrži sve poveznice imena računala i pripadajućih IP adresa (IP adresa → ime računala).
`/etc/hosts.allow` – sadrži IP adrese računala (ako je definirano) kojima se dozvoljava pristup na ovaj sustav.
`/etc/hosts.deny` – sadrži IP adrese računala (ako je definirano) kojima se NE dozvoljava pristup na ovaj sustav.
`/etc/inittab` – za sustave koji koriste **init**, definicija *runlevela* u koji se sustav pokreće.
`/etc/inputrc` – globalne postavke za definiciju povezivanja posebnih znakova, za rad u terminalu.
`/etc/issue` – sadrži poruku koju će nam sustav prikazati prije nego se logiramo na fizičko računalo.
`/etc/issue.net` – sadrži poruku koju će nam sustav prikazati prije nego se logiramo na udaljeno računalo.
`/etc/ksmtuned` – konfiguracijska datoteka za **KSM** (*Kernel Samepage Merging*) servis.
`/etc/ld.so.conf` – sadrži putanje u kojima će sustav tražiti nove datoteke s putanjama do biblioteka (*Library's*).
`/etc/libuser.conf` – definicija parametara i opcija vezanih za korisnike i korisničke grupe.
`/etc/locale.conf` – definicija regionalnih (lokalnih) postavki sustava.
`/etc/localtime` – simbolička veza na datoteku u kojoj se definira vremenska zona (`/usr/share/zoneinfo/…`).
`/etc/login.defs` – sadrži postavke sustava vezane za korisničke račune i korisničke grupe.
`/etc/logrotate.conf` – konfiguracijska datoteka **logrotate** servisa.
`/etc/machine-id` – sadrži jedinstveni identifikator ovog računala (kao 32.bitni heksadecimalni broj).
`/etc/mail.rc` – sadrži konfiguraciju klijenta elektroničke pošte (`mail` ili `mailx`).
`/etc/mke2fs.conf` – sadrži parametre koji se koriste prilikom formatiranja particija s nekim od *EXT* datotečnih sustava.
`/etc/nscd.conf` – konfiguracijska datoteka **nscd** (*Name Service Cache Daemon*) servisa.
`/etc/nsswitch.conf` – konfiguracijska datoteka **nss** (*Name Service Switch*) servisa.
`/etc/ntp.conf` – konfiguracijska datoteka **ntp** (*Network Time Protocol*) servisa.
`/etc/os-release` – datoteka koja sadrži informacije o inačicama i osnovnim postavkama sustava.
`/etc/passwd` – sadrži imena korisničkih računa, **UID** brojeve, kućni direktorij i drugo, za svakog korisnika.
`/etc/profile` – datoteka koja sadrži inicijalne postavke ***BASH*** ljuske.
`/etc/protocols` –datoteka u kojoj se nalazi definicija svih nižih mrežnih protokola (*IP*, *IPv4*, *TCP*, *UDP*, *RDP*, *DCCP* ...).
`/etc/rc.local` – simbolička veza na datoteku: `/etc/rc.d/rc.local`.
`/etc/rear/local.conf` - konfiguracijska datoteke *ReaR* sustava za izradu sigurnosnih kopija sustava.
`/etc/redhat-release` –datoteka koja sadrži podatke o točnoj inačici *Red Hat* (ili kompatibilne) distribucije Linuxa.
`/etc/resolv.conf` – datoteka u kojoj su definirana pravila za razlučivanje imena i DNS poslužitelji.
`/etc/rsyslog.conf` – konfiguracijska datoteka **rsyslogd** *(syslog)* servisa.
`/etc/sensors.conf ili /etc/sensors3.conf` – konfiguracijska datoteka **SMBus/I2C** senzora.
`/etc/services` – datoteka u kojoj se nalazi definicija svih mrežnih protokola prema imenu i portu (TCP ili UDP)..
`/etc/sestatus.conf` – datoteka koja sadrži listu svih direktorija i datoteka uključenih u **SELinux** kontekste.
`/etc/shadow` – sadrži *hash* vrijednosti lozinki za svakog korisnika uz druge parametre korisničkih računa.
`/etc/shells` –datoteka koja sadrži putanje do validnih ljuski na sustavu (primjerice za: ***bash***, ***sh***, ***ksh***, ***zsh***, …).
`/etc/sudo.conf` – konfiguracijska datoteka ***sudo*** mehanizma.
`/etc/sudoers` – konfiguracijska datoteka za ***sudo*** korisnike i njihova prava.
`/etc/sysctl.conf` – konfiguracijska datoteka koja sadrži ***sysctl*** postavke za optimizaciju sustava.
`/etc/system-release` – datoteka koja ispisuje podatke o distribuciji linuxa.
Za *RedHat* (kompatibilne) distribucije Linuxa pokazuje na: `/etc/redhat-release`.
`/etc/updatedb.conf` – konfiguracijska datoteka koja sadrži postavke za `updatedb` i `mlocate` sustava za brzo pretraživanje datoteka na disku.
`/etc/xattr.conf` – datoteka koja sadrži napredne ovlasti [*atribute]*, ako su definirani.
`/etc/xinetd.conf` - konfiguracijska datoteka ***xinetd*** servisa.
`/etc/yum.conf` – konfiguracijska datoteka s osnovnim postavkama za **YUM** servis.

---

Slijede konfiguracijske datoteke u drugim direktorijima (mapama):
**`/lib/modules/`** – direktoriji koji sadrže datoteke i poddirektorije vezane za kernel i kernel module (upravljačke programe).
`/lib/modules/5.15.3-1.el8.elrepo.x86_64/` – datoteke i direktoriji za kernel inačice: `5.15.3-1`.
`/lib/modules/5.15.3-1.el8.elrepo.x86_64/kernel/drivers/` – za navedeni kernel inačice `5.15.3-1` ovo je direktoriji s datotekama koje su kernel moduli odnosno upravljački programi za pripadajući hardver.

---

**/proc/** – datoteke i direktoriji koji predstavljaju posebne unose u kernelu (konfiguracije, opcije ili parametre rada).
/proc/**PID**/ – direktoriji s brojčanom oznakom svakog procesa, prema **PID** broju, (sadrže sve parametre rada tog procesa.
/proc/… →



Popis ovih datoteka pogledajte u poglavlju: **13.9.1. Direktorij /proc.**

**/root/anaconda-ks.cfg** – ***kickstart*** datoteka koja se kreira tijekom instalacije sustava.



Pogledajte poglavlje: **25.8.4.3. Dio s BOOT MANAGERom i TFTP klijentom - nastavak.**

**/sys/** – datoteke i direktoriji koji predstavljaju posebne strukturirane unose u kernelu.
/sys/… →



Popis ovih datoteka pogledajte u poglavlju: **13.9.2. Direktorij /sys.**

**/usr/** – datoteke i direktoriji koji se koriste za korisničke programe i alate.
/usr/share/hwdata/pci.ids – datoteka koja je baza **PCI ID** identifikatora (hardvera na *PCI* sabirnici).
/usr/share/hwdata/usb.ids – datoteka koja je baza **USB ID** identifikatora (hardvera na *USB* sabirnici).

**/var/** – datoteke i direktoriji koji se koriste za privremene promjenjive (varijabilne) podatke, poput log datoteka i slično.
/var/cache/ – ovdje se nalaze datoteke unutar kojih se čuvaj privremeni podaci.

/var/crash/ – ovdje su datoteke koje sadrže (*kernel*) *dump* datoteke u slučaju rušenja sustava.



Pogledajte poglavlje: **16. Kernel Dump/Crashdump/core dump.**

/var/log/ – ovdje se nalaze datoteke koje sadrže log podatke od sustava ili raznih programa.



Pogledajte njihov popis u poglavlju: **17. Log datoteke.**

/var/run/**xy**.pid – datoteka koju može snimiti program/aplikacija (proces) tijekom pokretanja, a sadrži njegov **PID** broj.

- **PID broj je jedinstveni identifikator (broj procesa) pokrenutog programa!**

**Izvori informacija su prethodna poglavlja knjige te: (1031),(1061)**

# 29.5. Popis često korištenih servisa u Linuxu

Pogledajmo tablicu s popisom često korištenih servisa (*daemona*) i važnih procesa u Linuxu, od kojih smo neke spomenuli, a druge nismo, ali smatramo da će vam biti od koristi.

| Naziv servisa | Opis | Naziv servisa | Opis |
|---|---|---|---|
| `acpid` | Za rad s ***ACPI*** sistemskim pozivima. | `nfslock` | ***NFS*** *lock* funkcionalnost za ***NFS.*** |
| `alsa` | Servis za zvučni podsustav. | `nscd` | ***Name Service Cache Daemon***. |
| `anacron` | Servis za **Anacron** sustav. | `NetworkManager` | Servis Network Manager (*za mrežu*). |
| `apmd` | Servis za „*Power Managemenent*" sustav. | `ntpd` | **NTP** poslužitelj. |
| `atd` | Zadužen za **AT** sustav izvršavanja zadataka u određenom vremenu. | `pop3s` | **POP3** poslužitelj elektroničke pošte. |
| `auditd` | Servis za praćenje sustava (*audit*). | `portmap` | Podrška za **RPC** protokol. |
| `autofs` | Servis zadužen za automatsko montiranje particija. | `postgresql` | ***PostgreSQL*** baza podataka. |
| `bind` | DNS poslužiteljski servis. | `postfix` | Poslužitelj elektroničke pošte. |
| `chronyd` | Servis **Chrony** (za NTP). | `pptpd` | Poslužitelj za ***PPTP*** protokol. |
| `bluetooth` | Servis za *Bluetooth* sustav. | `rngd` | Za generiranje slučajnih brojeva. |
| `cpufreqd` | Servis zadužen za postavke frekvencije rada procesora. | `rsync` | ***Rsync*** servis. |
| `crond` | ***Cron*** servis za izvršavanje zadataka. | `rsyslog` | ***Rsyslog*** (noviji *syslog*) servis. |
| `conntrackd` | Servis za praćenje mrežnih konekcija. | `sendmail` | Poslužitelj elektroničke pošte. |
| `cups` | Zadužen je za sustav za tisak (*štampanje/printanje*). | `smartd` | ***S.M.A.R.T.*** servis. |
| `dbus` | Zadužen je za *Desktop Bus* (***D-bus***). | `smb` | *SAMBA* servis. |
| `dhcpd` | ***DHCP*** servis. | `snmpd` | Servis za **SNMP** protokol |
| `dm` | Zadužen je za grafičko sučelje (X-Server). | `squid` | Proxy poslužitelj (*Squid*) |
| `dnsmasq` | ***DNS***, ***DHCP*** i ***TFTP*** poslužitelj. | `sshd` | SSH servis |
| `firewalld` | Zadužen je za vatrozid (noviji). | `sssd` | ***System Security Services Daemon.*** |
| `frr` | ***Free Range Routing*** servis. | `sysstat` | ***Sysstat*** servis. |
| `gpm` | Zadužen je za konzolni rad miša. | | |
| `haproxy` | Load balancer za ***HTTP***, ***HTTPS*** i ***TCP*** protokole. | **`systemd`** | Vršni proces važan za sustav i sve programe na sustavu(ako se ne koristi **init**). |
| `httpd` | ***Apache*** (web) poslužiteljski servis. | `systemd-journal` | *Systemd* servis za logiranje poruka sustava. |
| `imaps` | Secure ***IMAP*** poslužitelj. | `systemd-logind` | *Systemd* servis za prijavljivanje korisnika na sustav. |
| **`init`** | Vršni (***SysVinit***) proces važan za sustav i sve programe na sustavu. | `systemd-remount-fs` | *Systemd* servis za montiranje particija. |
| `ipop2` | ***POP2*** poslužitelj elektroničke pošte. | `systemd-timedated` | *Systemd* servis za rad sa postavkama vremena (vremenska zona, sat, …). |
| `ipop3` | ***POP*** poslužitelj elektroničke pošte. | `systemd-udevd` | *Systemd* servis za prepoznavanja hardvera. |
| `ipsec` | Poslužitelj za ***IPSEC*** protokol. | `systemd-tmpfiles` | *Systemd* servis za rad s privremenim (*tmp*) datotekama. |
| `iptables` | Servis za vatrozid (stariji). | | |
| `irqbalance` | Zadužen je za balansiranje ***IRQ***-a. | `syslog` | ***Syslog*** servis. |
| `iscsid` | Zadužen za iSCSI Initiator-a. | | |
| `keepalived` | Servis za ***VRRP*** mrežni protokol. | `tuned` | Servis za optimizaciju sustava. |
| `ksm` | Zadužen je za *Kernel Same-page Merging* mehanizam. | `upsd` | Servis za ***UPS*** uređaje. |
| `ksmtuned` | Za automatsko podešavanje ***ksm*** servisa. | `vncserver` | Servis za ***VNC*** protokol. |
| `lldpd` | Servis za ***LLDP*** protokol. | `webmin` | Servis za administraciju sustava. |
| `mysqld` | ***MySQL*** baza podataka | `winbind` | ***Samba Name Server*** (pogl. **samba**). |
| `named` | ***BIND*** (DNS) poslužitelj. | `wine` | ***WINE*** emulator za pokretanje Windows programa. |
| `network` | Mrežni servis (za ***RedHat*** do v.8.x) | `xinetd` | ***Xinet*** servis (Za pokretanje drugih servisa). |
| `nfs` | ***NFS*** poslužitelj. | | |
| `nfsfs` | Komponenta ***NFS*** poslužitelja. | | |
| `nftables` | Novi vatrozid (*firewall*) na Linuxu. | | |

**Izvori informacija su prethodna poglavlja knjige te: (1293)**

# 29.6. Propusnosti sučelja

Pogledajmo propusnosti raznih sučelja od kojih smo neka spominjali u knjizi.

| Sučelje | Propusnost (Upload/Download) | Sučelje | Propusnost (Upload/Download) |
|---|---|---|---|
| **WAN tehnologije** | | **Lokalne (LAN) mreže** | |
| ADSL (G.lite) | 1,5 Mbps / 512 kbps | Token Ring (original) | 4 Mbps |
| SDSL | 2,3 Mbps | Ethernet (10BASE-X) | 10 Mbps |
| T2 | 6,3 Mbps | Fast Ethernet (100Base-X) | 100 Mbps |
| ADSL 2 | 12 Mbps / 1,4 Mbps | Fiber Distributed Data Interface (FDDI) | 100 Mbps |
| ADSL 2+ | 24,5 Mbps / 3,5 Mbps | Gigabit Ethernet (1000BASE-X) | 1 Gbps |
| STS-1 / OC-1 / STM-0 | 51,8 Mbps | Infiniband SDR 1 | 2 Gbps |
| VDSL | 52 Mbps | 2.5 Gigabit Ethernet (2.5GBASE-T) | 2,5 Gbps |
| VDSL2 | 100 Mbps | Infiniband QDR 1 | 8 Gbps |
| T4 | 274 Mbps | 10 Gigabit Ethernet | 10 Gbps |
| OC-9 | 466 Mbps | Infiniband DDR 4 | 16 Gbps |
| OC-24 | 1,2 Gbps | 25 Gigabit Ethernet | 25 Gbps |
| . . . | . . . | Infiniband QDR 4 | 32 Gbps |
| OC-256 | 13,2 Gbps | 40 Gigabit Ethernet | 40 Gbps |
| OC-3072 / STM-1024 | 159,2 Gbps | Infiniband HDR 1 | 50 Gbps |
| **Mobilne mreže** | | 50 Gigabit Ethernet | 50 Gbps |
| GSM CSD (**2G**) | 14,4 kbps / 14,4 kbps | 100 Gigabit Ethernet | 100 Gbps |
| GPRS (**2.5G**) | 57,6 kbps / 28,8 kbps | Omni-path | 100 Gbps |
| EDGE (**2.75G**) [*type 1MS*] | 236,8 kbps / 236,8 kbps | Infiniband HDR 4 | 200 Gbps |
| UMTS (**3G**) | 384 kbps / 384 kbps | 200 Gigabit Ethernet | 200 Gbps |
| EDGE Evolution [*type 2 MS*] | 1894 kbps / 947 kbps | Infiniband EDR 12 | 300 Gbps |
| LTE *Cat 1* (**4 G**) | 10 Mbps / 650 kbps | 400 Gigabit Ethernet | 400 Gbps |
| HSPA (**3.5G**) | 13,98 Mbps / 720 kbps | Infiniband HDR 12 | 600 Gbps |
| HSPA+ (**4G**) | 42,2 Mbps / 11,5 Mbps | **Računalne sabirnice** | |
| LTE *Cat 2* (**4G**) | 50 Mbps / 25 Mbps | I2C | 3,4 Mbps |
| LTE *Cat 3* | 100 Mbps / 50 Mbps | ISA | 66,64 Mbps (8,3 MB/s) |
| . . . | . . . | PCI 32-bit/33 MHz | 1067 Mbps (133 MB/s) |
| LTE Cat 21 | 1,4 Gbps / 300 Mbps | PCI 64-bit/33 MHz | 2,1 Gbps (266 MB/s) |
| LTE *Cat 8* | 3 Gbps / 1,5 Gbps | AGP 1x | 2,1 Gbps (266 MB/s) |
| LTE *Cat 14* | 3,9 Gbps / 1,5 Gbps | PCI Express 1.0 (×1 link) | 2,5 Gbps (250 MB/s) |
| **Bežične (Wireless) mreže** | | PCI Express 2.0 (×1 link) | 5 Gbps (500 MB/s) |
| 802.11 | 2 Mbps | PCI-X DDR 16-bit | 4,26 Gbps (533 MB/s) |
| 802.11b | 11 Mbps | PCI Express 3.0 (×1 link) | 8 Gbps (984 MB/s) |
| 802.11a | 54 Mbps | Unified Media Interface (UMI) (×4 link) | 10 Gbps (1 GB/s) |
| 802.11g | 54 Mbps | PCI Express 1.0 (×4 link) | 10 Gbps (1 GB/s) |
| 802.16 (WiMAX) (**4G**) | 70 Mbps | PCI-X 133 | 8,5 Gbps (1 GB/s) |
| 802.11g (Super G) | 108 Mbps | InfiniBand single 4× | 8 Gbps (1 GB/s) |
| 802.11n (Wi-Fi 4) | 600 Mbps | HyperTransport (800 MHz) | 25,6 Gbps (3,2 GB/s) |
| 802.11ac (Wi-Fi 5) | ~ 6.8 Gbps | PCI Express 3.0 (×4 link) | 32 Gbps (3,94 GB/s) |
| 802.11ad | ~ 7.1 Gbps | DMI 3.0; ×4 link | 40 Gbps (3,94 GB/s) |
| 802.11ax (Wi-Fi 6) | 11 Gbps | PCI Express 5.0 (×4 link) | 128 Gbps (15,75 GB/s) |
| 802.11be (Wi-Fi 7) | 40 Gbps | QPI (4.80GT/s, 2.40 GHz) | 153,6 Gbps (19,2 GB/s) |
| **Druge bežične tehnologije** | | QPI (7.2GT/s, 3.6 GHz) | 230,4 Gbps (28,8 GB/s) |
| irDA | 115 kbps | PCI Express 6.0 (×4 link) | 242 Gbps (30,25 GB/s) |
| Bluetooth 1.1 | 1 Mbps | QPI (9.6GT/s, 4.8 GHz) | 307,2 Gbps (38,4 GB/s) |
| Bluetooth 2.0 + EDR | 3 Mbps | HyperTransport 3.0 | 332,8 Gbps (41,6 GB/s) |
| irDA-FIR | 4 Mbps | HyperTransport 3.1 | 409,6 Gbps (51,2 GB/s) |
| IrDA-VFIR | 16 Mbps | PCI Express 5.0 (×16 link) | 512 Gbps (63,02 GB/s) |
| Bluetooth 3.0 | 25 Mbps | NVLink 1.0 | 640 Gbps (80 GB/s) |
| Bluetooth 4.0 | 25 Mbps | PCI Express 6.0 (×16 link) | 968 Gbps (121 GB/s) |
| Bluetooth 5.0 | 50 Mbps | NVLink 2.0 | 1,2 Tbps (150 GB/s) |
| IrDA-UFIR | 96 Mbps | Infinity Fabric | 4,1 Tbps (512 GB/s) |

*Za sabirnice, ovisno o načinu kodiranja signala; propusnost (pr. Gbps) i efektivna propusnost (pr. GB/s) se mogu razlikovati.*

**Izvori informacija: (1294)**

# 29.7. Usporedba platformi za virtualizaciju

Pogledajmo usporednu tablicu važnijih značajki nekoliko platformi za virtualizaciju:

| Značajka | Proxmox VE | VMware | OpenStack | Hyper-V |
|---|---|---|---|---|
| **Podrška za virtualizaciju** | **Da** (KVM/QEMU) | **Da** | **Da** (KVM/QEMU) | **Da** |
| **Direktna podrška za (Linux) kontejnere** | **Da** (**LXC** nativno)<br><br>**Kubernetes/Docker** (inside VM) | **Kubernetes/Docker** *Potrebna je posebna licenca (vSphere Enterprise Plus ili vSphere with Kubernetes ili VMware Tanzu [više mogućih licenci])* | **Kubernetes /Docker** | **Kubernetes/Docker**<br><br>*Licenca za Windows Server podržava Kubernetes.* |
| **Maksimalan broj CPU jezgri po fizičkom poslužitelju** | 8.192 | 896 | 8.192 | 1024 |
| **Maksimalna količina RAM memorije po fizičkom poslužitelju** | 128 TB | 24 TB | 128 TB | 24 TB |
| **Nativna podrška za *cluster*** | Da | Da | Da | Da |
| **Podrška za *Cluster Resource Scheduler*** | Da (integrirano) | Da *Upotrebom Distributed Resource Scheduler (**DRS**) . Potrebna licenca za vSphere Enterprise Plus edition* | Da<br><br>*Upotrebom OpenStack schedulera (integrirano)* | Da<br><br>*Upotrebom Cluster-Aware Updating (**CAU**)* |
| **Podrška za *High availiability cluster*** | Da (integrirano)<br><br>Podržava kreiranje više HA grupa | Da<br><br>*Upotrebom vSphere High Availability (HA)).*<br><br>*Potrebna minimalno licenca: vSphere Standard edition* | Da (integrirano) | Da Upotrebom: *Windows Server Failover Clustering (**WSFC**). Postoje ograničenja na broj virtualnih računala (ovisno o Windows Server Standard ili Datacenter Edition)* |
| **Maksimalan broj fizičkih poslužitelja u *clusteru*** | **32**+ (*) | **96** (**) | **700** *Nova* ***(***)*** *+Neutron +Glance +Cinder +Keystone, …* | **64** (****) |
| **Mrežni model** | *Bridge i/ili Routing, VLAN tagging* | *Bridge i/ili Routing, VLAN tagging* | *Bridge i/ili Routing, VLAN tagging* | *Bridge i/ili Routing, VLAN tagging* |
| **Redundancija mreže: (*Agregacija/bonding/teaming* [*LACP,Active-Backup, …]*)** | Da (Sve navedeno) | *Potrebna licenca za vSphere Distributed Switch* | Da (Sve navedeno) | *Napredni Switch-Embedded Teaming (**SET**) traži Windows Server Datacenter Edition* |
| **Firewall (vatrozid)** | Da, integrirano u sustav.<br><br>*Standardne Firewall mogućnosti su:*<br>• *Per server*<br>• *Per VM*<br>• *Per cluster* | Potreban ***NSX Firewall*** unutar NSX okruženja, *plaća se dodatna licenca ili upotrebom vSphere Distributed Firewall (**DFW**) koji dolazi uz određene licence vSphere-a (slabije licence imaju neka ograničenja).* | Da, integrirano u sustav. | Da integrirano u *Windows Firewall.* |

| Značajka | Proxmox VE | VMware | OpenStack | Hyper-V |
|---|---|---|---|---|
| **Software Defined Network (SDN)** | Da<br>*Podrška za: VLAN, VxLAN, QinQ, EVPN Linux native i/ili Open vSwitch, Free Range Routing* | Da<br>*NSX-T Data Center*: *VxLAN/Geneve* tunneling protocols. *Plaćaju se licence za NSX: NSX Standard, NSX Advanced i NSX Enterprise Plus)* | Da<br>*Neutron networking s VLAN, GRE, VxLAN, ….* | Da<br>*Hyper-V Extensible Switch/Windows Server Software-Defined Networking (**SDN**) uključeni u licencu Windows Server. Koriste: GRE, VxLAN* |
| **Deduplikacija memorije** | Da<br>(Linux nativno **ksmd** *servis*) | Osnovno *(Transparent Page Sharing) se ne plaća. Potpuna deduplikacije memorije virtualnih računala se plaća preko licenci za: vSphere Standard, Enterprise ili Enterprise Plus* | Da | Samo na *Windows Server Datacenter inačici.* |
| **Koji su podržani sustavi za pohranu virtualnih računala ili (Linux)kontejnera (*storages*)** | **Lokalno:**<br>• ZFS<br>• Direktorij<br>• LVM<br>• LVM-thin<br>**NAS** (*File level*):<br>• CephFS<br>• NFS<br>• CIFS<br>• GlusterFS<br>• ***Proxmox Backup Server*** | **NAS** (*File level*)<br>• NFS<br>• CIFS<br>• CephFS (*ručnom konf.*).<br>• GlusterFS (*ručnom konf.*). | **NAS** (*File level*):<br>• CephFS<br>• NFS<br>• GlusterFS<br>• CIFS | **NAS** (*File level*):<br>• SMB<br>• NFS |
| | **SAN** (*blok*):<br>• ZFS over iSCSI<br>• iSCSI/kernel<br>• iSCSI/libiscsi<br>• Ceph/RBD<br>• FC (*ručnom konf.*) | **SAN** (*blok*):<br>• iSCSI<br>• FC<br>• FCoE<br>• vSAN | **SAN** (*blok*):<br>• Ceph<br>• iSCSI<br>• FC<br>• FCoE | **SAN** (*blok*):<br>• iSCSI<br>• FCoE<br>• DAS |
| **Mogućnost replikacije sustava za pohranu** | Da<br><br>*Pomoću Proxmox Storage replication* mehanizma koji podržava različite metode replikacije virtualnih računala između lokalnih ili dijeljenih sustava za pohranu odnosno lokalnih ili mrežnih dijeljenih diskova. | Da<br>Pomoću "*vSphere Replication*" komponente upotrebom: *vSphere Storage Replication Adapter* (**SRA**) za komunikaciju te *VMware Site Recovery Manager* (**SRM**) mehanizma. *Licenciranje je potrebno unutar vSphere paketa licenci.* | Da<br><br>Ovisi o odabranom sustavu ili komponenti (obično otvorenog koda), koja je zadužena za taj zadatak | Da<br><br>Pomoću komponente zvane "***Hyper-V Replica***" uključene licencno u Windows Server.<br><br>Za "***Storage Replica***" za replikaciju na razini diska potencijalno potrebna nova licenca. |
| **Integriran backup/restore** | Da, kao interni na bilo koji sustav za pohranu i/ili eksterni [pr. ***Proxmox Backup Server***].<br><br>Postoje API i CLI | Dostupno kao komponente:<br>+*vSphere Data Protection* (**VDP**). *Potrebne licence: vSphere Essentials Plus, Standard ili Enterprise.*<br>+*vSphere APIs for Data Protection* (***VADP***)<br>+*third-party* | Potrebna zasebna komponenta:<br><br>Backup as a Service (**BaaS**) | Dostupno kao komponente:<br>• Windows Server Backup<br>• System Center Data Protection Manager (**DPM**)<br>• Ili third-party<br>*Potrebne zasebne licence.* |

| Značajka | Proxmox VE | VMware | OpenStack | Hyper-V |
|---|---|---|---|---|
| **Izrada predložaka i klonova virtualnih računala (templates & clones)** | Da (oboje) | Da<br>*Potrebna VMware vSphere licenca* | Da (oboje) | Da (za klonove). Ne postoji direktna mogućnost izrade predložaka (template), ali postoji indirektna metoda. Nije potrebna dodatna licenca. |
| **Migracija virtualnih računala unutar clustera** | Da | Da<br>*Potrebna licenca za Live migrate [vMotion]) koja se dobiva uz: vSphere Standard, Enterprise ili Enterprise* Plus licence. | Da | Da upotrebom *Live Migration* mehanizma. *Potrebna licenca za Live migrate koja je uključena u Windows Server Datacenter.* |
| **Podržani načini migracije virtualnih računala** | Live, Offline | Live, Offline | Live, Offline | Live, Offline |
| **Podrška za Snapshot virtualnih računala** | Da | Da<br>*Potrebna VMware vSphere licenca.* | Da | Da |
| **Prava pristupanja sustavu** | • Lokalno<br>• Active Directory<br>• LDAP<br>Korisnički računi i grupe | • Lokalno<br>• Active Directory<br>• LDAP<br>Korisnički računi i grupe | • Lokalno<br>• Active Directory<br>• LDAP<br>Korisnički računi i grupe | • Lokalno<br>• Active Directory<br>• LDAP<br>Korisnički računi i grupe |
| **Licenciranje** | **AGPL v.3.**<br>Sve značajke su besplatne/otključane | **Vlasnička licenca (VMware)**<br>+Plaćanje dodatnih značajki | **Apache License2.0**<br>*Postoji model plaćanja podrške raznim tvrtkama* | **Vlasnička licenca (Microsoft)** |
| **Pretplata** | ***Pretplata je dobrovoljna !*** | ***Essential***: 576 U$ godišnje | *Model plaćanja podrške raznim tvrtkama (ovisi o razini podrške)* | ***Windows Server Standard***: 1.069 U$ |
| | ***Community***: 105€ godišnje po fizičkom CPU -u | | | ***Windows Server Datacenter***: 6.155 US |
| | ***Basic***: 325€ godišnje po fizičkom CPU -u | ***Standard***: 1.394 U$ godišnje (*Basic server consolidation, vShield Endpoint, vSphere Replication +* vMotion) | | |
| | ***Standard***: 490€ godišnje po fizičkom CPU -u | ***Enterprise plus***: 4.780 U$ godišnje (*Za dodatne značajke: VM Encryption, vSphere Trust Authority, DRS)* | | |
| | ***Premium***: 980€ godišnje po fizičkom CPU -u | ***Essential plus***: 5.596 U$ godišnje (sve značajke) | | |

**Navedene informacije su informativne i podložne su promjenama** **(za aktualne i precizne podatke kontaktirajte tvrtke vlasnike sustava) !**

* → (1521) ➔ ograničenje zbog *Corosync clustera [rade i sustavi sa 50+ fizičkih poslužitelja u klasteru]*
** → (1522)
*** → (1523).(1524)
**** → (1525)

**Izvori informacija: (1517),(1518),(1519),(1520).(1521),(1522),(1523).(1524),(1525),(1700),(1701),(1702),(1703),(1704).**

# 30. Drugi dodaci

## 30.1. Formati zapisa (ekstenzije) datoteka

| **Ekstenzija** | **Opis** | **Ekstenzija** | **Opis** |
|---|---|---|---|
| `.3ds` | 3D Studio format datoteke | `.emf` | Meta datoteka [*Microsoft Enhanced Metafile*] |
| `.3gp` | Video format na mobilnim uređajima | `.emz` | Meta datoteka komprimirana sa ZIP-om [*Microsoft Enhanced Metafile*] |
| `.7z` | 7 Zip komprimirana arhiva | `.eps` | *Enkapsulirani PostScript* format |
| `.aac` | *Advanced Audio coding* – audio format | `.epub` | Elektronička publikacija za E-čitače |
| `.ac3` | *Dolby Digital* audio format | `.exe` | Izvršni (binarni) programi za: *DOS, Windows, OpenVMS, OS/2 i Symbian* |
| `.aiff` | *Audio Interchange File Format* (Audio format) | `.f`, `.f77` | Programski kôd jezika **Fortran** |
| `.asf` | Streaming Audio/Video format (*Windows*) | `.flac` | *Free lossless codec* – audio format |
| `.asm` | Programski kôd jezika **Assembler** | `.flv` | *Flash* video format |
| `.avi` | Audio/Video kontejnerski format | `.gif` | *Graphics (grafički) Interchange Format* |
| `.awk` | **AWK** skripta [**AWK** programski kôd] | `.gz`, `.gzip` | **GZIP** arhivski komprimirani format |
| `.bas` | Programski jezik **BASIC** [programski kôd] | `.gpg` | *Sigurnosni certifikat (GNU privacy Guard)* |
| `.bat` | *Batch* skriptna datoteka: za *DOS* ili *Windows OS* | `.h` | Zaglavlja za programski jezik **C** |
| `.bin` | Binarna datoteka | `.hh` | Zaglavlja za **C++** programski jezik |
| `.bmp` | Grafički rasterski format (*BitMap Picture*) | `.htm` `.html` | *Hypertext Markup Language programski* kôd |
| `.bz2`, `.bzip2` | **BZIP** komprimirana arhiva | `.icc` | Konfiguracijski profil za boje |
| `.c` | Programski kôd jezika **C** | `.ico` | Windows format za ikone |
| `.cpp`, `.c++` `.cc` | Programski kôd jezika **C++** | `.ini` | Konfiguracijska datoteka |
| `.cab` | *Cabinet* arhiva | `.iso` | ISO 9660 datotečni format |
| `.cdr` | Vektorski format (*Corel Draw*) | `.j2c`, `.jp2` | JPEG 2000 grafički format |
| `.cer` `.crt` | Sigurnosni certifikati | `.jar` | Arhiva (ZIP) programskog jezika **JAVA** |
| `.cgm` | *Computer Graphic Metafile* | `.java` | Programski kôd jezika **JAVA** |
| `.chm` | Kompilirana *help* datoteka (za upute) | `.jpeg` | Grafički rasterski format (za slike) |
| `.class` | Java class datoteka | `.jp2` | *JPEG2000* grafički rasterski format |
| `.cmd` | *Command prompt batch* skripta (*DOS/Win*) | `.js` | Programski kôd **Java Script** |
| `.crt` | Sigurnosni certifikat | `.json` | JSON (*Java Script* objektna notacija) |
| `.css` | *Cascading style sheet* programski kôd | `.jsp` | *Jakarta Server Pages* format |
| `.com` | Izvršni binarni format (**DOS** i **Windows**) | `.kdb` `.kdbx` | *Keepass* baza podataka (S kriptiranim podacima/lozinkama) |
| `.conf` | Konfiguracijska datoteka | `.ko` `.ko.gz` | Linux kernel moduli (*upravljački programi*) |
| `.csv` | *Comma Separated Value* format datoteke | `.ldif` | datoteka s **LDAP** podacima |
| `.deb` | Softverski paket za ***Debian*** distrib. Linuxa | `.lha` | Arhivski (komprimirani) format |
| `.db` | Datoteka baza podataka (pr. *SQLite, Android, iOS, …*) | `.lisp` | Programski kôd **LISP** jezika |
| `.dbs` | Datoteka baza podataka (*MS SQL*) | `.ll` | **LLVM** Asemblerski kôd |
| `.dic` | Rječnik (*Dictionary*) | `.log` | Log datoteka |
| `.djvu` | Format dokumenta (poput **PDF**) | `.lua` | **LUA** programski kôd |
| `.dll` | Programska biblioteka (*Windows*) (*Dynamic Link Library*) | `.lz` | Arhivski komprimirani format |
| `.doc`, `.docx` | *MS Word* dokument | `.m` | **MatLab** format datoteke |
| `.dochtml` `.docmhtml` | Microsoft Word (Mime) HTML dokumenti | `.mdf` | **SQL server** (*Microsoft*) format baze podataka |
| `.dot`, `.dotm` | Microsoft Word predlošci (dokumenti) | `.mid` | Standardna *MIDI* (audio) datoteka |
| `.dtd` | *Document Type Definition* za *markup* jezike (pr.: GML, SGML, XML, HTML) | `.mobi` | Elektronička publikacija za E-čitače |
| `.dwf` | Autodesk dokument (za pregled) | `.mov` | Animacija (Video) [*Quick Time*] |
| `.dwg` | Autodesk dokument (za crtanje) | `.mp1` | Audio (MPEG) format (*MPEG Layer 1*) |
| | | `.nft` | Nftables pravila (za vatrozid) |

| Ekstenzija | Opis | Ekstenzija | Opis |
|---|---|---|---|
| .dxf | Vektorski format datoteke | .mp2 | Audio (MPEG) format (*MPEG Layer 2*) |
| .efi | *Extensible Firmware Interface format* | .mp3 | *Audio MPEG audio stream, layer 3* |
| .elf | Izvršna binarna datoteka za Linux [*Executable and Linkable File*] | .mp4 | Multimedijski kontejner format (Audio/Video) |
| .email | *Outlook express* format datoteke | .mpeg | Multimedijski format (Audio/Video) |
| .msi | Instalacijski softverski paket (Windows) | .scala | Programski kôd jezika **Scala** |
| .mso | Metapodaci (MS Outlook) | .scd .sla | Format datoteke programa **Scribus** |
| .o | Kompilirana binarna C/C++ datoteka | .sec | Tajni ključ (za **PGP** sustav) |
| .obj | Objektni programski kôd | .sfx | Samo ekstrahirajuća arhivska datoteka |
| .odb | Otvoreni format za baze podataka. | .sh | Shell skripta (*Unix/Linux*) |
| .odf | Otvoreni format za dokumente, formule i druge sustave. | .shar | Samo ekstrahirajuća arhivska datoteka |
| .odg | Otvoreni vektorski format za dokumente | .svg | Vektorski grafički format datoteka |
| .ods | Otvoreni format za dokumente | .sql | *Structured Query Language* datoteka (Za baze poput: ***MySQL***, ***MariaDB***, ...) |
| .odt | Otvoreni format za dokumente | .sqlite | Datoteka **SQLite** baze podataka |
| .ogg | *Vorbis* audio u OGG kontejneru | .swf | *Shockwave Flash* multimedija |
| .ova | *Open Virtual Appliance* format | .tar | **TAR** arhivska datoteka (Unix/Linux) |
| .ovf | *Open Virtualization Format* | .tcl | Programski kôd jezika **TCL** |
| .ovpn | *Open VPN konfiguracija (klijenta)* | .temp .tmp | Privremena datoteka |
| .p | Programski kôd jezika **Pascall** | .tex | TeX format dokumenta |
| .p12 .pfx | Za spremanje certifikata i ključeva (**PKCS #12**) | .tga | *Truevision TGA* - rasterski grafički format (s kompresijom) |
| .pcap .pcapng | *Packet capture* formati za programe poput: *tcpdump*,*Wireshark*, *tshark*. | .tgz | TAR + GZIP komprimirana arhiva |
| .pdf | *Portable Document Format* (Prijenosni format dokumenta) | .tiff | *Tag Image File Format* - rasterski grafički format (s kompresijom) |
| .pdi | *Portable Database Image* datoteka | .ttf | *True Type font (font datoteka)* |
| .php | Programski kôd jezika **PHP** | .troff | *Text Processor for Typesetters* format |
| .pid | Datoteka koja sadrži ***PID*** broj procesa | .txt | Tekstualna datoteka |
| .png | *Portable Network Graphic* – komprimirani (bez gubitaka kvalitete) rasterski grafički format | .uha | *Ultra High Archive Compression* – arhivska komprimirana datoteka |
| .ppm | *Portable Pixmap* rasterski grafički format | .vbox | *Virtual Box* konfiguracija (**XML** format) |
| .ppk | *Putty* format za privatne (SSH) ključeve | .vbs | *Visual basic* skripta |
| .pps .ppt .pptx | *Power point* prezentacije | .vbx | *Visual basic ekstenzija* |
| .ps | *Adobe Post Script* datoteka | .vdi | Disk image format (za virtualno računalo) |
| .pst | *MS Outlook* arhivska datoteka | .vhd .vhdx | Disk image format (za virtualno računalo) |
| .ps1 | *Windows Powe shell* skripta | .vmdk | Disk image format (za virtualno računalo) |
| .pem | *Privacy-enhanced Electronic Mail* (Sigurnosni certifikat) | .vpc | Disk image format (za virtualno računalo) |
| .pub | *Sigurnosni ključevi (javni ključ) ili Microsoft Publisher* format | .vob | *Video object* kontejnerski format za DVD |
| .py | Programski kôd jezika **Python** | .wav | Audio: *Microsoft Windows RIFF WAVE* |
| .qcow2 | *QCOW2* Disk image format (Za virtualno računalo) | .wma | Audio: *Windows Media Audio* |
| .qt | *Quick Time* video format | .wmv | Video: *Windows Media Video* |
| .rar | Komprimirana arhiva | .xcf | Grafički format za program ***GIMP*** |
| .raw | Disk image format (za virtualno računalo) ili rasterski grafički format bez kompresije | .xls .xlsx | *Microsoft Excel* format datoteke |
| .rb | Programski kôd jezika **Rubby** | .xml | *eXtensible Markup Language* jezik |
| .rdp | Za udaljeni pristup preko **RDP** protokola | .xmf | Audio format (*Extensible Music Format*) |
| .repo | Konfiguracija softverskog repozitorija | .xps | *Open XML Paper Specification* format |
| .rm | Audio format *Real Media* | .xz | Unix/Linux komprimirana datoteka |
| .rpm | Softverski paket za ***Red Hat*** distr. Linuxa | .yuv | Sirovi (*raw*) video format |
| .rs | Programski kôd jezika **Rust** | .yml .yaml | **YAML** programski kôd ili *markup* datoteka |
| .rtf | *Rich Text* format | .z | Unix/Linux komprimirana datoteka |
| .run | Izvršna instalacijska datoteka (Unix/Linux) | .zip | Komprimirana datoteka (***ZIP***) |
| | | .zst | Komprimirana datoteka (***ZSTD***) |

**Izvori informacija: (1058),(1059).**

## 30.2. TCP i UDP portovi u čestoj primjeni

Pogledajmo tablicu s TCP i UDP portovima i pripadajućim mrežnim servisima koji su često u upotrebi:

| Broj porta | Opis | Broj porta | Opis |
|---|---|---|---|
| 9 | **Wake on Lan** | 443 | **Hypertext Transfer Protocol Secure (HTTPS)** |
| 18 | **Message Send Protocol** | 445 | **Microsoft-DS (Directory Services) Active Directory i SMB file sharing** |
| 20 | **FTP Data** | 464 | **Kerberos Change/Set password** |
| 21 | **FTP control** | 465 | **SMTP over SSL/TLS (SMTPS)** |
| 22 | **SSH protokol** | 500 | **Internet Security Association and Key Management Protocol (ISAKMP) Internet Key Exchange (IKE)** |
| 23 | **Telnet protokol** | 512 | **Remote Process Execution (Rexec)** |
| 25 | **Simple Mail Transfer Protocol (SMTP)** | 513 | **Rlogin Servis** |
| 43 | **Whois protokol** | 514 | **Remote shell / Syslog (UDP 514 ILI TCP 6514)** |
| 49 | **TACACS protokol** [AAA protokol] | 515 | **Line Printer Daemon (LPD)** |
| 53 | **Domain Name System (DNS)** | 520 | **Routing Information Protocol (RIP)** |
| 67 | **Bootp (DHCP) Poslužitelja (Server)** | 521 | **Routing Information Protocol Next Generation (RIPng)** |
| 68 | **Bootp (DHCP) Klijent** | 530 | **Remote procedure call (RPC)** |
| 69 | **Trivial File Transfer Protocol (TFTP)** | 543 | **Kerberos login – klogin** |
| 70 | **Gopher protokol** | 544 | **Kerberos Remote shell - kshell** |
| 79 | **Finger protokol** | 546 | **DHCPv6 client** |
| 88 | **Kerberos protokol** | 547 | **DHCPv6 server** |
| 109 | **Post Office Protocol v.2 (POP2)** | 548 | **Apple Filing Protocol (AFP) preko TCP** |
| 110 | **Post Office Protocol v.3 (POP3)** | 554 | **Real Time Streaming Protocol (RTSP)** |
| 111 | **(Sun) Remote Procedure Call (RPC)** | 563 | **NNTP preko TLS/SSL (NNTPS)** |
| 115 | **Simple File Transfer Protocol (SFTP)** | 587 | **Email message submission (SMTP)** |
| 119 | **Network News Transfer Protocol (NNTP)** | 631 | **Internet Printing Protocol (IPP). Common Unix Printing System (CUPS)** *adm. konzola* |
| 123 | **Network Time Protocol (NTP)** | 636 | **Lightweight Directory Access Protocol preko TLS/SSL (LDAPS)** |
| 137 | **NetBIOS Name Service (***Name registration***)** | 639 | **Multicast Source Discovery Protocol (MSDP)** |
| 138 | **NetBIOS Name Service (***Datagram Service***)** | 674 | **Application Configuration Access Protocol (ACAP)** |
| 139 | **NetBIOS Name Service (***Session Service***)** | 694 | **Linux-HA high-availability heartbeat** |
| 143 | **Internet Message Access Protocol (IMAP)** | 700 | **Extensible Provisioning Protocol (EPP)** |
| 156 | **Structured Query Language (SQL) Servis** | 749 | **Kerberos (protokol) - administracija** |
| 161 | **Simple Network Management Protocol (SNMP)** | 750 | **Kerberos (protokol) inačica IV** |
| 162 | **Simple Network Management Protocol (SNMPTRAP) [***Trap***]** | 751 | **Kerberos autentikacija - kerberos_master** |
| 177 | **X Display Manager Control Protocol (XDMCP)** | 752 | **Kerberos password (***kpasswd***) - passwd_server** |
| 179 | **Border Gateway Protocol (BGP)** | 753 | **Kerberos userreg server - userreg_server** |
| 194 | **Internet Relay Chat (IRC)** | 829 | **Certificate Management Protocol** |
| 209 | **Quick Mail Transfer Protocol** | 830 | **NETCONF preko SSH protokola** |
| 213 | **Internetwork Packet Exchange (IPX)** | 832 | **NETCONF preko HTTPS protokola** |
| 220 | **Internet Message Access Protocol (IMAP), version 3** | 853 | **DNS preko TLS** |
| 319 | **Precision Time Protocol (PTP)** *event* | 860 | **iSCSI protokol** |
| 320 | **Precision Time Protocol (PTP)** *general* | 873 | **Rsync (protokol za sinkronizaciju datoteka)** |
| 389 | **Lightweight Directory Access Protocol (LDAP)** | 989 | **FTPS Protokol (***data***), FTP over TLS/SSL** |
| 433 | **Network News Transfer Protocol (NNTP)** | 990 | **FTPS Protokol (***control***), FTP over TLS/SSL** |

| Broj porta | Opis | Broj porta | Opis |
|---|---|---|---|
| 992 | **Telnet preko TLS/SSL protokola** | 2727 | **Media Gateway Control Protocol (MGCP)** |
| 993 | **Internet Message Access Protocol preko TLS/SSL (IMAPS)** | 2775 | **Short Message Peer-to-Peer (SMPP)** |
| 995 | **Post Office Protocol 3 preko TLS/SSL (POP3S)** | 2498 - 2949 | **WAP push Multimedia Messaging Service (MMS)** |
| | | 2967 | **Symantec System Center agent** (SSC-AGENT) |
| | **Slijede viši portovi (1.024 – 10.000)** | 3000 | **Ruby on Rails development** |
| 1080 | **Socks proxy** | 3004 | **Apple iSync** |
| 1098 | **Java RMI** | 3020 | **Common Internet File System (CIFS)** |
| 1194 | **Open VPN** | 3128 | **Squid caching web proxy** |
| 1220 | **QuickTime Streaming Server** | 3225 | **Fibre Channel over IP (FCIP)** |
| 1234 | **VLC media player UDP/RTP stream** | 3260 | **iSCSI protokol** |
| 1270 | **Microsoft System Center Operations Manager (SCOM)** | 3268 | **Microsoft Global Catalog** |
| 1293 | **Internet Protocol Security (IPSec)** | 3283 | **Apple Remote Desktop** |
| 1433 | **Microsoft SQL Server database management** | 3386 | GTP' 3GPP GSM/UMTS CDR logging protocol |
| 1503 | **Windows Live Messenger (Whiteboard and Application Sharing)** | 3306 | **MySQL baza podataka** |
| 1521 | **Oracle database default listener**<br>*(za buduće inačice)* | 3478 | **STUN** i **TURN** NAT traversal |
| 1527 | **Oracle Net Services**<br>**Apache Derby Network Server** | 3493 | **Network UPS Tools** (NUT) |
| 1589 | **Cisco VLAN Query Protocol** | 3544 | **Teredo tunneling** |
| 1701 | **Layer 2 Forwarding Protocol (L2F)**<br>**Layer 2 Tunneling Protocol (L2TP)** | 3601 | **SAP Message Server Port** |
| 1707 | **L2TP/IPsec:** za uspostavu i inicijalizaciju veze | 3659 | **Apple SASL** |
| 1719 | **H.323** *registration and alternate communication* | 3689 | **Digital Audio Access Protocol (DAAP)** |
| 1720 | **H.323** *call signaling* | 3799 | **RADIUS change of authorization** |
| 1723 | **Point-to-Point Tunneling Protocol (PPTP)** | 3868 | **Diameter protokol [***AAA protokol***]** |
| 1812 | **RADIUS protokol (Autentikacija) [***AAA protokol***]** | 4001 | ***Etcd*** servis |
| 1813 | **RADIUS protokol (Accounting) [***AAA protokol***]** | 4195 | **AWS cloud remoting** |
| 1883 | **MQTT transport** | 4243 | **Docker** ***impl, redistibution & setup*** |
| 1900 | **Simple Service Discovery Protocol (SSDP)** | 4244 | **Viber** aplikacija |
| 1935 | **Real Time Messaging Protocol (RTMP)** | 4500 | **IPSec NAT Traversal** |
| 1985 | **Cisco Hot Standby Router Protocol (HSRP)** | 4569 | **Inter-Asterisk eXchange (IAX2)** |
| 2049 | **Network File System (NFS)**<br>Ako se radi o **NFSv4** ovo je jedini potreban port | 4604 | **Identity Registration Protocol** |
| 2375 | **Docker REST API (plain)** | 4739 | **IP Flow Information Export** |
| 2376 | **Docker REST API (SSL)** | 4789 | **Virtual eXtensible Local Area Network (VXLAN)** |
| 2377 | **Docker Swarm cluster management** | 5000 | **UPnP, Docker registry,** ***Apple air play receiver*** |
| 2427 | **Media Gateway Control Protocol (MGCP)** | 5001 | **Iperf tool, Synology konzola** |
| 2483 | **Oracle database default listener** | 5004 | **Real-time Transport Protocol media data** |
| 2484 | **Oracle database default listener**<br>*(SSL)* | 5005 | **Real-time Transport Protocol control protocol** |
| 2598 | **Citrix Independent Computing Architecture (ICA) with Session Reliability** | 5060<br>5061 | **Session Initiation Protocol (SIP)**<br>**Session Initiation Protocol (SIP over TLS)** |

# 30.3. Usporedba konfiguracijskih datoteka ovisno o distribuciji Linuxa ili Unixa

U ovoj kratkoj cjelini napravit ćemo usporedbu važnijih konfiguracijskih datoteka, ovisno o distribuciji Linuxa ili Unixa.

| **Konfiguracija mrežnih sučelja (pr. `eth0`)** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle*** | `/etc/sysconfig/network-scripts/ifcfg-eth0` |
| ***SUSE*** | `/etc/sysconfig/network/ifcfg-eth0` |
| ***Debian/Ubuntu*** | `/etc/network/interfaces`<br>**Ili (ako se koristi Netplan):** `/etc/netplan/*.yaml` |
| ***FreeBSD Unix*** | `/etc/rc.conf` → ***FreeBSD*** drugačije naziva mrežna sučelja |
| ***Solaris Unix*** | `/etc/hostname.e1000g0` → ***Solaris*** drugačije naziva mrežna sučelja |

| **Konfiguracija globalnih pravila usmjeravanja (*routes*) *[obično podrazumijevanog usmjerivača]*** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle*** | `/etc/sysconfig/network` |
| ***SUSE*** | `/etc/sysconfig/network/routes` |
| ***Debian/Ubuntu*** | `/etc/network/interfaces` |
| ***FreeBSD Unix*** | `/etc/rc.conf` |
| ***Solaris Unix*** | `/etc/defaultrouter` |

| **Konfiguracija dodatnih pravila usmjeravanja (*routes*) na razini mrežnih sučelja (pr. `eth0`)** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle*** | `/etc/sysconfig/network-scripts/route-eth0` |
| ***SUSE*** | `/etc/sysconfig/network/ifroute-eth0` |
| ***Debian/Ubuntu*** | `/etc/network/interfaces` |
| ***FreeBSD Unix*** | `/etc/rc.conf` |

| **Konfiguracija zasebnih pravila usmjeravanja (*policy based routing*) mrežnih sučelja (pr. `eth0`)** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle*** | `/etc/sysconfig/network-scripts/rule-eth0` |
| ***Debian/Ubuntu*** | `/etc/network/interfaces` |
| ***FreeBSD Unix*** | `/etc/rc.conf` |

| ***Network Manager - konfiguracija mrežnih sučelja, svih parametara mreže i usmjeravanja*** | |
|---|---|
| ***NetworkManager - za sve distribucije koje ga koriste [pr. Red Hat 9+]*** | `/etc/NetworkManager/system-connections/` |

| **Konfiguracija imena računala** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle/SUSE*** | `/etc/hostname` |
| ***Debian/Ubuntu*** | |
| ***FreeBSD Unix*** | |
| ***Solaris Unix*** | `/etc/nodename`<br>**Koriste se i:** `/etc/inet/netmasks` **te** `/etc/defaultdomain` |

| **Mapiranje IP adresa u imena računala (IP → hostname)** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle/SUSE*** | `/etc/hosts` |
| ***Debian/Ubuntu*** | |
| ***FreeBSD/Solaris Unix*** | `/etc/hosts` **Solaris koristi i** `/etc/inet/hosts` |

| **Konfiguracija dostupnih DNS poslužitelja** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle/SUSE*** | `/etc/resolv.conf` |
| ***Debian/Ubuntu*** | |
| ***FreeBSD/Solaris Unix*** | |

| **Konfiguracija *name service switch* servisa (NSS)** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle/SUSE*** | `/etc/nsswitch.conf` |
| ***Debian/Ubuntu*** | |
| ***FreeBSD/Solaris Unix*** | |

| **Konfiguracija točki montiranja particija** | |
|---|---|
| ***RedHat/CentOS/Fedora/Rocky/Oracle/SUSE*** | `/etc/fstab` |
| ***Debian/Ubuntu*** | |
| ***FreeBSD Unix*** | |
| ***Solaris Unix*** | `/etc/vfstab` |

| Konfiguracija eksportiranih direktorija (mapa) dijeljenih preko mrežnog NFS servisa | |
|---|---|
| *RedHat/CentOS/Fedora/Rocky/Oracle/SUSE* | **/etc/exports** |
| *Debian/Ubuntu* | **/etc/exports** |
| *FreeBSD Unix* | **/etc/exports** |
| *Solaris Unix* | **/etc/dfs/dfstab** |

| Konfiguracija bazičnih postavki servisa | |
|---|---|
| *RedHat/CentOS/Fedora/Rocky/Oracle* | **/etc/default/*** |
| *Debian/Ubuntu* | **/etc/default/*** |
| *FreeBSD Unix* | **/etc/rc.conf** |

| Inicijalizacijske skripte (i/ili postavke) servisa | | |
|---|---|---|
| | Init (System V) skripte | Systemd skripte |
| *RedHat/CentOS/Fedora/Rocky/Oracle* | **/etc/init.d/*** (/etc/rc.d/init.d/*) | **/lib/systemd/system/*** |
| *Debian/Ubuntu* | **/etc/init.d/*** (/etc/rc.d/init.d/*) | **/lib/systemd/system/*** |
| *Solaris Unix* | **/etc/init.d/*** | |
| *FreeBSD Unix* | **/etc/rc.d/*** | |
| *Solaris Unix* | **/lib/svc/manifest** | |

*** svaki servis je konfiguriran u zasebnoj datoteci**

Slijede konfiguracijske datoteke vezane za korisnike i korisničke grupe

| Definicija (konfiguracija) korisničkih računa | |
|---|---|
| *RedHat/CentOS/Fedora/Rocky/Oracle/SUSE* | **/etc/passwd** |
| *Debian/Ubuntu* | |
| *FreeBSD Unix* | |
| *Solaris Unix* | |

| Definicija lozinki (provjernih brojeva [*hash*]) i pravila vezanih za korisničke račune | |
|---|---|
| *RedHat/CentOS/Fedora/Rocky/Oracle/SUSE* | **/etc/shadow** |
| *Debian/Ubuntu* | |
| *FreeBSD Unix* | |
| *Solaris Unix* | |

| Definicija (konfiguracija) korisničkih grupa | |
|---|---|
| *RedHat/CentOS/Fedora/Rocky/Oracle/SUSE* | **/etc/group** |
| *Debian/Ubuntu* | |
| *FreeBSD Unix* | |
| *Solaris Unix* | |

Potom imamo konfiguracijske datoteke vezane za menadžere softverskih (programskih) paketa:

| Lokacija konfiguracijskih datoteka repozitorija | |
|---|---|
| *RedHat/CentOS/Fedora/Rocky/Oracle* | **/etc/yum.repos.d/*.repo** |
| *SUSE* | **/etc/zypp/repos.d/*.repo** |
| *Debian/Ubuntu* | **/etc/apt/sources.list.d/*.repo** |
| *FreeBSD Unix* | **/usr/local/etc/pkg/repos/*.conf** |
| *Solaris Unix* | **/var/share/pkg/repositories/*** |

*** svaki repozitorij je konfiguriran u zasebnoj datoteci**

Pogledajmo i listu naziva paketnih menadžera i naredbi ovisno o distribuciji Linuxa ili Unixa.

| Distribucija Linuxa ili Unixa | Naredbe paketnog menadžera |
|---|---|
| *RedHat/CentOS/Fedora/Rocky/Oracle Linux* | **rpm** [-i\|-U\|-e\|-q\|…]<br>**yum** [update\|install\|search\|…]<br>**dnf** [update\|install\|search\|…] |
| *Debian/Ubuntu Linux* | **dpkg** [--info\|--install\|--remove\|…]<br>**apt** [upgrade\|update\|install\|remove\|…]<br>**apt-get** [update\|upgrade\|install\|…]<br>**apt-cache** [search\|…]<br>. . . |
| *SuSe Linux* | **rpm** [-i\|-U\|-e\|-q\|…]<br>**zypper** [install\|remove\|search\|patch\|…]<br>**yast** → grafički/tekstualno grafički program |
| *FreeBSD Unix* | **pkg** [search\|update\|info\|install\|delete\|…] |
| *Solaris Unix* | **pkgadd** → za dodavanje paketa<br>**pkgchk** → za provjeru paketa<br>**pkginfo** → za informacije o paketu<br>**pkgrm** → za deinstalaciju paketa |

# 30.4. Popis programa otvorenog programskog kôda

Pogledajmo i tablicu sa značajnijim programima i sustavima otvorenog kôda, prema kategorijama.
Naveli smo samo neke od njih!

| Naziv | Opis | Naziv | Opis |
|---|---|---|---|
| **Operativni sustavi** | | **Baze podataka i imenični servisi** | |
| *Arch Linux* | *Distribucije Linuxa* | *ApacheDS* | *LDAP poslužitelj* |
| *CentOS Linux* | | *Couchbase* | *NoSQL poslužitelj/baza* |
| *Debian Linux* | | *MariaDB* | *SQL poslužitelj/baza* |
| *Fedora Linux* | | *MongoDB* | *NoSQL poslužitelj/baza* |
| *Gentoo Linux* | | *MySQL* | *SQL poslužitelj/baza* |
| *OpenSUSE Linux* | | *OpenLDAP* | *LDAP poslužitelj* |
| *Rocky Linux* | | *PostgreSQL* | *SQL poslužitelj/baza* |
| *Slackware Linux* | | *SQLite* | *SQL database engine* |
| *Ubuntu Linux* | | ***Verzioniranje kôda*** | |
| | | *Apache Subversion (SVN)* | *Verzioniranje programskog kôda* |
| *FreeBSD* | *UNIX* | *Git* | |
| *NetBSD* | | *Mercurial* | |
| *OpenBSD* | | ***Uredski alati*** | |
| *OpenIndiana* | *UNIX (OpenSolaris)* | *Dia* | *Crtanje dijagrama i shema* |
| **Mrežni servisi** | | *Emacs* | *Uređivač teksta* |
| ***Web i aplikacijski poslužitelji*** | | *FreeMind* | *Crtanje „mind mapping“ dijagrama* |
| *Apache* | *Web poslužitelj* | *GIMP* | *Obrada fotografija* |
| *Apache Tomcat* | *Java servlet kontejner* | *Inkscape* | *Vektorsko crtanje* |
| *Glassfish* | *Aplikacijski poslužitelj* | *Kodi* | *(Multi)medija centar* |
| *Jetty* | *Web posl. i servlet kontejner* | *LaTeX* | *Izrada dokumenata* |
| *JBoss* | *Aplikacijski poslužitelj* | *LibreOffice* | *Skup uredskih programa* |
| *Lighttpd* | *Web poslužitelj* | *Media Player Classic* | *Reproduktor videa* |
| *Nginx* | *Web poslužitelj* | *Notepad++* | *Uređivač teksta* |
| ***DHCP, DNS, FTP/TFTP poslužitelji*** | | *OBS Studio* | *Snimanje videa/zaslona* |
| *BIND* | *DNS poslužitelj* | *Scribus* | *Grafička priprema za tisak* |
| *Dnsmasq* | *DNS, DHCP, TFTP,FTP* | *SubtitleEdit* | *Uređivač titlova* |
| *Filezilla Server* | *FTP poslužitelj* | *TeXstudio* | *LaTeX grafički uređivač* |
| *NSD* | *DNS poslužitelj* | *VIM* | *Uređivač teksta* |
| *ProFTPD* | *FTP poslužitelj* | *VLC* | *Reproduktor multimedije* |
| *Unbound* | *DNS poslužitelj* | *WinMerge* | *Uspoređivanje sadržaja tekstualnih datoteka* |
| *Vsftpd* | *FTP poslužitelj* | ***Audio/Video obrada*** | |
| ***Drugi mrežni servisi i značajnije biblioteke*** | | *Audacity* | *Obrada zvuka* |
| *Dovecot* | *E-mail poslužitelj* | *Fre:AC* | *Audio konverter* |
| *Exim* | *E-mail poslužitelj* | *Avidemux* | *Video/audio konverter* |
| *LibreSSL* | *SSL/TLS biblioteke* | *FFmpeg* | |
| *OpenNTPD* | *NTP poslužitelj* | *HandBrake* | |
| *OpenSSH* | *SSH poslužitelj* | *OpenShot* | *Video uređivač* |
| *OpenSSL* | *SSL/TLS biblioteke* | ***Mrežni programi i klijenti*** | |
| *OpenVPN* | *VPN poslužitelj/klijent* | *Mozilla Firefox* | *Web preglednik* |
| *Postfix* | *E-mail poslužitelj* | *Mozilla Thunderbird* | *E-mail klijent* |
| *SAMBA* | *SMB/CIFS/AD poslužitelj* | *Evolution* | *E-mail klijent* |
| *Sendmail* | *E-mail poslužitelj* | *Filezilla* | *FTP/FTTPS/SFTP klijent* |
| *Squid proxy* | *Proxy poslužitelj* | *PuTTY* | *SSH/Telnet klijent* |
| ***CMS (content management system), e-commerce i drugi*** | | *WinSCP* | *FTP/FTTPS/SFTP klijent* |
| *Drupal* | *CMS* | *Wireshark* | *Dohvaćanje, prikaz i analiza mrežnih paketa* |
| *DokuWiki* | *Wikipedia* | *Xming* | *X Server (poslužitelj)* |
| *Joomla* | *CMS* | *Zenmap/nmap* | *Skeniranje mreže* |
| *Magento* | *e-commerce* | ***Upravljanje projektima*** | |
| *MediaWiki* | *Wikipedia* | *Bugzilla* | *Bug tracking i testing* |
| *OpenCart* | *e-commerce* | *Gerrit* | *Code collaboration* |
| *OpenCMS* | *CMS* | *Jenkins* | *CI (Continuous Integration)* |
| ***Sigurnost*** | | *OpenProject* | *Upravljanje projektima/bug tracking* |
| *ClamAV* | *Antivirusni program* | *Redmine* | |
| *Keepass* | *Password manager* | *Trac* | |
| *Suricata* | *IDS/IPS sustav* | | |
| *TrueCrypt* | *Kriptografski program* | | |

# 30.5. Popis često korištenih datotečnih sustava

Pogledajmo popis nekih od češće korištenih datotečnih sustava:

| Naziv | Maks. veličina particije | Maks. veličina datoteke | POSIX ovlasti | Copy on write (Snapshot) | Enkripcija (transparenta) | Sparse files | Journaling |
|---|---|---|---|---|---|---|---|
| APFS | - | *8 EB* | *DA* | *DA* | *DA* | *DA* | *DA* |
| BTRFS | ***16 EB*** | *16 EB* | *DA* | *DA* | *NE* | *DA* | *DA* |
| exFAT | ***128 PB*** | *128 PB* | *NE* | *NE* | *NE* | *NE* | *NE* |
| ext4 | ***1 EB*** | *16 TB* | *DA* | *NE* | *DA* | *DA* | *DA* |
| FAT 16B | ***4 GB*** | *4 GB* | *NE* | *NE* | *NE* | *NE* | *NE* |
| FAT 32 | ***16 TB*** | *4 GB* | *NE* | *NE* | *NE* | *NE* | *NE* |
| HPFS | ***2 TB*** | *2 GB* | *NE* | *NE* | *NE* | *NE* | *NE* |
| ISO 9660 (1988) | ***8 TB*** | *4 GB (Level 1i2)* *8 TB (Level 3)* | *NE* | *NE* | *NE* | *NE* | *NE* |
| NTFS | ***8 PB*** | *8 PB* | *DA* | *Indirektno* ***(shadow copy)*** | *DA* | *DA* | *DA* |
| ReFS | ***1 YB*** | *16 EB* | *DA* | *DA* | *DA* | *DA* | *DA* |
| UFS2 | ***512 ZB*** | *32 PB* | *DA* | *DA* | *NE* | *DA* | *DA* |
| XFS | ***8 EB*** | *8 EB* | *DA* | *NE* | *NE* | *DA* | *DA* |
| ZFS | **$2^{128}$ *bajta*** | *16 Eb* | *DA* | *DA* | *DA* | *DA* | *DA* |
| | | | | | | | |

*Volume manageri* i logička RAID polja:

| Naziv | *Snapshot* | *Thin provisioning* | *Podrška za RAID polja (0/1/5/10)* |
|---|---|---|---|
| BTRFS | *DA* | *NE* | *0 / 1 / 5 / 10* |
| (Microsoft) Logical Disk Manager | *DA* | *NE* | *0 / 1 / 5* |
| Linux Logical Volume Manager 2 (LVM) | *DA* | *DA* (LVM Thin) | *0 / 1 / 5 / 10* |
| Solaris Volume manager | *NE* | *NE* | *0 / 1 / 5 / 10* |
| Windows 8 Storage spaces | *DA* | *DA* | *1 / 5* |
| Windows 10 Storage spaces | *DA* | *DA* | *0 / 1 / 5 / 10* |
| ZFS | *DA* | *DA* | *0 / 1 / 5 / 6 / 10* |

**Klasterski redundatni sustavi za pohranu:**

| Naziv | *Replikacija /redundancija* | *Klaster* | *Vrsta: BLOK/objekt ili Datotečni* | *Licenca* |
|---|---|---|---|---|
| BeeGFS | *DA* | *DA* | *Datotečni* | ***Klijent*** *- GPL.v.2* ***Server*** *- vlasnička licenca -* |
| CEPH | *DA* | *DA* | *Blok i objekt* | *LGPL.v.2.1.* |
| CEPH FS | *DA* | *DA* | *Datotečni* | |
| CERN EOS | *DA* | *DA* | *Blok i objekt* | *Server Side Public License, v 1.* |
| DRBD | *DA* | *NE* | *Blok* | *GPL* |
| GFS2 | *Indirektno (pr. +DRBD)* | *DA* | *Datotečni* | *GPL* |
| GlusterFS | *DA* | *DA* | *Datotečni* | *GPL v.3* |
| Lustre | *DA* | *DA* | *Datotečni* | *GPL.v.2. i LGPL* |
| MooseFS | *DA* | *DA* | *Datotečni* | *GPL.v.2. + vlasnička licenca* |
| OCFS2 | *DA* | *DA* | *Datotečni* | *GPL* |

**Izvori informacija: (1408),(K-8).**

# 31. Izvori informacija

**Online izvori informacija:**


**(1) https://www.opensource-osijek.org/wordpress/2016/02/16/switching-i-routing-jucer-danas-sutra/**
**(2) https://en.wikipedia.org/wiki/Twisted_pair**
**(3) https://en.wikipedia.org/wiki/Category_5_cable**
**(4) https://en.wikipedia.org/wiki/Category_6_cable**
**(5) https://en.wikipedia.org/wiki/Medium-dependent_interface**
**(6) http://netoptimizer.blogspot.hr/2014/05/the-calculations-10gbits-wirespeed.html**
**(7) http://networkengineering.stackexchange.com/questions/5057/what-is-the-actual-size-of-an-ethernet-mtu**
**(8) http://www.cisco.com/c/dam/global/da_dk/assets/docs/presentations/CVU-juni_X_S-series.pdf**
→ **(Technical Deep Dive Catalyst 3750-X and Catalyst 2960-S)** *[zadnji put pristupano: 10.mj.2020]*
**(9) https://ciscointerworking.wordpress.com/2015/06/14/asic-redundancy/**
**(10) http://www.cisco.com/networkers/nw03/presos/docs/RST-2011.pdf**
**(11) https://fdocuments.net/reader/full/brkarc-3437** → **(Cisco Catalyst Switching architecture)** *[10.mj.2020]*
**(12) http://networking.ventrefamily.com/2010/08/asic-to-port-mappings.html**
**(13) https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000599-en.pdf**
→ **(Juniper QFX10000 Switches architecture)** *[10.mj.2020]*
**(14) http://www.cisco.com/assets/global/DK/seminarer/pdfs/Catalyst_Switching_Deep_Dive_Feb2016.pdf** *[10.mj.2020]*
**(15) https://calomel.org/freebsd_network_tuning.html**
**(16) http://kb.pert.geant.net/PERTKB/MultiThreadDemux**
**(17) https://github.com/gokzy/freebsd-rps/wiki/Receive-Packet-Steering-on-FreeBSD**
**(18) https://wiki.freebsd.org/201305DevSummit/NetworkReceivePerformance/ComparingMutiqueueSupportLinuxvsFreeBSD**
**(19) http://www.chelsio.com/wp-content/uploads/resources/T5-40Gb-FreeBSD-Netmap.pdf**
→ **(FreeBSD 40GbE Netmap Performance)** *[10.mj.2020]*
**(20) https://www.mjmwired.net/kernel/Documentation/networking/scaling.txt#308**
**(21) https://conferences.oreilly.com/oscon/oscon-tx/public/schedule/detail/56727**
**(22) https://lwn.net/Articles/382428/**
**(23) http://www.dell.com/downloads/global/power/1q04-her.pdf**
**(24) http://www.chelsio.com/wp-content/uploads/resources/T6-100G-DDP-FreeBSD.pdf**
→ **(Industry's First 100G Offload with FreeBSD)** *[10.mj.2020]*
**(25) http://www.chelsio.com/wp-content/uploads/resources/T5-40Gb-FreeBSD-TOE-DDP.pdf**
→ **(FreeBSD 40GbE TOE Performance)** *[10.mj.2020]*
**(26) https://www.opensource-osijek.org/dokuwiki/wiki:knjige:uvod_u_linux#numa**
**(27) https://calomel.org/network_performance.html**
**(28) https://en.wikipedia.org/wiki/PCI_Express**
**(29) http://luca.ntop.org/10g.pdf**
→ **(10 Gbit/s Line Rate Packet Processing Using Commodity Hardware: Survey and new Proposals)** *[10.mj.2020]*
**(30) https://pdfs.semanticscholar.org/cf0c/c4bdabc3e5f04221ee08ae31bc8714f2367d.pdf**
→ **(netmap: memory mapped access to network devices)** *[10.mj.2020]*
**(31) https://www.netgate.com/blog/building-a-behemoth-router.html**
**(32) http://dpdk.org/**
**(33) http://info.iet.unipi.it/~luigi/netmap/**
**(34) https://www.linux-kvm.org/images/c/c5/Kvm-forum-2013-High-Performance-IO-for-VMs.pdf**
**(35) https://pdos.csail.mit.edu/papers/click:tocs00/paper.pdf (The Click Modular Router)** *[10.mj.2020]*
**(36) http://openvswitch.org/support/ovscon2014/18/1630-ovs-rizzo-talk.pdf**
→ **(OVS: accelerating the datapath through netmap/VALE)** *[10.mj.2020]*
**(37) http://www.ntop.org/products/packet-capture/pf_ring/**
**(38) https://en.wikipedia.org/wiki/TCP_offload_engine#Freed-up_CPU_cycles**
**(39) http://www.nanogrids.org/jaidev/papers/ispass03.pdf** → **(TCP Performance Re-Visited)** *[10.mj.2020]*
**(40) https://lwn.net/Articles/551284/**
**(41) https://www.freebsd.org/cgi/man.cgi?query=polling&apropos=0&sektion=0&manpath=FreeBSD+11.0-stable&arch=default&format=html**
**(42) http://www.science.unitn.it/~fiorella/guidelinux/tlk/node86.html**
**(43) https://en.wikipedia.org/wiki/Interrupt_coalescing**
**(44) https://en.wikipedia.org/wiki/Ethernet**
**(45) https://en.wikipedia.org/wiki/Cut-through_switching**
**(46) https://en.wikipedia.org/wiki/Store_and_forward**
**(47) https://medium.com/speedtest-by-ookla/engineer-maximizes-internet-speed-story-c3ec0e86f37a**
**(48) https://www.pfsense.org/hardware/**
**(49) https://docs.opnsense.org/manual/hardware.html**
**(50) https://www.sophos.com/en-us/products/unified-threat-management/tech-specs.aspx#software**
**(51) http://www.brocade.com/en/products-services/software-networking/network-functions-virtualization/vrouter.html**
**(52) https://www.netgate.com//products/sg-1000.html**
**(53) http://www.androidauthority.com/arm-vs-x86-key-differences-explained-568718/**
**(53.1) https://www.firewallhardware.it/dimensionare_hardware_pfsense.html**
**(53.2) https://www.ntop.org/products/packet-capture/pf_ring/**
**(54) https://en.wikipedia.org/wiki/New_API**
**(55) https://software.intel.com/en-us/articles/how-intel-quickassist-technology-accelerates-nfv-use-cases**
**(55.1)http://ark.intel.com/products/77988/Intel-Atom-Processor-C2758-4M-Cache-2_40-GHz**
**(56) https://fd.io/news/announcement/2017/06/fast-data-fdio-project-issues-fourth-release-furthers-position-universal**
**(57) https://wondernetwork.com/pings/**
**(58) http://www.dell.com/en-us/work/shop/productdetails/force10-s2410**
**(59) http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5020-switch/white_paper_c11-465436.html**
**(60) https://en.wikipedia.org/wiki/IEEE_802.1Q**
**(61) https://wiki.wireshark.org/MTU**

**(62) https://en.wikipedia.org/wiki/Maximum_transmission_unit**
**(63) http://bradhedlund.com/2008/12/19/how-to-calculate-tcp-throughput-for-long-distance-links/**
**(64) https://www.opensource-osijek.org/dokuwiki/wiki:knjige:uvod_u_linux#ip_adrese**
**(65)http://en.wikipedia.org/wiki/List_of_Linux_adopters**
**(66)http://opensourceforamerica.org/category/case-studies/**
**(67)http://www.comparebusinessproducts.com/fyi/50-places-linux-running-you-might-not-expect**
**(68)http://www.unterstein.net/su/docs/CathBaz.pdf (The Cathedral and the Bazaar)** *[10.mj.2020]*
**(69)https://www.levenez.com/unix/**
**(70)https://en.wikipedia.org/wiki/Motherboard**
**(71)https://en.wikipedia.org/wiki/Central_processing_unit**
**(72)https://en.wikipedia.org/wiki/Computer_keyboard**
**(73)https://en.wikipedia.org/wiki/Code_page**
**(74)https://en.wikipedia.org/wiki/Code_page_437**
**(75)https://en.wikipedia.org/wiki/Real-time_clock**
**(76)https://en.wikipedia.org/wiki/System_time**
**(77)http://tldp.org/HOWTO/Clock-2.html**
**(78)https://en.wikipedia.org/wiki/Time_zone**
**(79)https://wiki.archlinux.org/index.php/locale**
**(80)http://www.cyberciti.biz/faq/how-to-set-locales-i18n-on-a-linux-unix/**
**(81)https://en.wikipedia.org/wiki/Interrupt_request_%28PC_architecture%29**
**(82)https://en.wikipedia.org/wiki/Advanced_Programmable_Interrupt_Controller**
**(83)https://en.wikipedia.org/wiki/Interrupt_handler**
**(84)https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-cpu-irq.html**
**(85)https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_MRG/1.3/html/Realtime_Tuning_Guide/sect-Realtime_Tuning_Guide-General_System_Tuning-Interrupt_and_Process_Binding.html**
**(86)https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Performance_Tuning_Guide/sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Performance_Monitoring_Tools-irqbalance.html**
**(87)https://en.wikipedia.org/wiki/Direct_memory_access**
**(88)https://en.wikipedia.org/wiki/Linux_kernel**
**(89)https://www.ibm.com/developerworks/library/l-linux-kernel/**
**(90) https://payberah.github.io/files/download/device-driver/arch.pdf → (Linux Kernel Architecture)** *[10.mj.2021]*
**(91)http://tldp.org/HOWTO/Module-HOWTO/**
**(92)https://en.wikipedia.org/wiki/Library_%28computing%29**
**(93)http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html**
**(94)http://www.tldp.org/LDP/GNU-Linux-Tools-Summary/html/x1712.htm**
**(95)https://en.wikipedia.org/wiki/Unix_shell**
**(96)https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29**
**(97)http://linux.about.com/library/cmd/blcmdl1_who.htm**
**(98)http://www.tldp.org/LDP/GNU-Linux-Tools-Summary/GNU-Linux-Tools-Summary.pdf**
**→ (GNU/Linux Command−Line Tools Summary)** *[10.mj.2021]*
**(99)https://en.wikipedia.org/wiki/Dmesg**
**(100)http://www.linfo.org/dmesg.html**
**(101)http://linux.die.net/man/8/dmidecode**
**(102)http://linux.die.net/man/8/lspci**
**(103)http://www.tldp.org/LDP/tlk/fs/filesystem.html**
**(104)http://www.tldp.org/LDP/lpg/node15.html**
**(104.1) https://www.howtoforge.com/linux-mkfifo-command/**
**(105)http://www.tldp.org/LDP/sag/html/dev-fs.html**
**(106)http://www.linfo.org/path_env_var.html**
**(107)https://www.linux.com/learn/understanding-linux-file-permissions**
**(108)https://wiki.archlinux.org/index.php/File_permissions_and_attributes**
**(109)http://www.cyberciti.biz/faq/how-to-use-chmod-and-chown-command/**
**(110)http://www.unixtutorial.org/2014/07/difference-between-chmod-and-chown/**
**(111)http://www.tldp.org/LDP/tlk/kernel/processes.html**
**(112)http://www.tldp.org/LDP/tlk/tlk-toc.html**
**(113)https://stackoverflow.com/questions/4087280/approximate-cost-to-access-various-caches-and-main-memory**
**(114)https://wiki.mikejung.biz/OS_Tuning#CPU_Overview**
**(115)https://wiki.mikejung.biz/OS_Tuning#Viewing_NUMA_stats_with_numastat**
**(116)http://www.admin-magazine.com/Archive/2014/20/Best-practices-for-KVM-on-NUMA-servers**
**(117)https://www.ibm.com/support/knowledgecenter/linuxonibm/liaai.hpctune/cpuandmemorybinding.htm**
**(118)http://www.glennklockwood.com/hpc-howtos/process-affinity.html**
**(119)https://www.sqlskills.com/blogs/jonathan/understanding-non-uniform-memory-accessarchitectures-numa/**
**(120)https://wiki.mikejung.biz/OS_Tuning#CPU_Overview**
**(121)https://wiki.mikejung.biz/OS_Tuning#Viewing_NUMA_stats_with_numastat**
**(122)http://www.admin-magazine.com/Archive/2014/20/Best-practices-for-KVM-on-NUMA-servers**
**(123)https://www.ibm.com/support/knowledgecenter/linuxonibm/liaai.hpctune/cpuandmemorybinding.htm**
**(124)http://www.glennklockwood.com/hpc-howtos/process-affinity.html**
**(125)https://www.kernel.org/doc/Documentation/vm/page_migration**
**(126)https://software.intel.com/en-us/articles/getting-high-performance-on-numa-based-nehalem-ex-system-with-mkl-without-controlling-numa**
**(127)http://h20564.www2.hpe.com/hpsc/doc/public/display?docId=emr_na-c03261871&lang=en-us&cc=us**
**(128)http://queue.acm.org/detail.cfm?id=2513149**
**(129)https://en.wikipedia.org/wiki/Unix_signal**
**(130)http://unix.stackexchange.com/questions/85364/how-can-i-check-what-signals-a-process-is-listening-to**
**(131)http://www.thegeekstuff.com/2012/03/linux-signals-fundamentals/**
**(132)http://www.tldp.org/LDP/abs/html/parameter-substitution.html#PSUB2**
**(133)http://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/subshells.html**
**(134)http://stackoverflow.com/questions/8748831/when-do-we-need-curly-braces-in-variables-using-bash**
**(135)http://linuxcommand.org/lc3_lts0080.php**

(136)http://unix.stackexchange.com/questions/4899/var-vs-var-and-to-quote-or-not-to-quote
(137)https://en.wikipedia.org/wiki/Memory_management_unit
(138)http://bashitout.com/2009/08/30/Linux-Compression-Comparison-GZIP-vs-BZIP2-vs-LZMA-vs-ZIP-vs-Compress.html
(139)https://administratosphere.wordpress.com/2008/05/23/sparse-files-what-why-and-how/
(140) https://hackaday.com/2017/12/28/34c3-hacking-into-a-cpus-microcode/
(141) https://en.wikipedia.org/wiki/Instruction_pipelining
(142) https://en.wikipedia.org/wiki/Machine_code
(143) https://en.m.wikipedia.org/wiki/Instruction_cycle
(144) https://cs.stackexchange.com/questions/21895/what-happens-at-the-decode-phase-of-the-instruction-cycle
(145)http://syssec.rub.de/media/emma/veroeffentlichungen/2017/08/16/usenix17-microcode.pdf
→ **(Reverse Engineering x86 Processor Microcode)** *[10.mj.2021]*
(146) https://www.pcsuggest.com/update-cpu-microcode-in-linux/
(147) https://www.debian.org/releases/stable/i386/ch02s02.html.en
(148) https://wiki.ubuntu.com/Kernel/Firmware
(149) https://wiki.centos.org/HowTos/Laptops/Wireless
(150) https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-koppe.pdf
→ **(Reverse Engineering x86 Processor Microcode)** *[10.mj.2021]*
(151) https://passlab.github.io/CSE564/resources/MicrocodeIntro_Matloff_Franklin04.pdf
→ **(Introduction to Microcoded Implementation of a CPU Architecture)**
(152) http://people.ee.duke.edu/~sorin/ece252/lectures/4.2-tomasulo.pdf → **(Register Renaming)** *[10.mj.2021]*
(153) https://www.kernel.org/doc/Documentation/x86/early-microcode.txt
(154) https://downloadcenter.intel.com/search?keyword=Linux*+Processor+Microcode+Data+File
(155) https://www.dcddcc.com/docs/2014_paper_microcode.pdf
→ **(Security Analysis of x86 Processor Microcode)** *[10.mj.2021]*
(156) http://inertiawar.com/microcode/
(157) https://www.kernel.org/doc/Documentation/sysctl/vm.txt
(158) https://www.thomas-krenn.com/en/wiki/Linux_Page_Cache_Basics
(159) https://en.wikipedia.org/wiki/Page_cache
(160) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-tunables
(161) https://docs.oracle.com/cd/NOSQL/html/AdminGuide/linuxcachepagetuning.html
(162) http://docs.gluster.org/en/latest/Administrator%20Guide/Linux%20Kernel%20Tuning/
(163) https://www.thomas-krenn.com/en/wiki/Configuring_IPMI_under_Linux_using_ipmitool
(164) https://discuss.pivotal.io/hc/en-us/articles/206396927-How-to-work-on-IPMI-and-IPMITOOL
(165) https://wiki.ncsa.illinois.edu/display/ITS/IPMI+configuration+from+within+the++CentOS+6%2C7+Operating+System
(166) https://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface
(167) https://en.wikipedia.org/wiki/Linux_distribution
(168) https://en.wikipedia.org/wiki/IBM_PC_compatible
(169) https://en.wikipedia.org/wiki/Computer_hardware
(170) https://en.wikipedia.org/wiki/ATX
(171) https://en.wikipedia.org/wiki/Motherboard
(172) https://en.wikipedia.org/wiki/Chipset
(173) https://en.wikipedia.org/wiki/Northbridge_(computing
(174) https://en.wikipedia.org/wiki/Southbridge_(computing
(175) https://en.wikipedia.org/wiki/Conventional_PCI
(176) https://en.wikipedia.org/wiki/PCI-X
(177) https://en.wikipedia.org/wiki/PCI_Express
(178) https://en.wikipedia.org/wiki/IBM_PC_keyboard
(179) https://en.wikipedia.org/wiki/Computer_keyboard
(180) https://en.wikipedia.org/wiki/Scancode
(181) https://en.wikipedia.org/wiki/Keyboard_technology
(182) https://en.wikipedia.org/wiki/ASCII
(183) https://en.wikipedia.org/wiki/Code_page
(184) https://en.wikipedia.org/wiki/Code_page_437
(185) https://en.wikipedia.org/wiki/ISO/IEC_8859-1
(186) https://en.wikipedia.org/wiki/Windows-1250
(187) https://en.wikipedia.org/wiki/ISO/IEC_8859-2
(188) https://en.wikipedia.org/wiki/UTF-8
(189) https://en.wikipedia.org/wiki/AltGr_key
(190) https://en.wikipedia.org/wiki/System_request
(191) https://en.wikipedia.org/wiki/Magic_SysRq_key
(192) https://en.wikipedia.org/wiki/Print_screen
(193) https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/PCI/msi-howto.rst
(193.1) http://tldp.org/LDP/tlk/dd/pci.html
(193.2) https://en.wikipedia.org/wiki/Message_Signaled_Interrupts
(194) https://events.static.linuxfound.org/sites/events/files/slides/Chaiken_ELCE2016.pdf
→ **(IRQs: the Hard, the Soft, the Threaded and the Preemptible)** *[10.mj.2021]*
(195) http://beyond-syntax.com/blog/2011/03/diving-into-linux-networking-i/
(196) https://www.safaribooksonline.com/library/view/understanding-the-linux/0596005652/ch04s07.html
(197) https://0xax.gitbooks.io/linux-insides/content/interrupts/interrupts-9.html
(198) https://www.kernel.org/doc/Documentation/kernel-per-CPU-kthreads.txt
(199) https://wiki.linuxfoundation.org/networking/napi
(200) https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#irqs
(201) https://www.systutorials.com/docs/linux/man/1-mpstat/
(202) https://elixir.free-electrons.com/linux/v4.1/source/Documentation/kernel-per-CPU-kthreads.txt
(203) https://en.wikipedia.org/wiki/Long-term_support
(204) https://wiki.debian.org/Modules
(205) https://www.linuxnix.com/using-lsmod-and-modprobe-to-work-with-modules-in-linux/
(206) https://www.cyberciti.biz/faq/add-remove-list-linux-kernel-modules/
(207) https://wiki.parabola.nu/Kernel_modules
(208) https://linux.die.net/man/5/modprobe.d

**(209) https://en.wikipedia.org/wiki/Modprobe**
**(210) https://docs.oracle.com/cd/E37670_01/E41138/html/ol_bootmodules.html**
**(211) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-persistent_module_loading**
**(212) https://linux.die.net/man/5/modules.dep**
**(213) https://www.computerhope.com/unix/depmod.htm**
**(214) http://elixir.free-electrons.com/linux/latest/source/Documentation/admin-guide/devices.txt**
**(215) https://www.kernel.org/doc/Documentation/iostats.txt**
**(216) https://www.cyberciti.biz/tips/linux-raid-increase-resync-rebuild-speed.html**
**(217) https://www.kernel.org/doc/Documentation/iostats.txt**
**(218) http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/proc.txt**
**(219) https://www.kernel.org/doc/Documentation/ABI/testing/dev-kmsg**
**(220) http://kothamasusatish.blogspot.hr/2013/03/dmesg-printk-and-kernel-log-buffer.html**
**(221) https://www.kernel.org/doc/Documentation/sysctl/kernel.txt**
**(222) https://unix.stackexchange.com/questions/195057/what-is-an-open-file-description/195164#195164**
**(223) http://man7.org/linux/man-pages/man2/lseek.2.html**
**(224) https://linux.die.net/man/8/lsof**
**(225) https://en.wikipedia.org/wiki/Bus_(computing)**
**(226) https://www.thegeekstuff.com/2011/11/strace-examples/**
**(227) https://www.tecmint.com/strace-commands-for-troubleshooting-and-debugging-linux/**
**(228) http://www.brendangregg.com/perf.html**
**(229) https://perf.wiki.kernel.org/index.php/Perf_examples**
**(230) https://events.static.linuxfound.org/sites/events/files/slides/ELCE%20-%20fighting%20latency.pdf**
→ **(Fighting latency - How to optimize your system using perf)** *[10.mj.2021]*
**(231) https://www.ibm.com/developerworks/library/l-analyzing-performance-perf-annotate-trs/**
**(232) http://vger.kernel.org/~acme/perf-collabsummit-2015.pdf** → **(Linux Perf Tools)** *[10.mj.2021]*
**(233) https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/PerfTools**
**(234) https://www.slideshare.net/brendangregg/kernel-recipes-2017-using-linux-perf-at-netflix?ref=http://www.brendangregg.com/perf.html**
**(235) http://man7.org/linux/man-pages/man1/ltrace.1.html**
**(236) https://www.go4expert.com/articles/ltrace-linux-debugging-utility-tutorial-t29095/**
**(237) https://www.thegeekstuff.com/2012/03/reverse-engineering-tools/?utm_source=feedburner**
**(238) http://cdn.ttgtmedia.com/searchEnterpriseLinux/downloads/Linux_Toolbox.pdf**
→ **(Debugging IPC with Shell Commands)** *[10.mj.2021]*
**(239) https://en.wikipedia.org/wiki/Unix_domain_socket**
**(240) https://en.wikipedia.org/wiki/Berkeley_sockets**
**(241) https://www.tldp.org/LDP/tlk/ipc/ipc.html**
**(242) https://en.wikipedia.org/wiki/Inode**
**(243) https://www.slashroot.in/inode-and-its-structure-linux**
**(244) http://www.linux-mag.com/id/8658/**
**(245) https://unix.stackexchange.com/questions/63081/what-are-and-directories**
**(246) https://stackoverflow.com/questions/1401359/understanding-linux-proc-id-maps**
**(247) https://linux.101hacks.com/unix/pmap/**
**(248) https://stackoverflow.com/questions/9828810/understanding-pmap-output**
**(249) https://www.tldp.org/LDP/tlk/mm/memory.html**
**(250) https://www.cyberciti.biz/faq/linux-command-to-see-major-minor-pagefaults/**
**(251) http://blog.scoutapp.com/articles/2015/04/10/understanding-page-faults-and-memory-swap-in-outs-when-should-you-worry**
**(252) https://www.quora.com/What-is-the-difference-between-minor-and-major-page-fault-in-Linux**
**(253) http://www.linuxjournal.com/article/4681**
**(254) https://linux.101hacks.com/unix/anacron/**
**(255) https://www.digitalocean.com/community/tutorials/how-to-schedule-routine-tasks-with-cron-and-anacron-on-a-vps**
**(256) https://www.tecmint.com/cron-vs-anacron-schedule-jobs-using-anacron-on-linux/**
**(257) https://docs.oracle.com/cd/E37670_01/E41138/html/ch08s03.html**
**(258) https://en.wikipedia.org/wiki/Anacron**
**(259) https://community.mellanox.com/docs/DOC-2496**
**(260) https://fasterdata.es.net/assets/Papers-and-Publications/100G-Tuning-TechEx2016.tierney.pdf**
→ **(Recent Linux TCP Updates, and how to tune your 100G host)** *[10.mj.2021]*
**(261) https://codywu2010.wordpress.com/2015/11/26/pci-express-max-read-request-max-payload-size-and-why-you-care/**
**(262) https://www.linuxplumbersconf.org/2017/ocw/system/presentations/4737/original/mps.pdf**
→ **(Maximum Payload Size (MPS) vs. Maximum Read Request Size (MRS))** *[10.mj.2021]*
**(263) http://billauer.co.il/blog/2011/05/pcie-pci-express-linux-max-payload-size-configuration-capabilities-tlp-lspci/**
**(264) https://www.slideshare.net/datacenters/enabling-high-performance-bulk-data-transfers-with-ssh**
**(265) https://www.spirentfederal.com/documents/tcp_network-latency-and-throughput_whitepaper.pdf**
→ **(TCP - Network Latency and Throughput - Whitepaper)** *[01.mj.2019]*
**(266) https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Flow_control**
**(267) https://tools.ietf.org/html/draft-ietf-ippm-tcp-throughput-tm-05**
**(268) https://en.wikipedia.org/wiki/TCP_tuning#Window_size**
**(269) http://www.firewall.cx/networking-topics/protocols/tcp/137-tcp-window-size-checksum.html**
**(270) https://en.wikipedia.org/wiki/TCP_window_scale_option**
**(271) http://packetlife.net/blog/2010/aug/4/tcp-windows-and-window-scaling/**
**(272) http://packetbomb.com/understanding-throughput-and-tcp-windows/**
**(273) https://blog.thousandeyes.com/a-very-simple-model-for-tcp-throughput/**
**(274) http://sgros.blogspot.hr/2012/12/controlling-which-congestion-control.html**
**(275) https://linuxgazette.net/135/pfeiffer.html**
**(276) http://interstream.com/node/1084**
**(277) https://arxiv.org/pdf/1610.03534.pdf**
→ **(Comparative study of High-speed Linux TCP Variants over High-BDP Networks)** *[10.mj.2021]*
**(278) https://www.theseus.fi/bitstream/handle/10024/60091/Li%20Jie-thesis.pdf?sequence=1&isAllowed=y**
→ **(Performance Evaluation of Different TCP Congestion Control Schemes in 4G System)** *[10.mj.2021]*

**(279)** https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf
→ **(CUBIC: A New TCP-Friendly High-Speed TCP Variant)** *[10.mj.2021]*
**(280)** https://wiki.linuxfoundation.org/networking/netem
**(281)** https://fasterdata.es.net/host-tuning/linux/
**(282)** https://linux.die.net/man/1/join
**(283)** https://www.howtoforge.com/tutorial/linux-join-command/
**(284)** https://en.wikipedia.org/wiki/Join_(Unix)
**(285)** https://www.howtoforge.com/linux-paste-command/
**(286)** https://en.wikipedia.org/wiki/Paste_(Unix)
**(287)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-configuring_yum_and_yum_repositories
**(288)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-setting_repository_options
**(289)** https://en.wikipedia.org/wiki/Rsync
**(290)** https://wiki.archlinux.org/index.php/rsync
**(291)** https://www.thegeekstuff.com/2010/09/rsync-command-examples/
**(292)** https://www.fastwebhost.in/blog/13-rsync-command-examples-on-linux/
**(293)** https://en.wikipedia.org/wiki/Signal_(IPC)
**(294)** https://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html?m=1
**(295)** https://en.m.wikipedia.org/wiki/Process_group
**(296)** https://www.safaribooksonline.com/library/view/linux-system-programming/0596009585/ch05s06.html
**(297)** https://www.webhostinghero.com/how-to-create-a-process-group-in-linux/
**(298)** https://stackoverflow.com/questions/392022/whats-the-best-way-to-send-a-signal-to-all-members-of-a-process-group?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
**(299)** https://unix.stackexchange.com/questions/404054/how-is-a-process-group-id-set?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
**(300)** https://en.wikipedia.org/wiki/Character_encoding
**(301)** http://kunststube.net/encoding/
**(302)** https://en.wikipedia.org/wiki/Newline
**(303)** https://en.wikipedia.org/wiki/Carriage_return
**(304)** https://stackoverflow.com/questions/4087280/approximate-cost-to-access-various-caches-and-main-memory
**(305)** https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf
→ **(Performance Analysis Guide for Intel® Core™ i7 Processor and Intel® Xeon™ 5500 processors)** *[10.mj.2021]*
**(306)** https://www.readex.eu/wp-content/uploads/2017/06/ICPE2017_authors_version.pdf
→ **(Detecting Memory-Boundedness with Hardware Performance Counters)** *[10.mj.2021]*
**(307)** ftp://ftp.hp.com/pub/c-products/servers/options/Memory-Config-Recommendations-for-Intel-Xeon-5500-Series-Servers-Rev1.pdf
→ **(Memory Configuration Recommendations for Intel Xeon 5500 Series Servers)** *[01.mj.2019]*
**(308)** https://people.freedesktop.org/~narmstrong/meson_drm_doc/admin-guide/ras.html
**(309)** https://en.wikipedia.org/wiki/Registered_memory
**(310)** https://en.wikipedia.org/wiki/Fully_Buffered_DIMM
**(311)** http://fibrevillage.com/sysadmin/240-how-to-identify-defective-dimm-from-edac-error-on-linux-2
**(312)** https://metebalci.com/blog/a-minimum-complete-tutorial-of-linux-ext4-file-system/
**(313)** https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout
**(314)** https://en.wikipedia.org/wiki/Ext4
**(315)** https://opensource.com/article/17/5/introduction-ext4-filesystem
**(316)** https://en.wikipedia.org/wiki/Linux_namespaces
**(317)** https://lwn.net/Articles/531114/
**(318)** https://lwn.net/Articles/531381/
**(319)** https://blogs.igalia.com/dpino/2016/04/10/network-namespaces/
**(320)** https://wiki.archlinux.org/index.php/advanced_traffic_control
**(321)** https://www.coverfire.com/articles/queueing-in-the-linux-network-stack/
**(321.1)** https://www.cubrid.org/blog/understanding-tcp-ip-network-stack
**(321.2)** https://www.cs.dartmouth.edu/~sergey/me/netreads/path-of-packet/Lab9_modified.pdf
→ **(The Journey of a Packet Through the Linux Network Stack)** *[10.mj.2021]*
**(322)** https://lwn.net/Articles/616241/
**(323)** https://fasterdata.es.net/host-tuning/linux/
**(324)** http://assimilationsystems.com/2015/12/29/bufferbloat-network-best-practice/
**(325)** https://www.kernel.org/doc/Documentation/sysctl/net.txt
**(326)** http://courses.washington.edu/ee461/handouts/TCP_Slow_Start.pdf
→ **(CP - Transmission Control Protocol (TCP Slow Start))** *[10.mj.2021]*
**(327)** http://www.eecs.umich.edu/courses/eecs489/w07/LectureSlides/lec8_tcp.pdf
→ **(Electrical Engineering and Computer Science (EECS), University of Michigan: Lecture TCP)** *[01.mj.2019]*
**(328)** https://blog.stackpath.com/glossary/cwnd-and-rwnd/
**(329)** https://www.utdallas.edu/~venky/acn/CongestionControl.pdf
→ **(The university of Texas at Dallas: Congestion Control)** *[01.mj.2019]*
**(330)** https://www.eventhelix.com/RealtimeMantra/Networking/TCP_Fast_Retransmit_and_Recovery.pdf
→ **(Server_Socket Interfaces (TCP Fast Retransmit and Recovery))** *[10.mj.2021]*
**(331)** https://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_WANIDA/DR/JavisInActionFastRecoveryFrame.html
**(332)** https://pdfs.semanticscholar.org/40ee/cb0ad2213fcfcb9a841f43a2bdf602369a05.pdf
→ **(TCP fast recovery strategies: analysis and improvements)** *[10.mj.2021]*
**(333)** https://en.wikipedia.org/wiki/Explicit_Congestion_Notification
**(334)** http://blog.catchpoint.com/2015/10/30/tcp-flags-cwr-ece/
**(335)** http://man7.org/linux/man-pages/man7/tcp.7.html
**(336)** https://superuser.com/questions/240456/how-to-interpret-the-output-of-netstat-o-netstat-timers
**(337)** https://www.wisdomjobs.com/e-university/linux-tutorial-277/tcp-timers-1075.html
**(338)** https://serverfault.com/questions/329845/how-to-forcibly-close-a-socket-in-time-wait
**(339)** http://www.lognormal.com/blog/2012/09/27/linux-tcpip-tuning/
**(340)** https://www.frozentux.net/ipsysctl-tutorial/chunkyhtml/tcpvariables.html
**(341)** https://www.kernel.org/doc/Documentation/sysctl/net.txt

(342) https://access.redhat.com/sites/default/files/attachments/20150325_network_performance_tuning.pdf
→ **(Red Hat Enterprise Linux Network Performance Tuning Guide)** *[10.mj.2021]*
(343) http://fibrevillage.com/sysadmin/91-tcp-performance-tuning-10g-nic-with-high-rtt-in-linux
(344) https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#monitoring-network-data-processing
(345) http://veithen.github.io/2014/01/01/how-tcp-backlog-works-in-linux.html
(346) https://medium.com/@pawilon/tuning-your-linux-kernel-and-haproxy-instance-for-high-loads-1a2105ea553e
(347) https://github.com/ton31337/tools/wiki/Is-net.ipv4.tcp_abort_on_overflow-good-or-not%3F
(348) https://support.hpe.com/hpsc/doc/public/display?docId=mmr_kc-0125158
(349) https://kb.informatica.com/faq/5/Pages/80204.aspx
(350) https://www.ibm.com/support/knowledgecenter/en/SSQPD3_2.6.0/com.ibm.wllm.doc/usingethtoolrates.html
(351) http://processors.wiki.ti.com/index.php/Linux_Core_CPSW_User%27s_Guide
(352) http://www.cs.unc.edu/~qianwen/Rapid_test/rapid_v0.9.2/turn_off_IC.html
(353) https://blog.cloudflare.com/how-to-achieve-low-latency/
(354) https://www.intel.com/content/dam/doc/application-note/82575-82576-82598-82599-ethernet-controllers-latency-appl-note.pdf
→ **(Improving Measured Latency in Linux for Intel® 82575/82576 or X540/82598/82599 Ethernet Controllers)** *[01.mj.2018]*
(355) https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/
(356) https://www.utdallas.edu/~cxl137330/paper/ACMSE08.pdf
→ **(Heavyweight or Lightweight: A Process Selection Guide for Developing Grid Software)** *[10.mj.2021]*
(357) https://medium.com/hungys-blog/linux-kernel-process-scheduling-8ce05939fabd
(358) https://tampub.uta.fi/bitstream/handle/10024/96864/GRADU-1428493916.pdf
→ **(A complete guide to Linux process scheduling)** *[01.mj.2018]*
(359) http://www.it.uu.se/education/course/homepage/os/vt18/module-4/implementing-threads/
(360) https://unix.stackexchange.com/questions/472324/what-are-the-relations-between-processes-kernel-threads-lightweight-processes
(361) https://medium.com/cracking-the-data-science-interview/the-10-operating-system-concepts-software-developers-need-to-remember-480d0734d710
(362) https://www.geeksforgeeks.org/maximum-number-threads-can-created-within-process-c/
(363) https://www.broadcom.com/support/knowledgebase/1211161499716/configuring-smp-affinity-in-linux
(364) https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA4-9294ENW-2013.pdf
*[01.mj.2018]*
(365) https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/
(365.1) https://www.kernel.org/doc/Documentation/networking/scaling.txt
(366) https://stackoverflow.com/questions/23730268/ixgbe-setting-the-number-of-rx-tx-queues
(367) https://null.53bits.co.uk/index.php?page=numa-and-queue-affinity
(368) https://sourceforge.net/p/e1000/mailman/message/36133196/
(369) https://events.static.linuxfound.org/sites/events/files/slides/LinuxConJapan2016_makita_160714.pdf
→ **(Boost UDP Transaction Performance)** *[01.mj.2021]*
(370) https://blog.cloudflare.com/how-to-achieve-low-latency/
(371) https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/
(372) https://www.mellanox.com/related-docs/prod_software/Performance_Tuning_Guide_for_Mellanox_Network_Adapters_Archive.pdf
→ **(Performance Tuning Guidelines for Mellanox Network Adapters)** *[10.mj.2021]*
(373) https://www.es.net/assets/Uploads/100G-Tuning-TechEx2016.tierney.pdf
→ **(Recent Linux TCP Updates, and how to tune your 100G host)** *[10.mj.2021]*
(374) https://events.static.linuxfound.org/sites/events/files/slides/scsi.pdf
→ **(High Performance Storage with blk-mq and scsi-mq)** *[10.mj.2021]*
(375) https://www.kernel.org/doc/Documentation/scsi/hpsa.txt
(376) http://kernel.dk/blk-mq.pdf
→ **(Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems)** *[10.mj.2021]*
(377) https://www.thomas-krenn.com/en/wiki/Linux_Multi-Queue_Block_IO_Queueing_Mechanism_(blk-mq)
(378) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.2_release_notes/storage
(379) https://www.thegeekdiary.com/what-is-hba-queue-depth-and-how-to-check-the-current-queue-depth-value-and-how-to-change-it/
(380) https://www.unixarena.com/2017/12/linux-scsi-device-management-identifying-devices.html/
(381) https://lwn.net/Articles/552904/
(382) https://hyunyoung2.github.io/2016/09/14/Multi_Queue/
(383) http://ari-ava.blogspot.com/2014/07/opw-linux-block-io-layer-part-4-multi.html
(384) https://miuv.blog/2017/10/21/linux-block-mq-simple-walkthrough/
(385) https://wiki.archlinux.org/index.php/VLAN
(386) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-configure_802_1q_vlan_tagging_using_the_command_line
(387) https://www.cyberciti.biz/tips/howto-configure-linux-virtual-local-area-network-vlan.html
(388) https://wiki.archlinux.org/index.php/VLAN
(389) https://en.wikipedia.org/wiki/Virtual_LAN
(390) https://en.wikipedia.org/wiki/IEEE_802.1Q
(391) https://forum.manjaro.org/t/checking-blk-mq-is-active/1484/6
(392) http://www.linux-kvm.org/images/f/f6/01x07a-Vhost.pdf → **(KVM I/O performance and end-to-end reliability)** *[10.mj.2021]*
(393) https://mahmoudhatem.wordpress.com/2016/02/08/oracle-uek-4-where-is-my-io-scheduler-none-multi-queue-model-blk-mq/
(394) https://www.reddit.com/r/Fedora/comments/66snp3/fedora_25_cant_enable_blk_mq/
(395) https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=320ae51feed5c2f13664aa05a76bec198967e04d
(396) https://fasterdata.es.net/host-tuning/100g-tuning/interrupt-binding/
(397) https://www.privateinternetaccess.com/blog/2016/01/linux-networking-stack-from-the-ground-up-part-4/
(398) https://github.com/torvalds/linux/blob/v3.13/Documentation/networking/scaling.txt
(399) https://github.com/torvalds/linux/blob/v3.13/Documentation/networking/scaling.txt#L138-L164
(400) https://en.wikipedia.org/wiki/Interrupt#RPS
(401) https://greenhost.net/2013/04/10/multi-queue-network-interfaces-with-smp-on-linux/
(402) https://docs.cloud.oracle.com/iaas/Content/Compute/References/updatingkernel.htm
(402.1) https://software.intel.com/en-us/articles/setting-up-intel-ethernet-flow-director
(403) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch-kernel#s1-kernel-packages
(404) http://elrepo.org/tiki/kernel-ml
(405) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch-sysconfig#
(406) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/ch-the_sysconfig_directory
(407) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s1-networkscripts-interfaces

**(408)** https://www.centos.org/docs/5/html/Deployment_Guide-en-US/s1-networkscripts-control.html
**(409)** https://developer.rackspace.com/blog/lacp-bonding-and-linux-configuration/
**(410)** https://en.wikipedia.org/wiki/Link_aggregation#Link_Aggregation_Control_Protocol
**(411)** https://backdrift.org/lacp-configure-network-bonding-linux
**(412)** https://www.thomas-krenn.com/en/wiki/Link_Aggregation_and_LACP_basics
**(413)** https://wiki.linuxfoundation.org/networking/bonding
**(414)** https://www.centos.org/docs/5/html/5.1/Deployment_Guide/s3-modules-bonding-directives.html
**(415)** https://www.kernel.org/doc/Documentation/networking/bonding.txt
**(416)** https://en.wikipedia.org/wiki/Link_aggregation
**(417)** https://wiki.debian.org/Bonding
**(418)** https://www.cyberciti.biz/tips/linux-bond-or-team-multiple-network-interfaces-nic-into-single-interface.html
**(419)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-network_bonding_using_the_command_line_interface
**(420)** http://www.mustbegeek.com/configure-lacp-etherchannel-in-cisco-ios-switch/
**(421)** https://www.freedesktop.org/software/systemd/man/systemd.service.html
**(422)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/sect-managing_services_with_systemd-unit_files
**(423)** https://fedoramagazine.org/systemd-getting-a-grip-on-units/
**(424)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/desktop_migration_and_administration_guide/logind
**(425)** https://wiki.archlinux.org/index.php/systemd-networkd
**(426)** https://www.digitalocean.com/community/tutorials/systemd-essentials-working-with-services-units-and-the-journal
**(427)** https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files
**(428)** https://www.systutorials.com/docs/linux/man/5-systemd.unit/
**(429)** https://www.linuxnix.com/introducing-systemd/
**(430)** https://en.wikipedia.org/wiki/Systemd
**(431)** https://askubuntu.com/questions/911525/difference-between-systemctl-init-d-and-service
**(432)** https://www.tecmint.com/systemd-replaces-init-in-linux/
**(433)** https://www.thegeekdiary.com/centos-rhel-7-how-to-set-datetime-ntp-and-timezone-using-timedatectl/
**(434)** https://www.digitalocean.com/community/tutorials/how-to-use-journalctl-to-view-and-manipulate-systemd-logs
**(435)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-configuring_the_date_and_time
**(436)** https://www.linuxnix.com/introducing-systemd/
**(437)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec_configuring_host_names_using_hostnamectl
**(438)** https://techblog.smashing.net/2012/11/21/beware-the-ntp-false-ticker-or-do-the-time-warp-again/
**(439)** https://serverfault.com/questions/591325/both-my-ntp-servers-are-marked-as-falsetickers-in-the-status
**(440)** https://en.wikipedia.org/wiki/Network_Time_Protocol
**(441)** https://manpages.debian.org/testing/ntp/ntpq.1.en.html
**(442)** https://www.linuxquestions.org/questions/linux-server-73/ntp-offset-4175461130/
**(443)** https://www.linuxjournal.com/article/6812
**(444)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s1-checking_the_status_of_ntp
**(445)** https://serverfault.com/questions/95342/ntp-is-running-system-clock-still-not-in-time-what-gives
**(446)** https://www.tecmint.com/install-ntp-server-in-centos/
**(447)** https://www.thegeekstuff.com/2014/06/linux-ntp-server-client
**(448)** https://serverfault.com/questions/768280/what-is-ntp-dispersion-and-how-do-i-control-it
**(449)** https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-58/154-ntp.html
**(450)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/s1-configure_ntp
**(451)** https://en.wikipedia.org/wiki/Network_File_System
**(452)** https://wiki.archlinux.org/index.php/NFS
**(453)** https://www.tecmint.com/how-to-setup-nfs-server-in-linux/
**(454)** https://www.centos.org/docs/5/html/Deployment_Guide-en-US/s1-nfs-client-config-options.html
**(455)** https://www.slashroot.in/how-do-linux-nfs-performance-tuning-and-optimization
**(456)** http://www.admin-magazine.com/HPC/Articles/Useful-NFS-Options-for-Tuning-and-Management
**(457)** https://www.cyberciti.biz/faq/mount-nfs4-reason-given-by-server-no-such-file-directory/
**(458)** https://www.unixtutorial.org/atime-ctime-mtime-in-unix-filesystems/
**(459)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/network-rfs
**(460)** http://mvapich.cse.ohio-state.edu/static/media/publications/abstract/vaidyana-ispass07.pdf
→ **(Benefits of I/O Acceleration Technology (I/OAT) in Clusters)** *[10.mj.2021]*
**(461)** http://timetobleed.com/enabling-bios-options-on-a-live-server-with-no-rebooting/
**(462)** https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/#intels-io-acceleration-technology-ioat
**(463)** https://en.wikipedia.org/wiki/I/O_Acceleration_Technology
**(464)** https://cateee.net/lkddb/web-lkddb/INTEL_IOATDMA.html
**(465)** https://blog.linuxplumbersconf.org/2015/ocw/system/presentations/2799/original/pushing-kernel-networking.pdf
→ **(Pushing Kernel networking)** *[10.mj.2018]*
**(466)** https://lwn.net/Articles/412062/
**(467)** https://en.wikipedia.org/wiki/Ethernet_flow_control
**(468)** http://www.dell.com/content/topics/global.aspx/power/en/ps1q01_hernan?c=us&l=en&cs=04
**(469)** https://opencores.org/websvn/filedetails?repname=1000base-x&path=%2F1000base-x%2Ftrunk%2Fdoc%2F802.3-2008_section3.pdf&bcsi_scan_a0908cff39a19e84=d8oKHwBeNe1MKUlLXKyChVr0O%2FU8AAAAoI%2FZPA%3D%3D
→ **(IEEE Std 802.3-2008, Section 3)** *[10.mj.2018]*
**(470)** https://blog.ine.com/2008/07/08/802-3x-flow-control
**(471)** https://help.ubuntu.com/community/UbuntuLTSP/FlowControl
**(472)** http://virtualthreads.blogspot.com/2006/02/beware-ethernet-flow-control.html
**(473)** https://community.mellanox.com/docs/DOC-2814
**(474)** https://en.wikipedia.org/wiki/Large_receive_offload
**(475)** https://community.mellanox.com/docs/DOC-2814#jive_content_id_LRO_on_New_Kernels
**(476)** https://en.wikipedia.org/wiki/Swappiness

(477) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-tunables
(478) https://www.kernel.org/doc/Documentation/sysctl/vm.txt
(479) https://www.kernel.org/doc/Documentation/vm/overcommit-accounting
(480) https://en.wikipedia.org/wiki/Swappiness
(481) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-tunables
(482) https://www.oracle.com/technetwork/articles/servers-storage-dev/oom-killer-1911807.html
(483) https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_MRG/1.3/html/Realtime_Tuning_Guide/sect-Realtime_Tuning_Guide-General_System_Tuning-Swapping_and_Out_Of_Memory_Tips.html
(484) https://lwn.net/Articles/317814/
(485) http://engineering.pivotal.io/post/virtual_memory_settings_in_linux_-_the_problem_with_overcommit/
(486) https://www.hawatel.com/blog/kernel-out-of-memory-kill-process/
(487) https://lwn.net/Articles/28345/
(488) https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-panic-params.html
(489) http://fibrevillage.com/storage/61-systool-a-useful-tool-for-san-as-well-for-sysfs-devices
(490) http://www.theunixway.com/2013/11/how-to-find-hba-and-add-new-lun-in.html
(491) https://www.kernel.org/doc/Documentation/sysctl/vm.txt
(492) https://access.redhat.com/solutions/90883
(493) https://queue.acm.org/detail.cfm?id=2513149
(494) https://www.kernel.org/doc/gorman/html/understand/understand005.html
(495) https://community.mellanox.com/docs/DOC-2532
(496) https://observersupport.viavisolutions.com/html_doc/current/index.html#page/observer/expert_explanations_ethernet.html
(497) http://docs-legacy.fortinet.com/fos40hlp/43/wwhelp/wwhimpl/common/html/wwhelp.htm?context=fgt&file=troubleshooting_tools.41.9.html
(498) https://www.cyberciti.biz/faq/howto-linux-display-protocol-level-statistics/
(499) https://support.lenovo.com/hr/en/solutions/ht118072
(500) https://lwn.net/Articles/358910/
(501) https://en.wikipedia.org/wiki/Large_send_offload
(502) https://wiki.geant.org/pages/releaseview.action?pageId=103712345
(503) https://lwn.net/Articles/564978/
(504) http://kris.io/2015/10/01/kvm-network-performance-tso-and-gso-turn-it-off/
(505) https://sandilands.info/sgordon/segmentation-offloading-with-wireshark-and-ethtool
(506) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/network-nic-offloads
(507) https://www.kernel.org/doc/Documentation/networking/checksum-offloads.txt
(508) https://www.netdevconf.org/1.1/proceedings/slides/miller-hardware-checksumming.pdf
→ **(Hardware Checksumming)** *[10.mj.2021]*
(509) https://forums.he.net/index.php?topic=2826.0
(510) https://serverfault.com/questions/564945/linux-cannot-change-tx-checksumming
(511) http://docs.gz.ro/node/282
(512) https://stackoverflow.com/questions/4212817/skb-and-scatter-gather-feature?rq=1
(513) https://lwn.net/Articles/256368/
(514) https://www.oreilly.com/library/view/linux-system-programming/9781449341527/ch04.html
(515) https://blogs.gnome.org/markmc/2008/05/28/checksums-scatter-gather-io-and-segmentation-offload/
(516) https://sokratisg.net/2012/04/01/udp-tcp-checksum-errors-from-tcpdump-nic-hardware-offloading/
(517) https://cromwell-intl.com/open-source/performance-tuning/ethernet.html
(518) https://www.cyberciti.biz/faq/rhel-centos-debian-ubuntu-jumbo-frames-configuration/
(519) https://www.netadmintools.com/art416.html
(520) http://www.tldp.org/LDP/tlk/mm/memory.html
(521) http://people.redhat.com/nmurray/RHEL-2.1-VM-whitepaper.pdf
→ **(Virtual Memory Behavior in Red Hat Linux Advanced Server 2.1)** *[10.mj.2021]*
(522) https://lwn.net/Articles/384150/
(523) https://lonesysadmin.net/2013/12/22/better-linux-disk-caching-performance-vm-dirty_ratio/
(524) https://stackoverflow.com/questions/4984190/understanding-proc-sys-vm-lowmem-reserve-ratio
(525) https://en.wikipedia.org/wiki/Jumbo_frame
(526) https://linuxconfig.org/how-to-enable-jumbo-frames-in-linux
(527) http://www.geekprojects.org/2015/07/increase-centos-7s-mtu/
(528) https://www.smallnetbuilder.com/lanwan/lanwan-features/30201-need-to-know-jumbo-frames-in-small-networks?start=3
(529) https://en.wikipedia.org/wiki/Executable_space_protection
(530) https://eklitzke.org/memory-protection-and-aslr
(531) https://samsclass.info/127/proj/lbuf1.htm
(532) http://www.infosecisland.com/blogview/8211-Protecting-Linux-Against-Overflow-Exploits.html
(533) https://www.usenix.org/legacy/events/sec08/tech/full_papers/dalton/dalton_html/dalton_html.html
(534) http://www.madhur.co.in/blog/2011/08/06/protbufferoverflow.html
(535) https://docs.oracle.com/cd/E37670_01/E36387/html/ol_aslr_sec.html
(536) https://www.thegeekdiary.com/centos-rhel-7-how-to-modify-the-kernel-command-line/
(537) https://vincent.bernat.ch/en/blog/2011-ipv4-route-cache-linux
(538) https://linoxide.com/how-tos/how-to-flush-routing-table-from-cache/
(539) https://vincent.bernat.ch/en/blog/2011-ipv4-route-cache-linux
(540) http://linux-ip.net/html/routing-cache.html
(541) http://linux-ip.net/html/tools-ip-route.html
(542) https://unix.stackexchange.com/questions/207535/routing-cache-in-latest-linux-kernels
(543) ftp://ftp.wayne.edu/ldp/en/linux-ip/ch04s08.html#routing-table-local
(544)ftp://ftp.wayne.edu/ldp/en/linux-ip/ch04s08.html
(545)https://lwn.net/Articles/145406/
(546)https://www.kernel.org/doc/html/v4.12/admin-guide/kernel-parameters.html
(547)http://www.embeddedlinux.org.cn/linux_net/0596002556/understandlni-CHP-33.html#understandlni-CHP-33
(548) ftp://ftp.wayne.edu/ldp/en/Adv-Routing-HOWTO/Adv-Routing-HOWTO.pdf
→ **(Linux Advanced Routing & Traffic Control HOWTO)** *[10.mj.2021]*
(549) https://www.kernel.org/doc/Documentation/networking/policy-routing.txt
(550) http://www.system-rescue-cd.org/networking/Advanced-networking-and-policy-routing/
(551) https://blog.scottlowe.org/2013/05/29/a-quick-introduction-to-linux-policy-routing/

(552) http://linux-ip.net/html/routing-rpdb.html
(553) http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.fwmark.html
(554) https://unix.stackexchange.com/questions/467076/how-set-mark-option-works-on-netfilter-iptables
(555) http://stud.netgroup.uniroma2.it/cgrl/2018/slides/6-pbr.pdf → (***Policy-Based Routing***) *[07.mj.2019]*
(556) https://wiki.nftables.org/wiki-nftables/index.php/Main_Page
(557) https://en.wikipedia.org/wiki/Nftables
(558) https://wiki.archlinux.org/index.php/nftables
(559) https://lwn.net/Articles/720675/
(560) https://www.phoronix.com/scan.php?page=article&item=linux-412-io&num=3
(561) https://www.phoronix.com/scan.php?page=news_item&px=Linux-4.12-BFQ-Kyber
(562) https://wiki.centos.org/HowTos/I_need_the_Kernel_Source
(563) https://kompjuteras.com/en/how-to-install-the-latest-kernel-from-source-on-centos-7/
(564) https://www.tecmint.com/compile-linux-kernel-on-centos-7/
(565) https://www.quora.com/What-is-the-purpose-of-Module-symvers-in-Linux
(566) https://en.wikipedia.org/wiki/System.map
(567) https://www.cyberciti.biz/tips/linux-set-io-scheduling-class-priority.html
(568) https://www.askapache.com/optimize/optimize-nice-ionice/#Part_2_Optimizing_Disk_IO
(569) https://en.wikipedia.org/wiki/Binary-to-text_encoding
(570) https://www.linuxjournal.com/article/9001
(571) http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html
(572) https://www.tecmint.com/understand-linux-load-averages-and-monitor-performance/
(573) https://www.networkworld.com/article/3291616/linux/examining-linux-system-performance-with-dstat.html
(574) https://www.tecmint.com/dstat-monitor-linux-server-performance-process-memory-network/
(575) https://linuxaria.com/howto/linux-terminal-dstat-monitoring-tools
(576) https://events.static.linuxfound.org/images/stories/slides/jls09/jls09_wieers.pdf
→ **(Pluggable real-time performance monitoring)** *[10.mj.2021]*
(577) https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
(578) https://www.maketecheasier.com/check-sha1-sha256-sha512-hashes-on-linux/
(579) https://itsfoss.com/checksum-tools-guide-linux/
(580) https://www.tldp.org/LDP/lame/LAME/linux-admin-made-easy/shadow-file-formats.html
(581) https://www.howtoforge.com/linux-cksum-command/
(582) http://linuxconfig.net/manuals/howto/etc-group-file.html
(583) https://www.cyberciti.biz/faq/understanding-etcgroup-file/
(584) https://en.wikipedia.org/wiki/Passwd#Password_file
(585) https://linux.die.net/man/5/passwd
(586) http://man7.org/linux/man-pages/man5/login.defs.5.html
(587) https://www.thegeekdiary.com/understanding-etclogin-defs-file/
(588) https://en.wikipedia.org/wiki/Linux_PAM
(589) https://stackoverflow.com/questions/14471564/what-does-ulimit-s-unlimited-do
(590) https://stackoverflow.com/questions/4185017/maximum-number-of-bash-arguments-max-num-cp-arguments
(591) https://www.in-ulm.de/~mascheck/various/argmax/
(592) https://www.mkssoftware.com/docs/man1/getconf.1.asp
(593) https://unix.stackexchange.com/questions/120642/what-defines-the-maximum-size-for-a-command-single-argument
(594) https://www.mkssoftware.com/docs/man1/getconf.1.asp
(595) http://pubs.opengroup.org/onlinepubs/9699919799/utilities/getconf.html
(596) https://www.thegeekdiary.com/understanding-etc-security-limits-conf-file-to-set-ulimit/
(597) https://gerardnico.com/os/linux/limits.conf
(598) https://lzone.de/cheat-sheet/ulimit
(599) https://en.wikipedia.org/wiki/Name_Service_Switch
(600) https://developers.redhat.com/blog/2018/11/26/etc-nsswitch-conf-non-complexity/
(601) https://linux.die.net/man/5/nsswitch.conf
(602) https://en.wikipedia.org/wiki/Unix_domain_socket
(603) https://www.engineersgarage.com/tutorials/socket-linux
(604) https://stackoverflow.com/questions/47703401/is-there-any-purpose-to-bind-unix-domain-socket-client-processes
(605) https://linux.die.net/man/7/unix
(606) https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/kernel-parameters.txt
(607) https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-kbopt.html
(608) http://man7.org/linux/man-pages/man7/bootparam.7.html
(609) https://www.thegeekdiary.com/centos-rhel-7-how-to-modify-grub2-arguments-with-grubby/
(610) https://www.applianceshop.eu/security-appliances/19-rack-appliances/pfsense-based-55/pffirewall-quad-core-gen4-4x10gb-ssd.html
(611) https://www.netgate.com/products/appliances/
(612) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s1-networkscripts-functions
(613) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s1-networkscripts-control
(614) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-configuring_ip_networking_with_ifcg_files
(615) https://ahelpme.com/linux/howto-do-qemu-full-virtualization-with-bridged-networking/
(616) https://www.itzgeek.com/how-tos/mini-howtos/create-a-network-bridge-on-centos-7-rhel-7.html
(617) https://www.thegeekstuff.com/2017/06/brctl-bridge/
(618) https://sgros-students.blogspot.com/2013/11/comparison-of-brctl-and-bridge-commands.html
(619) https://www.cyberciti.biz/faq/how-to-install-kvm-on-centos-7-rhel-7-headless-server/
(620) https://www.linuxjournal.com/article/8172
(621) https://paulgorman.org/technical/linux-nftables.txt.html
(622) https://linoxide.com/firewall/configure-nftables-serve-internet/
(623) https://linux-audit.com/differences-between-iptables-and-nftables-explained/
(624) https://linux-audit.com/nftables-beginners-guide-to-traffic-filtering/
(625) https://wiki.nftables.org/wiki-nftables/index.php/Quick_reference-nftables_in_10_minutes
(626) https://www.linuxjournal.com/content/bash-extended-globbing
(627) https://www.linuxjournal.com/content/pattern-matching-bash?fbclid=IwAR29iESWGkjl_BrgMvHqt4ei_egNtkVlO7wNAWILvBP59lsL0kYD8tOYIno
(628) https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

→ **(Virtual Memory and Linux)** *[10.mj.2021]*
**(629)** https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt
**(630)** https://manybutfinite.com/post/anatomy-of-a-program-in-memory/
**(631)** https://github.com/prestodb/presto/issues/8993
**(632)** https://stackoverflow.com/questions/26041117/growing-resident-memory-usage-rss-of-java-process
**(633)**
https://www.ibm.com/developerworks/community/blogs/kevgrig/entry/linux_glibc_2_10_rhel_6_malloc_may_show_excessive_virtual_memory_usage?lang=en
**(634)** https://stackoverflow.com/questions/43799756/ubuntu-malloc-arena-max-default-value
**(635)** https://devcenter.heroku.com/articles/tuning-glibc-memory-behavior
**(636)** https://stackoverflow.com/questions/561245/virtual-memory-usage-from-java-under-linux-too-much-memory-used
**(637)** https://spring.io/blog/2019/03/11/memory-footprint-of-the-jvm
**(638)** https://kernelnewbies.org/Linux_2_6_32?cm_mc_uid=65963004215814536603021&cm_mc_sid_50200000=1465752986#head-d3f32e41df508090810388a57efce73f52660ccb
**(639)** http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/usingkeepalive.html
**(640)** http://willbryant.net/overriding_the_default_linux_kernel_20_second_tcp_socket_connect_timeout
**(641)** http://coryklein.com/tcp/2015/11/25/custom-configuration-of-tcp-socket-keep-alive-timeouts.html
**(642)** https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt
**(643)** http://www.bernardi.cloud/2012/08/07/playing-around-with-perf/
**(644)** https://stackoverflow.com/questions/22165299/what-are-stalled-cycles-frontend-and-stalled-cycles-backend-in-perf-stat-resul
**(645)** https://hackertarget.com/tcpdump-examples/
**(646)** https://opensource.com/article/18/10/introduction-tcpdump
**(647)** https://www.sciencedirect.com/topics/computer-science/tcpdump
**(648)** https://medium.com/@eranda/analyze-tcp-dumps-a089c2644f19
**(649)** https://www.tcpdump.org/manpages/tcpdump.1.html
**(650)** https://packetpushers.net/masterclass-tcpdump-interpreting-output/
**(651)** http://alumni.cs.ucr.edu/~marios/ethereal-tcpdump.pdf → **(TCPdump filters)** *[10.mj.2021]*
**(652)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-transhuge
**(653)** https://www.kernel.org/doc/Documentation/vm/transhuge.txt
**(654)** https://alexandrnikitin.github.io/blog/transparent-hugepages-measuring-the-performance-impact/
**(655)** https://www.thegeekdiary.com/centos-rhel-6-how-to-disable-transparent-huge-pages-thp/
**(656)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/tuning_and_optimizing_red_hat_enterprise_linux_for_oracle_9i_and_10g_databases/sect-oracle_9i_and_10g_tuning_guide-large_memory_optimization_big_pages_and_huge_pages-configuring_huge_pages_in_red_hat_enterprise_linux_4_or_5
**(657)** https://kerneltalks.com/services/what-is-huge-pages-in-linux/
**(658)** https://wiki.debian.org/Hugepages
**(659)** https://dino.ciuffetti.info/2011/07/howto-java-huge-pages-linux/
**(660)** https://www.thegeekdiary.com/centos-rhel-67-how-to-configure-hugepages/
**(661)** https://en.wikipedia.org/wiki/Slab_allocation
**(662)** http://www.secretmango.com/jimb/Whitepapers/slabs/slab.html
**(663)** https://www.kernel.org/doc/gorman/html/understand/understand011.html
**(664)** https://www.kernel.org/doc/gorman/pdf/understand.pdf → **(Understanding The Linux Virtual Memory Manager)** *[10.mj.2021]*
**(665)** https://linoxide.com/linux-command/kernel-slab-cache-information/
**(666)** https://en.wikipedia.org/wiki/Master_boot_record
**(667)** https://wiki.archlinux.org/index.php/Partitioning
**(668)** https://developer.ibm.com/articles/l-linuxboot/
**(669)** https://en.wikipedia.org/wiki/Disk_formatting
**(670)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/s1-filesystem-ext4-create
**(671)** https://ewx.livejournal.com/579283.html
**(672)** http://man7.org/linux/man-pages/man5/proc.5.html
**(673)** https://en.wikipedia.org/wiki/File_descriptor
**(674)** https://en.wikipedia.org/wiki/Sparse_file
**(675)** https://ryanstutorials.net/linuxtutorial/piping.php
**(676)** https://landoflinux.com/linux_pipes_redirection.html
**(677)** https://en.wikipedia.org/wiki/RAID
**(678)** https://en.wikipedia.org/wiki/Standard_RAID_levels
**(679)** https://en.wikipedia.org/wiki/Nested_RAID_levels
**(680)** https://en.wikipedia.org/wiki/Non-standard_RAID_levels
**(681)** https://blog.open-e.com/how-does-raid-5-work/
**(682)** https://en.wikipedia.org/wiki/Kernel_same-page_merging
**(683)** https://web.archive.org/web/20140702045709/http://linux-kvm.com/content/using-ksm-kernel-samepage-merging-kvm
**(684)** https://www.kernel.org/doc/Documentation/kdump/kdump.txt
**(685)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/kernel_administration_guide/kernel_crash_dump_guide
**(686)** https://help.ubuntu.com/lts/serverguide/kernel-crash-dump.html.en
**(687)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch-logfiles
**(688)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-viewing_and_managing_log_files
**(689)** https://www.eurovps.com/blog/important-linux-log-files-you-must-be-monitoring/
**(690)** https://www.cyberciti.biz/faq/linux-log-files-location-and-how-do-i-view-logs-files/
**(691)** https://www.plesk.com/blog/featured/linux-logs-explained/
**(692)** https://en.wikipedia.org/wiki/X_Window_System
**(693)** https://en.wikipedia.org/wiki/X.Org_Server
**(694)** https://en.wikipedia.org/wiki/X_window_manager
**(695)** https://en.wikipedia.org/wiki/Link-state_routing_protocol
**(696)** https://en.wikipedia.org/wiki/Distance-vector_routing_protocol
**(697)** https://en.wikipedia.org/wiki/Multicast
**(698)** https://en.wikipedia.org/wiki/Multicast_address

(699) https://www.linuxjournal.com/article/3041
(700) http://www.tldp.org/HOWTO/text/Multicast-HOWTO
(701) https://en.wikipedia.org/wiki/IP_fragmentation
(702) https://en.wikipedia.org/wiki/TUN/TAP
(703) https://www.kernel.org/doc/Documentation/networking/tuntap.txt
(704) http://www.naturalborncoder.com/virtualization/2014/10/17/understanding-tun-tap-interfaces/
(705) https://en.wikipedia.org/wiki/Address_Resolution_Protocol
(706) https://wiki.wireshark.org/Gratuitous_ARP
(707) https://en.wikipedia.org/wiki/Maximum_segment_size
(708) https://blog.confirm.ch/tcp-connection-states/
(709) https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.halu101/constatus.htm
(710) https://en.wikipedia.org/wiki/Hosts_(file)
(711) https://vitux.com/linux-hosts-file/
(712) https://en.wikipedia.org/wiki/Resolv.conf
(713) http://man7.org/linux/man-pages/man5/resolv.conf.5.html
(714) https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-s5-netconf.html
(715) ftp://ftp.iitb.ac.in/LDP/en/solrhe/chap9sec93.html
(716) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/ch-the_sysconfig_directory
(717) https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol
(718) https://en.wikipedia.org/wiki/File_Transfer_Protocol
(719) https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
(720) https://en.wikipedia.org/wiki/Bootstrap_Protocol
(721) https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/sec-Using_iptables.html
(722) https://en.wikipedia.org/wiki/Iptables
(723) https://blog.manasg.com/fun-with-ipset-and-iptables/
(724) https://www.linuxjournal.com/content/advanced-firewall-configurations-ipset
(725) https://www.unixmen.com/iptables-vs-firewalld/
(726) https://opensource.com/article/19/8/linux-kernel-21st-century
(727) https://www.shellscript.sh/trap.html
(728) https://www.linuxjournal.com/content/bash-trap-command
(729) https://hackaday.com/2019/08/26/linux-fu-its-a-trap/
(730) https://www.computerhope.com/unix/utrap.htm
(731) https://www.putorius.net/using-trap-to-exit-bash-scripts-cleanly.html
(732) https://loicpefferkorn.net/2018/09/linux-network-statistics-reference/
(733) https://cromwell-intl.com/open-source/sysfs.html
(734) https://fasterdata.es.net/assets/Papers-and-Publications/100G-Tuning-TechEx2016.tierney.pdf
→ **(Recent Linux TCP Updates, and how to tune your 100G host)** *[10.mj.2021]*
(735) https://www.lartc.org/howto/lartc.qdisc.html
(736) https://gist.github.com/SaveTheRbtz/19928e50f42985d0ad752d5261ebed83
(737) https://codel.bufferbloat.narkive.com/W4xg2O3k/hardware-multiqueue-in-fq-codel
(738) https://core.ac.uk/download/pdf/12521647.pdf *[10.mj.2019]*
(739) https://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/
(740) http://www.haifux.org/lectures/299/netLec7.pdf
→ **(Resource management: Linux kernel Namespaces and cgroups)** *[10.mj.2021]*
(741) https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking
(742) https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking/#veth
(743) https://www.fir3net.com/Networking/Terms-and-Concepts/virtual-networking-devices-tun-tap-and-veth-pairs-explained.html
(744) https://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/
(745) https://linux-blog.anracom.com/tag/nsenter/
(746) https://www.linuxjournal.com/content/everything-you-need-know-about-linux-containers-part-i-linux-control-groups-and-process
(747) https://www.2daygeek.com/linux-sed-to-find-and-replace-string-in-files/
(748) https://www.redhat.com/en/blog/if-you-bonding-you-will-love-teaming
(749) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-comparison_of_network_teaming_to_bonding
(750) https://www.lisenet.com/2016/configure-aggregated-network-links-on-rhel-7-bonding-and-teaming/
(751) https://www.tldp.org/LDP/abs/html/internalvariables.html
(752) http://lettieri.iet.unipi.it/virtualization/2015/io-paravirtualization-tour.pdf
→ **(Virtio networking: A case study of I/O paravirtualization)** *[10.mj.2021]*
(753) https://www.computerhope.com/unix/bash/source.htm
(754) https://bash.cyberciti.biz/guide/Source_command
(755) https://www.geeksforgeeks.org/source-command-in-linux-with-examples/
(756) https://en.wikipedia.org/wiki/Throughput
(757) https://en.wikipedia.org/wiki/Bandwidth_(computing)
(758) https://en.wikipedia.org/wiki/Medium-dependent_interface
(759) https://en.wikipedia.org/wiki/Carrier-sense_multiple_access_with_collision_detection
(760) https://en.wikipedia.org/wiki/Duplex_(telecommunications)
(761) https://en.wikipedia.org/wiki/Autonegotiation
(762) https://en.wikipedia.org/wiki/Latency_(engineering)#Packet-switched_networks
(763) https://en.wikipedia.org/wiki/OSI_model
(764) https://en.wikipedia.org/wiki/Application_layer
(765) https://en.wikipedia.org/wiki/Transport_layer
(766) https://en.wikipedia.org/wiki/Internet_layer
(767) https://en.wikipedia.org/wiki/Link_layer
(768) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system-level_authentication_guide/configuring_services
(769) https://medium.com/@pawilon/tuning-your-linux-kernel-and-haproxy-instance-for-high-loads-1a2105ea553e
(770) https://ixnfo.com/en/tuning-nf_conntrack.html

---

(771) https://en.wikipedia.org/wiki/Port_(computer_networking)

(772) https://developers.soundcloud.com/blog/shoot-yourself-in-the-foot-with-iptables-and-kmod-auto-loading
(773) https://en.wikipedia.org/wiki/Ephemeral_port
(774) https://making.pusher.com/ephemeral-port-exhaustion-and-how-to-avoid-it/
(775) http://aleccolocco.blogspot.com/2008/11/ephemeral-ports-problem-and-solution.html
(776) https://wiki.khnet.info/index.php/Conntrack_tuning
(777) https://en.wikipedia.org/wiki/Network_socket
(778) https://ops.tips/blog/how-linux-creates-sockets/
(779) http://people.cs.pitt.edu/~ihsan/pacing_cal.pdf → **(University of Pittsburgh: TCP Pacing Revisited)** *[07.mj.2019]*
(780) https://www.howtogeek.com/192628/mime-types-explained-why-linux-and-mac-os-x-dont-need-file-extensions/
(781) https://stackoverflow.com/questions/2227182/how-can-i-find-out-a-files-mime-type-content-type
(782) https://linux.die.net/man/1/mimetype
(783) https://en.wikipedia.org/wiki/Media_type
(784) https://tools.ietf.org/html/rfc6838
(785) https://www.iana.org/assignments/media-types/media-types.xhtml
(786) https://en.m.wikipedia.org/wiki/Magic_number_(programming)
(787) https://en.m.wikipedia.org/wiki/File_format#Magic_number
(788) https://en.m.wikipedia.org/wiki/List_of_file_signatures
(789) http://www.linfo.org/root_filesystem.html
(790) https://www.tldp.org/LDP/sag/html/root-fs.html
(791) https://elinux.org/images/b/b1/Filesystems-for-embedded-linux.pdf → **(Survey of Filesystems for Embedded Linux)** *[10.mj.2021]*
(792) https://en.wikipedia.org/wiki/Initial_ramdisk
(793) https://en.wikipedia.org/wiki/Spanning_Tree_Protocol
(794) https://en.wikipedia.org/wiki/Network_address_translation
(795) https://en.wikipedia.org/wiki/Hot_Standby_Router_Protocol
(796) https://en.wikipedia.org/wiki/Virtual_Router_Redundancy_Protocol
(797) https://www.keepalived.org/manpage.html
(798) https://milestone-of-se.nesuke.com/en/nw-basic/link-aggregation/lacp-format/
(799) https://milestone-of-se.nesuke.com/en/nw-basic/link-aggregation/slow-protocol/
(800) http://movingpackets.net/2017/10/17/decoding-lacp-port-state/
(801) https://www.slideshare.net/plvision/lacp-agreement-38206486
(802) https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
(803) https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#Datagram_structure
(804) https://linuxhandbook.com/findmnt-command-guide/
(805) https://www.automotivelinux.org/about/members
(806) https://www.linuxfoundation.org/membership/members/
(807) https://www.linuxjournal.com/content/why-largest-companies-world-count-linux-servers
(808) https://www.techrepublic.com/article/five-big-names-that-use-linux-on-the-desktop/
(809) https://en.wikipedia.org/wiki/List_of_Linux_adopters
(810) http://www.linuxandubuntu.com/home/10-top-companies-that-are-powered-by-linux
(811) https://www.tecmint.com/big-companies-and-devices-running-on-gnulinux/
(812) https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/public-money-public-code
(813) https://hostingtribunal.com/blog/linux-statistics/#gref
(814) https://www.linuxfoundation.org/projects/case-studies/linux/
(815) https://news.netcraft.com/archives/2019/12/10/december-2019-web-server-survey.html
(816) https://www.top500.org/statistics/sublist/
(817) https://www.gnu.org/software/bash/manual/bash.pdf → **(Bash Reference Manual)** *[10.mj.2021]* ***[K-13]***
(818) https://www.zdnet.com/article/can-the-internet-exist-without-linux/
(819) https://www.comparebusinessproducts.com/fyi/50-places-linux-running-you-might-not-expect
(820) https://www.howtoforge.com/linux-chattr-command/
(821) https://www.geeksforgeeks.org/chattr-command-in-linux-with-examples/
(822) https://en.wikipedia.org/wiki/Chattr
(823) https://en.m.wikipedia.org/wiki/Express_Data_Path
(824) https://en.m.wikipedia.org/wiki/Berkeley_Packet_Filter
(825) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.1_release_notes/new-features
(826) https://www.kernel.org/doc/html/v4.18/networking/af_xdp.html
(827) http://vger.kernel.org/lpc_net2018_talks/XDP_meta-data_LPC_final_final.pdf
→ **(SDP Acceleration using using NIC metadata)** *[10.mj.2021]*
(828) https://cilium.readthedocs.io/en/latest/bpf/
(829) https://www.netronome.com/blog/bpf-ebpf-xdp-and-bpfilter-what-are-these-things-and-what-do-they-mean-enterprise/
(830) https://github.com/tohojo/xdp-paper/blob/master/benchmarks/bench01_baseline.org#initial-data-from-jespers-runs
(831) https://www.netronome.com/media/documents/PB_NFP-6000-7-20.pdf
→ **(NFP-6xxx – A 22nm High-Performance Network Flow Processor for 200Gb/s Software Defined Networking)** *[04.mj.2020]*
(832) https://www.netronome.com/products/agilio-cx/
(833) https://www.servethehome.com/intel-ethernet-800-series-100gbe-nic-launch/
(834) https://www.netronome.com/m/documents/UG_Getting_Started_with_eBPF_Offload.pdf
→ **(eBPF Offload Getting Started Guide)** *[04.mj.2020]*
(835) http://www.brendangregg.com/blog/2019-01-01/learn-ebpf-tracing.html
(836) https://www.redhat.com/en/blog/introduction-ebpf-red-hat-enterprise-linux-7
(837) https://developers.redhat.com/blog/2018/12/03/network-debugging-with-ebpf/
(838) https://medium.com/@fntlnz/load-xdp-programs-using-the-ip-iproute2-command-502043898263
(839) https://github.com/iovisor/bpf-docs/blob/master/eBPF.md
(840) https://medium.com/@ugendreshwarkudupudi/bpfilter-your-next-firewall-engine-5f7dc63ebc3
(841) https://www.slideshare.net/Netronome/unifying-network-filtering-rules-for-the-linux-kernel-with-ebpf
(842) http://man7.org/linux/man-pages/man8/tc-bpf.8.html
(843) https://lwn.net/Articles/750845/
(844) https://qmo.fr/docs/talk_20180316_frnog_bpfilter.pdf → **(bpfilter, pare-feu Linux à la sauce eBPF)** *[10.mj.2021]*
(845) http://docs.openvswitch.org/en/latest/intro/install/afxdp
(846) https://www.ntop.org/wp-content/uploads/2019/05/Cardigliano.pdf → **(Joining Forces: PF_RING and XDP)** *[10.mj.2021]*
(847) https://www.stamus-networks.com/wp-content/uploads/2019/07/suricata-ebpf-xdp-1.pdf

→ **(Introduction to eBPF and XDP support in Suricata)** *[07.mj.2020]*
**(848)** https://en.wikipedia.org/wiki/Init
**(849)** https://en.wikipedia.org/wiki/Runlevel
**(850)** https://wiki.phoenixlzx.com/page/ssacli/
**(851)** https://serverfault.com/questions/962451/enabling-drive-write-cache-using-smart-storage-admin-cli
**(852)** https://en.wikipedia.org/wiki/Paging
**(853)** https://en.wikipedia.org/wiki/Memory_management
**(854)** https://en.wikipedia.org/wiki/Memory_segmentation
**(855)** https://en.wikipedia.org/wiki/Virtual_memory
**(856)** https://en.wikipedia.org/wiki/Page_table
**(857)** http://man7.org/linux/man-pages/man5/proc.5.html
**(858)** https://docs.openstack.org/developer/performance-docs/test_plans/hardware_features/hardware_offloads/plan.html
**(859)** https://www.cyberciti.biz/tips/what-is-devshm-and-its-practical-usage.html
**(860)** https://superuser.com/questions/45342/when-should-i-use-dev-shm-and-when-should-i-use-tmp
**(861)** https://en.wikipedia.org/wiki/RAM_drive
**(862)** http://man7.org/linux/man-pages/man5/tmpfs.5.html
**(863)** https://www.cyberciti.biz/files/linux-kernel/Documentation/filesystems/tmpfs.txt
**(864)** https://blog.cloudflare.com/speeding-up-linux-disk-encryption/amp/?fbclid=IwAR2FLGlXXVvwAAcxMz3j9U7Abn1wkha2oo1X7NYEgTMwXZitEooBklND0DA
**(865)** https://unix.stackexchange.com/questions/157041/how-linux-kernel-3-x-manage-ramdisk-as-block-device
**(866)** https://www.kernel.org/doc/html/latest/admin-guide/tainted-kernels.html
**(867)** https://www.suse.com/support/kb/doc/?id=000016321
**(868)** https://en.wikipedia.org/wiki/SeaBIOS
**(869)** https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine
**(870)** https://en.wikipedia.org/wiki/QEMU
**(871)** https://www.berrange.com/posts/2018/06/29/cpu-model-configuration-for-qemu-kvm-on-x86-hosts/
**(872)** https://vmsplice.net/~stefan/qemu-kvm-architecture-2015.pdf → **(KVM Architecture Overview)** *[10.mj.2021]*
**(873)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/virtualization_host_configuration_and_guest_installation_guide/chap-virtualization_host_configuration_and_guest_installation_guide-para_virtualized_drivers
**(874)** https://www.linux-kvm.org/images/d/dd/KvmForum2007%24kvm_pv_drv.pdf → **(KVM PV DEVICES)** *[10.mj.2021]*
**(875)** http://docs.oasis-open.org/virtio/virtio/v1.1/virtio-v1.1.pdf
→ **(Virtual I/O Device (VIRTIO) Version 1.1 Committee Specification 01)** *[10.mj.2021]*
**(876)** https://lettieri.iet.unipi.it/virtualization/2018/20161124-io-paravirtualization-tour.pdf
→ **(Virtio networking: A case study of I/O paravirtualization)** *[10.mj.2021]*
**(877)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-using_qemu_img-supported_qemu_img_formats
**(878)** https://www.cyberciti.biz/tips/linux-security.html
**(879)** https://www.tecmint.com/linux-server-hardening-security-tips/
**(880)** https://medium.com/viithiisys/10-steps-to-secure-linux-server-for-production-environment-a135109a57c5
**(881)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/pdf/using_authselect_on_a_red_hat_enterprise_linux_host/Red_Hat_Enterprise_Linux-8-Using_authselect_on_a_Red_Hat_Enterprise_Linux_host-en-US.pdf
→ **(Red Hat Enterprise Linux 8 Using authselect on a Red Hat Enterprise Linux host)** *[11.mj.2020]*
**(882)** https://www.mankier.com/7/authselect-migration
**(883)** https://documentation.suse.com/sles/11-SP4/html/SLES-kvm4zseries/cha-qemu-cachemodes.html
**(884)** https://computingforgeeks.com/resize-ext-and-xfs-root-partition-without-lvm/
**(885)** https://www.linux-kvm.org/page/Multiqueue#Multiqueue_virtio-net
**(886)** https://www.linux-kvm.org/images/0/06/2012-forum-Q35.pdf → **(A New Chipset For Qemu - Intel's Q35)** *[10.mj.2021]*
**(887)** https://qemu.weilnetz.de/doc/qemu-doc.html#Standard-options
**(888)** https://www.tecmint.com/install-create-run-lxc-linux-containers-on-centos/
**(889)** https://www.cyberciti.biz/faq/how-to-install-and-setup-lxc-linux-container-on-fedora-linux-26/
**(890)** https://unix.stackexchange.com/questions/358982/network-interface-name-has-with-at-sign-what-is-it
**(891)** https://unix.stackexchange.com/questions/441876/how-to-find-the-network-namespace-of-a-veth-peer-ifindex
**(892)** https://linuxcontainers.org/lxc/manpages/man5/lxc.container.conf.5.html
**(893)** https://snikt.net/blog/2014/04/07/how-to-convert-an-kvm-image-into-a-lxc-container/
**(894)** https://archives.flockport.com/lxc-advanced-guide/

---

**(895)** https://wiki.centos.org/HowTos/SELinux
**(896)** https://www.digitalocean.com/community/tutorials/an-introduction-to-selinux-on-centos-7-part-1-basic-concepts
**(897)** https://www.thegeekdiary.com/what-are-selinux-users-and-how-to-map-linux-users-to-selinux-users/
**(898)** https://wiki.gentoo.org/wiki/SELinux/Users_and_logins
**(899)** https://www.digitalocean.com/community/tutorials/an-introduction-to-selinux-on-centos-7-part-2-files-and-processes
**(900)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security-enhanced_linux/chap-security-enhanced_linux-selinux_contexts
**(901)** https://selinuxproject.org/page/TypeRules
**(902)** http://www.cse.psu.edu/~trj1/cse543-f07/slides/03-PolicyConcepts.pdf → **(SELinux Policy Concepts and Overview)** *[10.mj.2021]*
**(903)** https://www.thegeekstuff.com/2011/09/grub-password-command/
**(904)** https://www.thegeekdiary.com/how-to-password-protect-grub2-in-oracle-enterprise-linux-7/
**(905)** https://nvd.nist.gov/ncp/checklist/811
**(906)** https://nvd.nist.gov/ncp/checklist/909
**(907)** https://www.cs.nmsu.edu/~istrnad/cs480/lecture_notes/lecture14.html
**(908)** https://www.thegeekstuff.com/2011/02/linux-boot-process/
**(909)** https://www.ssi.gouv.fr/en/guide/configuration-recommendations-of-a-gnulinux-system/
**(910)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/configuring-policy-based-routing-to-define-alternative-routes_configuring-and-managing-networking
**(911)** https://linuxplumbersconf.org/event/7/contributions/670/attachments/596/1072/fb_netgpu.pdf → **(Facebook NETGPU)** *[10.mj.2021]*
**(912)** https://en.wikipedia.org/wiki/Round-robin_DNS
**(913)** https://keepalived.org/manpage.html
**(914)** https://www.redhat.com/sysadmin/sort-command-linux

**(915)** https://www.tecmint.com/sort-command-linux/
**(916)** https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c03474784
**(917)** https://superuser.com/questions/240456/how-to-interpret-the-output-of-netstat-o-netstat-timers
**(918)** https://en.wikipedia.org/wiki/Sudo
**(919)** https://www.howtogeek.com/111479/htg-explains-whats-the-difference-between-sudo-su/
**(920)** https://www.redhat.com/en/blog/red-hat-leading-enterprise-linux-server-market
**(921)** https://gravitational.com/blog/ssh-config/
**(922)** https://en.wikipedia.org/wiki/Bufferbloat
**(923)** https://www.bufferbloat.net/projects/bloat/wiki/What_can_I_do_about_Bufferbloat/
**(924)** https://www.kernel.org/doc/Documentation/filesystems/tmpfs.txt
**(925)** https://refspecs.linuxfoundation.org/FHS_3.0/fhs/ch03s15.html
**(926)** https://www.researchgate.net/publication/323123606_Performance_Analysis_of_Ivshmem_for_High-Performance_Computing_in_Virtual_Machines/link/5a8194c2458515ce614108b4/download
**(927)** https://github.com/qemu/qemu/blob/master/docs/specs/ivshmem-spec.txt
**(928)** https://github.com/FabricAttachedMemory/tm-ivshmem-server
**(929)** https://success.trendmicro.com/solution/TP000086372-TippingPoint-devices-and-Out-of-Order-Packets
**(930)** https://ask.wireshark.org/question/9070/how-does-wireshark-determine-if-a-tcp-packet-is-out-of-order/
**(931)** https://osqa-ask.wireshark.org/questions/51741/retransmitted-vs-out-of-order-packets
**(932)** https://man7.org/linux/man-pages/man8/systemd-rc-local-generator.8.html
**(933)** https://www.cyberciti.biz/faq/how-to-enable-rc-local-shell-script-on-systemd-while-booting-linux-system/
**(934)** https://www.systutorials.com/docs/linux/man/5-systemd.service/
**(935)** https://blog.pentesteracademy.com/linux-security-understanding-linux-capabilities-series-part-i-4034cf8a7f09
**(936)** https://rtodto.net/path-mtu-ip-fragmentation-and-mss/
**(937)** https://en.wikipedia.org/wiki/Path_MTU_Discovery
**(938)** https://vedslinux.blogspot.com/2014/06/login-process-in-linux.html
**(939)** https://www.linuxjournal.com/article/3121
**(940)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/managing_smart_cards/pluggable_authentication_modules
**(941)** https://www.techrepublic.com/article/controlling-passwords-with-pam/
**(942)** http://kb.linuxvirtualserver.org/wiki/Mini_Mini_Howto
**(943)** https://en.wikipedia.org/wiki/Linux_Virtual_Server
**(944)** https://www.techtransit.org/network-configuration-nmcli-manager-on-centos-7-rhel-7/
**(945)** https://linoxide.com/linux-command/nmcli-tool-red-hat-centos-7/
**(946)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/getting_started_with_networkmanager
**(947)** https://linuxhint.com/disk_quota_ubuntu/
**(948)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/storage_administration_guide/ch-disk-quotas
**(949)** https://www.linuxtechi.com/proxy-settings-yum-command-on-rhel-centos-servers
**(950)** https://en.wikipedia.org/wiki/Proxy_server
**(951)** https://en.wikipedia.org/wiki/Squid_(software)
**(952)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/ch-consistent_network_device_naming
**(953)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-understanding_the_device_renaming_procedure
**(954)** https://en.wikipedia.org/wiki/Anycast
**(955)** https://www.cloudflare.com/learning/cdn/glossary/anycast-network/
**(956)** https://www.imperva.com/blog/how-anycast-works/
**(957)** https://developers.google.com/speed/public-dns/faq
**(958)** http://ddiguru.com/blog/anycast-dns-using-rip-part-1
**(959)** http://ddiguru.com/blog/anycast-dns-part-4-using-ospf-basic
**(960)** https://blog.benjojo.co.uk/post/ping-with-loss-latency-split
**(961)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/installation_guide/sn-booting-from-pxe-x86
**(962)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/installation_guide/s1-netboot-pxe-config
**(963)** https://sysportal.carnet.hr/node/658
**(964)** https://www.ssh.com/ssh/tunneling/
**(965)** https://www.ssh.com/ssh/tunneling/example
**(966)** https://linuxize.com/post/how-to-setup-ssh-tunneling/
**(967)** https://en.wikipedia.org/wiki/Operating_system
**(968)** https://simple.wikipedia.org/wiki/Operating_system
**(969)** https://hr.wikipedia.org/wiki/Operacijski_sustav
**(970)** https://linuxize.com/post/wget-command-examples/
**(971)** https://www.computerhope.com/unix/wget.htm
**(972)** https://www.computerhope.com/unix/curl.htm
**(973)** https://www.keycdn.com/support/popular-curl-examples
**(974)** https://www.thegeekstuff.com/2012/04/curl-examples/
**(975)** https://en.wikipedia.org/wiki/Comma-separated_values
**(976)** https://en.wikibooks.org/wiki/OpenSSH/SSH_Protocols
**(977)** https://goteleport.com/blog/ssh-handshake-explained/
**(978)** https://en.wikipedia.org/wiki/Symmetric_multiprocessing
**(979)** https://en.wikipedia.org/wiki/Non-uniform_memory_access
**(980)** https://software.intel.com/content/www/us/en/develop/articles/intel-xeon-processor-scalable-family-technical-overview.html
**(981)** https://access.redhat.com/solutions/15693
**(982)** https://linuxconfig.org/installation-of-memtest-ram-memory-test-tool-on-redhat-7-linux
**(983)** https://en.wikipedia.org/wiki/Structured_cabling
**(984)** https://hr.wikipedia.org/wiki/Strukturno_kabliranje
**(985)** https://en.wikipedia.org/wiki/USB
**(986)** https://hr.wikipedia.org/wiki/USB
**(987)** https://opensource.com/article/20/8/usb-id-repository
**(988)** https://en.wikipedia.org/wiki/Subnetwork
**(989)** https://en.wikipedia.org/wiki/Routing_table

**(990)** https://en.wikipedia.org/wiki/Linear_Tape-Open
**(991)** https://en.wikipedia.org/wiki/Synchronous_dynamic_random-access_memory
**(992)** https://en.wikipedia.org/wiki/Dynamic_random-access_memory
**(993)** https://en.wikipedia.org/wiki/Error_correction_code
**(994)** https://en.wikipedia.org/wiki/SCSI
**(995)** https://en.wikipedia.org/wiki/Serial_ATA
**(996)** https://en.wikipedia.org/wiki/Parallel_ATA
**(997)** https://en.wikipedia.org/wiki/Serial_Attached_SCSI
**(998)** https://en.wikipedia.org/wiki/Advanced_Host_Controller_Interface
**(999)** https://en.wikipedia.org/wiki/NVM_Express
**(1000)** https://en.wikipedia.org/wiki/BIOS
**(1001)** https://en.wikipedia.org/wiki/Disk_storage
**(1002)** https://en.wikipedia.org/wiki/Cylinder-head-sector
**(1003)** https://en.wikipedia.org/wiki/File_system
**(1004)** https://en.wikipedia.org/wiki/File_Allocation_Table
**(1005)** https://en.wikipedia.org/wiki/NTFS
**(1006)** https://en.wikipedia.org/wiki/Hard_disk_drive_performance_characteristics
**(1007)** https://en.wikipedia.org/wiki/Solid-state_drive
**(1008)** https://www.howtogeek.com/745921/how-to-use-the-fsck-command-on-linux/
**(1009)** https://en.wikipedia.org/wiki/Logical_Volume_Manager_%28Linux%29
**(1010)** https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
**(1011)** https://en.wikipedia.org/wiki/Wide_area_network
**(1012)** https://en.wikipedia.org/wiki/Local_area_network
**(1013)** https://en.wikipedia.org/wiki/Small_form-factor_pluggable_transceiver
**(1014)** https://en.wikipedia.org/wiki/Medium-dependent_interface
**(1015)** https://en.wikipedia.org/wiki/Ethernet_hub
**(1016)** https://en.wikipedia.org/wiki/Bridging_(networking)
**(1017)** https://en.wikipedia.org/wiki/Application-specific_integrated_circuit
**(1018)** https://www.grotto-networking.com/BBSwitchArch.html
**(1019)** https://wiki.archlinux.org/title/Network_bridge
**(1020)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/networking_guide/index
**(1021)** https://en.wikipedia.org/wiki/Pcap
**(1022)** https://en.wikipedia.org/wiki/Time_to_live
**(1023)** http://smutz.us/techtips/NetworkLatency.html
**(1024)** https://arxiv.org/pdf/1610.03534.pdf
→ **(Comparative study of High-speed Linux TCP Variants over High-BDP Networks)** *[10.mj.2021]*
**(1025)** https://wiki.archlinux.org/title/Modalias
**(1026)** https://www.linuxfromscratch.org/lfs/view/development/chapter09/udev.html
**(1027)** https://lwn.net/Articles/47412/
**(1028)** https://stackoverflow.com/questions/54056731/where-does-the-systemd-builtin-kmod-gets-the-module-aliases-from
**(1029)** https://mirrors.edge.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf
→ **(The sysfs Filesystem)** *[10.mj.2021]*
**(1030)** https://en.wikipedia.org/wiki/Journaling_file_system
**(1031)** https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf
→ **(*Filesystem Hierarchy Standard*)** *[10.mj.2021]*
**(1032)** https://www.kernel.org/doc/html/latest/filesystems/vfs.html
**(1033)** https://en.wikipedia.org/wiki/Procfs
**(1034)** https://man7.org/linux/man-pages/man5/sysfs.5.html
**(1035)** https://www.kernel.org/doc/Documentation/filesystems/proc.txt
**(1036)** https://blog.apnic.net/2020/01/28/rss-a-new-model-for-elastic-high-speed-networking/
**(1037)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/network-rps
**(1038)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/network-rfs
**(1039)** https://community.mellanox.com/s/article/counters-troubleshooting-for-linux-driver
**(1040)** https://access.redhat.com/solutions/20278
**(1041)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-configuring_ip_networking_with_ifcg_files
**(1042)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-starting_networkmanager
**(1043)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/getting_started_with_networkmanager
**(1044)** https://en.wikipedia.org/wiki/Internet_Protocol
**(1045)** https://en.wikipedia.org/wiki/Network_topology
**(1046)** https://www.itjones.com/blogs/2020/11/22/a-guide-to-network-topology
**(1047)** https://www.arubanetworks.com/faq/what-is-spine-leaf-architecture/
**(1048)** https://community.fs.com/blog/leaf-spine-with-fs-com-switches.html
**(1049)** https://www.iana.org/assignments/ntp-parameters/ntp-parameters.xhtml
**(1050)** https://stuffphilwrites.com/2014/09/iptables-processing-flowchart/
**(1051)** https://en.wikipedia.org/wiki/Netfilter
**(1052)** https://www.ibm.com/docs/en/linux-on-systems?topic=choices-using-macvtap-driver
**(1053)** https://sreeninet.wordpress.com/2016/05/29/macvlan-and-ipvlan/
**(1054)** https://en.wikipedia.org/wiki/LXC
**(1055)** https://linuxcontainers.org/lxc/introduction/#LXC
**(1056)** https://docs.fedoraproject.org/en-US/Fedora/23/html/System_Administrators_Guide/sec-Configuring_GRUB_2_Using_the_grubby_Tool.html
**(1057)** https://en.wikipedia.org/wiki/Internet_Mix
**(1058)** https://en.wikipedia.org/wiki/List_of_file_formats
**(1059)** https://en.wikipedia.org/wiki/List_of_filename_extensions
**(1060)** https://www.gnu.org/software/gawk/manual/gawk.pdf → **(GNU AWK - GAWK: Effective AWK Programming )** *[11.mj.2021]*
**(1061)** https://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/
**(1062)** http://www.linfo.org/acronym_list.html
**(1063)** https://en.wikipedia.org/wiki/List_of_computing_and_IT_abbreviations

**(1064)** https://en.wikipedia.org/wiki/Single-mode_optical_fiber
**(1065)** https://en.wikipedia.org/wiki/Multi-mode_optical_fiber
**(1066)** https://en.wikipedia.org/wiki/IEEE_802.1ad
**(1067)** https://blog.nelhage.com/post/transparent-hugepages/
**(1068)** https://en.wikipedia.org/wiki/VT100
**(1069)** https://en.wikipedia.org/wiki/Box-drawing_character
**(1070)** https://developers.redhat.com/blog/2016/09/20/managing-temporary-files-with-systemd-tmpfiles-on-rhel7
**(1071)** https://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface
**(1072)** https://www.kernel.org/doc/html/latest/arm64/acpi_object_usage.html
**(1073)** https://uefi.org/specs/ACPI/6.4/21_ACPI_Data_Tables_and_Table_Def_Language/ACPI_Data_Tables.html
**(1074)** https://www.usenix.org/legacy/publications/library/proceedings/usenix02/tech/freenix/full_papers/watanabe/watanabe_html/node4.html
**(1075)** https://uefi.org/specs/ACPI/6.4/05_ACPI_Software_Programming_Model/ACPI_Software_Programming_Model.html
**(1076)** https://www.kernel.org/doc/ols/2005/ols2005v1-pages-59-76.pdf
→ **(ACPI in Linux)** *[12.mj.2021]*
**(1077)** https://en.wikipedia.org/wiki/Native_Command_Queuing
**(1078)** https://ata.wiki.kernel.org/index.php/Libata_FAQ
**(1079)** https://www.thessdreview.com/our-reviews/intel-ssd-dc-p3608-review-1-6tb-over-5gbs-and-850k-iops/
**(1080)** https://www.howtogeek.com/657972/nvme-vs.-sata-which-ssd-technology-is-faster/
**(1081)** https://opensource.com/article/21/9/nvme-cli
**(1082)** https://wiki.archlinux.org/title/Solid_state_drive/NVMe
**(1083)** https://itpeernetwork.intel.com/tuning-performance-intel-optane-ssds-linux-operating-systems/#gs.hrw3t7
**(1084)** https://nvmexpress.org/
**(1085)** https://en.wikipedia.org/wiki/Booting
**(1086)** https://en.wikipedia.org/wiki/Disk_sector
**(1087)** https://en.wikipedia.org/wiki/GNU_GRUB
**(1088)** https://en.wikipedia.org/wiki/Volume_boot_record
**(1089)** https://en.wikipedia.org/wiki/Design_of_the_FAT_file_system#Bootsector
**(1090)** https://en.wikipedia.org/wiki/High_availability
**(1091)** https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface
**(1092)**https://en.wikipedia.org/wiki/Real_mode
**(1093)** https://en.wikipedia.org/wiki/BIOS_boot_partition
**(1094)** https://en.wikipedia.org/wiki/EFI_system_partition
**(1095)** https://wiki.archlinux.org/title/EFI_system_partition
**(1096)** https://users.cms.caltech.edu/~donnie/cs124/lectures/CS124Lec05.pdf
→ **Caltech lectures (BOOTSTRAP, IA32, AND BIOS/UEFI)** *[1.12.2021]*
**(1097)** https://people.cs.rutgers.edu/~pxk/416/notes/02-boot.html
**(1098)** https://en.wikipedia.org/wiki/POSIX
**(1099)** https://en.wikipedia.org/wiki/Application_binary_interface
**(1100)** https://en.wikipedia.org/wiki/API
**(1101)** http://www.sco.com/developers/devspecs/gabi41.pdf → **System V Application Binary Interface** *[1.12.2021]*
**(1102)** https://hr.wikipedia.org/wiki/API
**(1103)** https://en.wikipedia.org/wiki/Binary-code_compatibility
**(1104)** https://en.wikipedia.org/wiki/Wine_(software)
**(1105)** https://en.wikipedia.org/wiki/Emulator
**(1106)** https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
**(1107)** https://www.guru99.com/operating-system-tutorial.html
**(1108)** https://netmarketshare.com/operating-system-market-share.aspx
**(1109)** https://www.datanyze.com/market-share/web-and-application-servers--425
**(1110)** https://en.wikipedia.org/wiki/Usage_share_of_operating_systems
**(1111)** https://en.wikipedia.org/wiki/Microkernel
**(1112)** https://en.wikipedia.org/wiki/Minix_3
**(1113)** https://hr.wikipedia.org/wiki/Vremenska_zona
**(1114)** https://en.wikipedia.org/wiki/Monolithic_kernel
**(1115)** https://en.wikipedia.org/wiki/Loadable_kernel_module
**(1116)** https://terenceli.github.io/%E6%8A%80%E6%9C%AF/2018/06/02/linux-loadable-module
**(1117)** https://www.kernel.org/doc/Documentation/kbuild/modules.txt
**(1118)** https://developers.redhat.com/blog/2018/03/28/analyzing-binary-interface-changes-linux-kernel
**(1119)** http://llseek.github.io/kernel/2015/08/08/linux-module-versioning.html
**(1120)** https://access.redhat.com/articles/rhel8-abi-compatibility
**(1121)** https://en.wikipedia.org/wiki/XZ_Utils
**(1122)** https://www.rootusers.com/13-simple-xz-examples/
**(1123)** https://en.wikipedia.org/wiki/Multi-channel_memory_architecture
**(1124)** https://beebom.com/single-channel-vs-dual-channel-memory/
**(1125)** https://en.wikipedia.org/wiki/List_of_interface_bit_rates
**(1126)** https://lwn.net/Articles/664688/
**(1127)** https://blog.cloudflare.com/programmable-packet-filtering-with-magic-firewall/
**(1128)** https://en.wikipedia.org/wiki/Email
**(1129)** https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
**(1130)** https://hr.wikipedia.org/wiki/Elektroni%C4%8Dka_po%C5%A1ta
**(1131)** https://en.wikipedia.org/wiki/Post_Office_Protocol
**(1132)** https://en.wikipedia.org/wiki/Email_address
**(1133)** https://en.wikipedia.org/wiki/Domain_name
**(1134)** https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml
**(1135)** https://en.wikipedia.org/wiki/Internet_Assigned_Numbers_Authority
**(1136)** https://en.wikipedia.org/wiki/Country_code_top-level_domain
**(1137)** https://en.wikipedia.org/wiki/Domain_Name_System
**(1138)** https://en.wikipedia.org/wiki/List_of_DNS_record_types
**(1139)** https://en.wikipedia.org/wiki/Unicode
**(1140)** https://lwn.net/Articles/668126/
**(1141)** https://www.cyberciti.biz/faq/linux-ip-command-examples-usage-syntax/

**(1142)** https://phoenixnap.com/kb/linux-ip-command-examples
**(1143)** https://linuxize.com/post/linux-ip-command/
**(1144)** https://www.deepspace6.net/docs/iproute2tunnel-en.html
**(1145)** https://www.lorier.net/docs/xfrm.html
**(1146)** https://www.kernel.org/doc/Documentation/networking/vrf.txt
**(1147)** https://www.kernel.org/doc/html/latest/networking/vrf.html
**(1148)** https://www.dasblinkenlichten.com/working-with-linux-vrfs/
**(1149)** https://docs.centos.org/en-US/centos/install-guide/Kickstart2/
**(1150)** https://docs.centos.org/en-US/8-docs/advanced-install/assembly_creating-kickstart-files/
**(1151)** https://wiki.syslinux.org/wiki/index.php
**(1152)** https://en.wikipedia.org/wiki/Preboot_Execution_Environment
**(1153)** https://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml
**(1154)** https://en.wikipedia.org/wiki/Linux_startup_process
**(1155)** https://en.wikipedia.org/wiki/Dnsmasq
**(1156)** https://www.winccoa.top/linux/E54695/html/ol7-install-pxe-dnsmasq.html
**(1157)** https://openthreat.ro/install-pxe-server-on-rhel-8-centos-8/
**(1158)** https://www.tecmint.com/install-pxe-network-boot-server-in-centos-7/
**(1159)** https://linuxhint.com/pxe_network_boot_centos8/
**(1160)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/performing_an_advanced_rhel_installation/preparing-for-a-network-install_installing-rhel-as-an-experienced-user
**(1161)** https://wiki.syslinux.org/wiki/index.php?title=PXELINUX
**(1162)** https://en.wikipedia.org/wiki/Server_Message_Block
**(1163)** https://en.wikipedia.org/wiki/AppleShare
**(1164)** https://en.wikipedia.org/wiki/WebDAV
**(1165)** https://en.wikipedia.org/wiki/Storage_area_network
**(1166)** https://en.wikipedia.org/wiki/Network-attached_storage
**(1167)** https://en.wikipedia.org/wiki/Fibre_Channel
**(1168)** https://en.wikipedia.org/wiki/Fibre_Channel_over_Ethernet
**(1169)** https://en.wikipedia.org/wiki/ISCSI
**(1170)** https://en.wikipedia.org/wiki/ATA_over_Ethernet
**(1171)** https://en.wikipedia.org/wiki/ZFS
**(1172)** https://en.wikipedia.org/wiki/RAID#Unrecoverable_read_errors_during_rebuild
**(1173)** https://en.wikipedia.org/wiki/Distributed_Replicated_Block_Device
**(1174)** https://en.wikipedia.org/wiki/Gluster#GlusterFS
**(1175)** https://en.wikipedia.org/wiki/Lustre_(file_system)
**(1176)** https://en.wikipedia.org/wiki/List_of_file_systems#Distributed_file_systems
**(1177)** https://ceph.com/en/discover/
**(1178)** https://indico.cern.ch/event/649159/contributions/2761965/attachments/1544385/2423339/hroussea-storage-at-CERN.pdf
→ **Upotreba CEPHa u CERNu** *[1.12.2021]*
**(1179)** https://ceph.io/geen-categorie/crushmap-example-of-a-hierarchical-cluster-map/
**(1180)** https://www.2daygeek.com/comparison-difference-between-dnf-vs-yum/
**(1180.1)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.0_release_notes/rhel-8_0_0_release#software-management
**(1181)** https://www.linuxtechi.com/configure-rsyslog-server-centos-8-rhel-8/
**(1182)** https://www.linuxfordevices.com/tutorials/remote-syslog-in-linux
**(1183)** https://stackify.com/syslog-101/
**(1184)** https://en.wikipedia.org/wiki/Syslog
**(1185)** https://en.wikipedia.org/wiki/Unix
**(1186)** https://linuxconfig.org/how-to-create-temporary-files-using-mktemp-on-linux
**(1187)** https://en.wikipedia.org/wiki/Temporary_folder
**(1188)** https://en.wikipedia.org/wiki/Temporary_file
**(1189)** https://www.freedesktop.org/software/systemd/man/systemd.resource-control.html
**(1190)** http://0pointer.de/blog/projects/resources.html
**(1191)** https://backreference.org/2013/04/03/firewall-ha-with-conntrackd-and-keepalived/index.html
**(1192)** https://conntrack-tools.netfilter.org/manual.html
**(1193)** https://lwn.net/Articles/330985/
**(1194)** https://en.wikipedia.org/wiki/EtherType
**(1195)** https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml
**(1196)** https://wiki.tldp.org/kernel_user_space_howto
**(1197)** https://embetronicx.com/tutorials/linux/device-drivers/sending-signal-from-linux-device-driver-to-user-space/
**(1198)** https://en.wikipedia.org/wiki/Cygwin
**(1199)** https://pythonspeed.com/articles/shell-scripts/
**(1200)** https://phoenixnap.com/kb/linux-set
**(1201)** https://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html
**(1202)** https://www.gnu.org/software/bash/manual/html_node/The-Shopt-Builtin.html
**(1203)** https://www.gnu.org/software/coreutils/manual/html_node/nice-invocation.html#nice-invocation
**(1204)** https://www.kernel.org/doc/Documentation/scheduler/sched-rt-group.txt
**(1205)** https://doc.opensuse.org/documentation/leap/archive/42.1/tuning/html/book.sle.tuning/cha.tuning.taskscheduler.html
**(1206)** https://en.wikipedia.org/wiki/Scheduling_(computing)
**(1207)** https://en.wikipedia.org/wiki/Completely_Fair_Scheduler
**(1208)** https://en.wikipedia.org/wiki/Brain_Fuck_Scheduler
**(1209)** https://en.wikipedia.org/wiki/SCHED_DEADLINE
**(1210)** https://notes.eddyerburgh.me/operating-systems/linux/process-scheduling
**(1211)** http://students.mimuw.edu.pl/ZSO/Wyklady/15_CPUschedulers2/ProcessScheduling2.pdf
→ **Process scheduling: O(N), O(1), RSDS, CFS, BFS, Deadline, MuQSS, etc. Performance comparison** *[1.4.2022]*
**(1212)** https://www.gnu.org/software/coreutils/manual/coreutils.html
**(1213)** https://docs.openstack.org/liberty/config-reference/content/networking-plugin-ml2_vxlan.html
**(1214)** https://ilearnedhowto.wordpress.com/2017/02/16/how-to-create-overlay-networks-using-linux-bridges-and-vxlans/
**(1215)** https://www.juniper.net/documentation/us/en/software/junos/evpn-vxlan/topics/topic-map/sdn-vxlan.html
**(1216)** https://en.wikipedia.org/wiki/Virtual_routing_and_forwarding

(1217) https://docs.kernel.org/networking/vrf.html
(1218) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/setting-your-routing-protocols_configuring-and-managing-networking
(1219) http://docs.frrouting.org/en/latest/
(1220) https://en.wikipedia.org/wiki/FRRouting
(1221) https://www.karlrupp.net/en/computer/nat_tutorial
(1222) https://developpaper.com/detailed-explanation-of-dnat-and-snat-settings-in-iptables-under-linux/
(1223) https://docs.jdcloud.com/en/virtual-machines/linux-system-configuration-snat
(1224) https://www.linuxtopia.org/Linux_Firewall_iptables/x4658.html
(1225) https://en.wikipedia.org/wiki/Multiple_Registration_Protocol
(1226) https://cateee.net/lkddb/web-lkddb/VLAN_8021Q_MVRP.html
(1227) https://en.wikipedia.org/wiki/Multiple_Registration_Protocol#Multiple_VLAN_Registration_Protocol
(1228) https://github.com/torvalds/linux/blob/master/net/8021q/Kconfig
(1229) https://en.wikipedia.org/wiki/VLAN_Trunking_Protocol
(1230) https://www.ieee802.org/1/pages/802.1ak.html
(1231) https://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/nx-os/layer2/configuration/guide/b_Cisco_Nexus_7000_Series_NX-OS_Layer_2_Switching_Configuration_Guide/config_mvrp.pdf
(1232) https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol
(1233) https://support.mellanox.com/s/article/howto-enable-lldp-on-linux-servers-for-link-discovery
(1234) https://mindmajix.com/devops/how-to-enable-lldp-on-linux-servers-for-link-discovery
(1235) https://en.wikipedia.org/wiki/Cisco_Discovery_Protocol
(1236) https://datatracker.ietf.org/doc/html/rfc2922
(1237) https://www.eetimes.com/tutorial-on-the-link-layer-discovery-protocol/
(1238) https://github.com/lldpd/lldpd/blob/master/tests/lldpcli.conf
(1239) https://en.wikipedia.org/wiki/System_Security_Services_Daemon
(1240) https://fedoraproject.org/wiki/Features/SSSD
(1241) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sssd-introduction
(1242) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system-level_authentication_guide/sssd
(1243) https://tldp.org/HOWTO/User-Authentication-HOWTO/x115.html
(1244) https://medium.com/@fengliplatform/understanding-nss-and-pam-using-a-ssh-example-80512eb0f39e
(1245) https://en.wikipedia.org/wiki/Hdparm
(1246) https://linuxhint.com/linux-hdparm-command-tutorial/
(1247) https://www.sanfoundry.com/hdparm-command-usage-examples-linux/
(1248) https://chromium.googlesource.com/chromium/src/+/master/docs/linux/cert_management.md
(1249) https://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art012
(1250) https://djangocas.dev/blog/test-tls-connectivity-with-openssl/
(1251) https://en.wikipedia.org/wiki/Certificate_authority
(1252) https://en.wikipedia.org/wiki/GnuTLS
(1253) https://en.wikipedia.org/wiki/Message_authentication_code
(1254) https://en.wikipedia.org/wiki/OpenSSL
(1255) https://en.wikipedia.org/wiki/Public-key_cryptography
(1256) https://en.wikipedia.org/wiki/Symmetric-key_algorithm
(1257) https://en.wikipedia.org/wiki/Transport_Layer_Security
(1258) https://en.wikipedia.org/wiki/X.509
(1259) https://en.wikipedia.org/wiki/X.690#DER_encoding
(1260) https://serverfault.com/questions/139728/how-to-download-the-ssl-certificate-from-a-website
(1261) https://thecybersecurityman.com/2018/04/25/https-the-tls-handshake-using-diffie-hellman-ephemeral/
(1262) https://thehftguy.com/2020/04/20/what-aes-ciphers-to-use-between-cbc-gcm-ccm-chacha-poly/
(1263) https://www.baeldung.com/openssl-self-signed-cert
(1264) https://www.cloudflare.com/learning/ssl/how-does-ssl-work/
(1265) https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/
(1266) https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-centos-8
(1267) https://www.techrepublic.com/article/how-to-enable-https-on-apache-centos/
(1268) https://www.cyberciti.biz/faq/how-to-find-out-aes-ni-advanced-encryption-enabled-on-linux-system/
(1269) https://www.chelsio.com/wp-content/uploads/resources/t6-100g-apache-linux.pdf
(1270) https://www.equinox.co.nz/blog/hardware-forward-secrecy-aes-ni-enhance-system-speed-security-free
(1271) https://github.com/openssl/openssl/blob/master/README-ENGINES.md
(1272) https://wiki.openssl.org/index.php/Random_Numbers
(1273) https://en.wikipedia.org/wiki/RDRAND
(1274) https://www.intel.com/content/www/us/en/developer/articles/guide/intel-digital-random-number-generator-drng- software-implementation-guide.html
(1275) https://www.xmodulo.com/check-aes-ni-enabled-openssl.html
(1276) https://en.wikipedia.org/wiki/PKCS_12
(1277) https://arxiv.org/pdf/2010.07094.pdf
(1278) https://wiki.qemu.org/Features/VirtioCrypto
(1279) https://www.intel.com/content/www/us/en/developer/articles/training/how-to-share-your-crypto-resource-with-the-data-plane-development-kit-virtio-crypto-poll.html
(1280) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-security_guide-encryption-using_the_random_number_generator
(1281) https://wiki.archlinux.org/title/Rng-tools
(1282) http://events17.linuxfoundation.org/sites/events/files/slides/Introduction%20of%20virtio%20crypto%20device.pdf
(1283) https://wiki.qemu.org/Features/VirtioCrypto
(1284) https://docs.kernel.org/crypto/architecture.html
(1285) https://en.wikipedia.org/wiki/Crypto_API_(Linux)
(1286) https://wiki.st.com/stm32mpu/wiki/Crypto_API_overview
(1287) https://www.chelsio.com/wp-content/uploads/resources/T6-Architecture.pdf
(1288) https://www.servethehome.com/intel-atom-c5000-and-p5000-acceleration-including-networking-and-qat/
(1289) https://www.servethehome.com/intel-quickassist-technology-and-openssl-setup-insights-and-initial-benchmarks/
(1290) https://cdrdv2.intel.com/v1/dl/getContent/709581 → Intel QAT i NGINX

(1291) https://cdrdv2.intel.com/v1/dl/getContent/709928 → Intel QAT i HAPROXY
(1292) https://www.haproxy.com/blog/whats-new-haproxy-1-8/#ssl-tls-mode-async
(1293) https://www.techrepublic.com/article/linux-101-a-comprehensive-list-of-available-linux-services/
(1294) https://en.wikipedia.org/wiki/List_of_interface_bit_rates
(1295) https://lwn.net/Articles/883713/
(1296) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/9.0_release_notes/index#deprecated-functionality_networking
(1297) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/configuring_firewalls_and_packet_filters/getting-started-with-nftables_firewall-packet-filters#assembly_migrating-from-iptables-to-nftables_getting-started-with-nftables
(1298) https://wiki.nftables.org/wiki-nftables/index.php/Performing_Network_Address_Translation_(NAT)
(1299) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-configuring_nat_using_nftables
(1300) https://talawah.io/blog/extreme-http-performance-tuning-one-point-two-million/
(1301) https://lwn.net/Articles/884104/
(1302) https://netdevconf.info/0x15/slides/35/BIG%20TCP.pdf
(1303) https://developer.nvidia.com/blog/developing-applications-with-nvidia-bluefield-dpu-and-dpdk/
(1304) https://en.wikipedia.org/wiki/Data_Plane_Development_Kit
(1305) https://en.wikipedia.org/wiki/Data_processing_unit
(1306) https://github.com/FRRouting/frr/wiki/Alternate-forwarding-planes:-VPP
(1307) https://www.packetcoders.io/what-is-dpdk/
(1308) https://medium.com/@ravi.eticala/understanding-how-linux-ethernet-bridge-is-setup-and-works-771ee75bdf67
(1309) http://www.linux-kongress.org/2002/papers/lk2002-spenneberg.pdf → Linux bridge tuning *[28.08.2022.]*
(1310) https://developers.redhat.com/articles/2022/04/06/introduction-linux-bridging-commands-and-features#basic_bridge_commands
(1311) https://en.wikipedia.org/wiki/Netlink
(1312) https://levelup.gitconnected.com/write-a-linux-firewall-from-scratch-based-on-netfilter-462013202686
(1313) https://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO-3.html
(1314) https://docs.mirantis.com/mosk/latest/deploy/deploy-openstack/advanced-config/advanced-compute/configure-cpu-isolation.html
(1315) https://access.redhat.com/solutions/480473
(1316) https://unix.stackexchange.com/questions/326579/how-to-ensure-exclusive-cpu-availability-for-a-running-process
(1317) https://codywu2010.wordpress.com/2015/09/27/isolcpus-numactl-and-taskset/
(1318) https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit
(1319) https://en.wikipedia.org/wiki/List_of_IOMMU-supporting_hardware
(1320) https://en.wikipedia.org/wiki/X86_virtualization#I/O_MMU_virtualization_(AMD-Vi_and_Intel_VT-d)
(1321) https://www.linux-kvm.org/page/How_to_assign_devices_with_VT-d_in_KVM
(1322) https://www.kernel.org/doc/html/latest/driver-api/vfio.html
(1323) https://support.mellanox.com/s/article/understanding-the-iommu-linux-grub-file-configuration
(1324) https://www.kernel.org/doc/html/v4.18/driver-api/uio-howto.html
(1325) https://dpdk-guide.gitlab.io/dpdk-guide/setup/binding.html
(1326) https://developer-old.gnome.org/NetworkManager/stable/nm-settings-ifcfg-rh.html
(1327) https://computingforgeeks.com/use-open-vswitch-bridge-on-kvm-virtual-machines/
(1328) https://github.com/openvswitch/ovs/blob/master/rhel/README.RHEL.rst
(1329) https://prolinuxhub.com/configure-start-up-scripts-for-ovs-on-centos-and-red-hat/
(1330) https://medium.com/@fiberoptics/openvswitch-and-openflow-what-are-they-whats-their-relationship-d0ccd39b9a5c
(1331) https://www.howtoforge.com/tutorial/software-defined-networking-sdn-openflow-and-ovsdb-connection/
(1332) https://en.wikipedia.org/wiki/NetFlow
(1333) https://en.wikipedia.org/wiki/OpenFlow
(1334) https://www.openvswitch.org/
(1335) https://cateee.net/lkddb/web-lkddb/VIRTIO_IOMMU.html
(1336) https://medium.com/@michael2012zhao_67085/virtio-iommu-789369049443
(1337) https://wiki.qemu.org/Features/VT-d
(1338) https://www.phoronix.com/news/Linux-5.14-IOMMU
(1339) https://terenceli.github.io/%E6%8A%80%E6%9C%AF/2018/09/06/qemu-interrupt-emulation
(1340) https://docs.oracle.com/en/learn/ol-linux-bonding/#create-the-bond-interface
(1341) https://www.thegeekdiary.com/redhat-centos-how-to-change-currently-active-slave-interface-of-bonding-online/
(1342) https://www.infradead.org/~tgr/libnl/doc/api/modules.html
(1343) https://www.freedesktop.org/wiki/Software/dbus/
(1344) https://backreference.org/2014/06/17/port-mirroring-with-linux-bridges/index.html
(1345) https://medium.com/swlh/traffic-mirroring-with-linux-tc-df4d36116119
(1346) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/assembly_port-mirroring_configuring-and-managing-networking
(1347) https://en.wikipedia.org/wiki/Port_mirroring
(1348) https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.adv-filter.u32.html
(1349) https://lartc.org/howto/lartc.adv-filter.html
(1350) https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.qdisc.filters.html
(1351) https://www.softprayog.in/tutorials/network-traffic-control-with-tc-command-in-linux
(1352) https://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html
(1353) https://tldp.org/HOWTO/Traffic-Control-HOWTO/classful-qdiscs.html
(1354) https://berthub.eu/lartc/lartc.qdisc.classful.html
(1355) https://en.wikipedia.org/wiki/Type_of_service
(1356) https://tldp.org/HOWTO/Traffic-Control-HOWTO/components.html
(1357) https://debian-handbook.info/browse/hr-HR/stable/sect.quality-of-service.html
(1358) https://en.wikipedia.org/wiki/Differentiated_services
(1359) https://en.wikipedia.org/wiki/Quality_of_service
(1360) https://en.wikipedia.org/wiki/Type_of_service#DSCP_and_ECN
(1361) https://linuxreviews.org/Type_of_Service_(ToS)_and_DSCP_Values
(1362) https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.cookbook.interactive-prio.htm
(1363) https://tucny.com/Home/dscp-tos
(1364) https://www.cs.unh.edu/cnrg/people/gherrin/linux-net.html

(1365) https://www.slashroot.in/understanding-differentiated-services-tos-field-internet-protocol-header
(1366) https://blog.wains.be/2007/2007-10-01-tcpdump-advanced-filters/
(1367) https://blog.packagecloud.io/monitoring-tuning-linux-networking-stack-sending-data/
(1369) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s1-ethtool
(1370) https://www.baeldung.com/linux/using-ethtool
(1371) https://linuxopsys.com/topics/ethtool-command
(1372) https://www.thegeekstuff.com/2010/10/ethtool-command/
(1373) https://www.kernel.org/doc/Documentation/networking/timestamping.txt
(1374) https://en.wikipedia.org/wiki/Error_correction_code#Forward_error_correction
(1376) http://linux-ip.net/html/nat-dnat.html
(1377) https://www.geeksforgeeks.org/difference-between-snat-and-dnat/
(1378) https://www.alibabacloud.com/blog/understanding-cpu-interrupts-in-linux_597128
(1379) https://access.redhat.com/documentation/en-us/reference_architectures/2017/html/deploying_mobile_networks_using_network_functions_virtualization/performance_and_optimization
(1380) https://telcocloudbridge.com/blog/dpdk-vs-sr-iov-for-nfv-why-a-wrong-decision-can-impact-performance/
(1381) https://www.metaswitch.com/blog/accelerating-the-nfv-data-plane
(1382) https://study-ccnp.com/sr-iov-pci-passthrough-ovs-dpdk/
(1383) https://medium.com/@jay.responsys/opnfv-nfv-vim-sriov-ovs-dpdk-48345cd22b84
(1384) http://www.diva-portal.org/smash/get/diva2:1111361/FULLTEXT02.pdf
(1385)https://www.researchgate.net/publication/324235513_Characterizing_the_Performance_of_Concurrent_Virtualized_Network_Functions_with_OVS-DPDK_FDIO_VPP_and_SR-IOV
→ Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV (PDF)
(1386) https://en.wikipedia.org/wiki/Single-root_input/output_virtualization
(1387) https://linuxconfig.org/how-to-remove-old-unused-kernels-on-centos-linux
(1388) https://www.tecmint.com/delete-old-kernels-in-centos-rhel-and-fedora/
(1389) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/4/html/reference_guide/s2-proc-modules
(1390) https://www.baeldung.com/linux/proc-meminfo
(1391) https://opensource.com/article/17/5/beginners-guide-syscalls
(1392) https://www.geeksforgeeks.org/linux-system-call-in-detail/
(1393) https://developers.redhat.com/blog/2017/09/14/vlan-filter-support-on-bridge#without_vlan_filtering
(1394) https://developers.redhat.com/articles/2022/04/06/introduction-linux-bridging-commands-and-features
(1395) https://westermo.github.io/2020/03/25/linux-networking-bridge/
(1396) https://www.webopedia.com/definitions/nested-virtualization/
(1397) https://pve.proxmox.com/wiki/Nested_Virtualization
(1398) https://pracucci.com/linux-tcp-rto-min-max-and-tcp-retries2.html
(1399) https://www.ibm.com/docs/en/bcfsoz?topic=tcpip-timeout-behavior-linux-application-server
(1400) https://certsimple.com/how-to-check-tcp-timeout-in-linux/
(1401) https://www.ibm.com/support/pages/tcp-syn-timeout-junction-server-communication
(1402) https://www.rfc-editor.org/in-notes/museum/ip-source-route-comments.txt
(1403) https://blog.cloudflare.com/when-the-window-is-not-fully-open-your-tcp-stack-is-doing-more-than-you-think/
(1404) http://arthurchiao.art/blog/tcp-listen-a-tale-of-two-queues/
(1405) https://www.kernel.org/doc/Documentation/sysctl/
(1406) https://www.kernel.org/doc/html/latest/networking/ip-sysctl.html
(1407) https://www.kernel.org/doc/Documentation/vm/ksm.txt
(1408) https://en.wikipedia.org/wiki/Comparison_of_file_systems
(1409) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/logical_volume_manager_administration/thinly_provisioned_volume_creation
(1410) https://pve.proxmox.com/wiki/Storage:_LVM_Thin
(1411) https://opensource.com/article/18/11/manage-storage-lvm
(1412) https://sleeplessbeastie.eu/2022/01/07/how-to-use-lvm-thin-provisioning/
(1413) https://engineerworkshop.com/blog/lvm-thin-provisioning-and-monitoring-storage-use-a-case-study/
(1414) https://documentation.suse.com/sles/15-SP3/single-html/SLES-virtualization-best-practices/index.html
(1415) https://www.intel.com/content/www/us/en/developer/articles/guide/kvm-tuning-guide-on-xeon-based-systems.html
(1416) https://mvysny.github.io/ssd-discard/
(1417) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_file_systems/discarding-unused-blocks_managing-file-systems
(1418) https://wiki.debian.org/%20SSDOptimization
(1419) https://developer.ibm.com/articles/l-async/
(1420) https://oxnz.github.io/2016/10/13/linux-aio/
(1421) https://stackoverflow.com/questions/28999765/how-does-the-linux-kernel-handle-asynchronous-i-o-aio-requests
(1422) http://davmac.org/davpage/linux/async-io.html
(1423) https://www.kernel.org/doc/Documentation/sysctl/fs.txt
(1424) https://lwn.net/Articles/776703/
(1425) https://medium.com/oracledevs/an-introduction-to-the-io-uring-asynchronous-i-o-framework-fad002d7dfc1
(1426) https://cateee.net/lkddb/web-lkddb/index_I.html
(1427) https://en.wikipedia.org/wiki/IP_Virtual_Server
(1428) http://www.linuxvirtualserver.org/docs/scheduling.html
(1429) http://www.linuxvirtualserver.org/docs/sysctl.html
(1430) https://www.progress.com/blogs/load-balancing-algorithms-explained
(1431) http://kb.linuxvirtualserver.org/wiki/Category:Job_Scheduling_Algorithms
(1432) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/load_balancer_administration/s1-lvs-scheduling-vsa
(1433) https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques
(1434) https://www.kernel.org/doc/Documentation/block/
(1435) https://www.kernel.org/doc/Documentation/filesystems/
(1436) https://www.theregister.com/2023/01/17/unix_is_dead/
(1437) https://www.redhat.com/en/blog/rhel-9-networking-say-goodbye-ifcfg-files-and-hello-keyfiles
(1438) https://people.freedesktop.org/~lkundrak/nm-docs/nm-settings.html
(1439) https://en.wikipedia.org/wiki/Proxy_ARP
(1440) https://sysctl-explorer.net/net/ipv4/
(1441) https://wiki.debian.org/BridgeNetworkConnectionsProxyArp

**(1442)** https://en.wikipedia.org/wiki/Network_bridge
**(1443)** https://en.wikipedia.org/wiki/MAC_address
**(1444)** https://www.iana.org/assignments/ethernet-numbers/ethernet-numbers.xhtml
**(1445)** https://en.wikipedia.org/wiki/Trusted_Platform_Module
**(1446)** https://en.wikipedia.org/wiki/Transmission_time
**(1447)** https://wondernetwork.com/pings
**(1448)** https://news.itsfoss.com/ssdfs-linux-nvme/
**(1449)** https://en.wikipedia.org/wiki/Jitter
**(1450)** https://lwn.net/Articles/920017/
**(1451)** https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
**(1452)** https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
**(1453)** https://en.wikipedia.org/wiki/Processor_register
**(1454)** https://en.wikipedia.org/wiki/Microarchitecture
**(1455)** https://en.wikipedia.org/wiki/Instruction_set_architecture
**(1456)** https://en.wikipedia.org/wiki/X86_instruction_listings
**(1457)** https://www.unomaha.edu/college-of-information-science-and-technology/research-labs/_files/enumerating-x86-64-instructions.pdf → **Enumerating x86-64 – It's Not as Easy as Counting.**
**(1458)** https://en.wikipedia.org/wiki/Universally_unique_identifier
**(1459)** https://en.wikipedia.org/wiki/Endianness
**(1460)** https://www.dpdk.org/wp-content/uploads/sites/35/2016/08/Day02-Session03-PeilongLi-DPDKUSASummit2016.pdf
→ **DPDK konferencija (prezentacija).**
**(1463)** https://support.xilinx.com/s/article/1148199?language=en_US
**(1464)** https://linuxhint.com/pcie-bridge/
**(1465)** https://www.mindshare.com/files/resources/MindShare_Intro_to_PCIe.pdf **--> Introduction to PCIe bus**
**(1466)** https://www.edn.com/bridge-chips-can-extend-the-pci-bus/
**(1467)** https://www.kernel.org/doc/Documentation/PCI/PCIEBUS-HOWTO.txt
**(1468)** https://electronics.stackexchange.com/questions/208767/does-pcie-hotplug-actually-work-in-practice/208796#208796
**(1469)** https://en.wikipedia.org/wiki/Second_Level_Address_Translation
**(1470)** https://elinux.org/images/8/89/Overview_of_PCIe_subsystem.pdf → **Pregled PCIe sustava**
**(1471)** https://ostconf.com/system/attachments/files/000/001/698/original/Sergei_Miroshnichenko_linux_piter_2019_presentation.pdf
→ **PCI sabirnica (detalji)**
**(1472)** https://www.khoury.northeastern.edu/~pjd/cs7680/homework/pci-enumeration.html
**(1473)** https://blog.quarkslab.com/digging-into-linux-namespaces-part-1.html
**(1474)** https://coderwall.com/p/uf_44a/quick-ip-netns
**(1475)** https://blog.quarkslab.com/digging-into-linux-namespaces-part-2.html
**(1476)** https://www.baeldung.com/linux/bind-mounts
**(1477)** https://en.wikipedia.org/wiki/Transmission_Control_Protocol
**(1478)** https://www.isc.org/dhcp/
**(1479)** https://lwn.net/Articles/925371/
**(1480)** https://www.phoronix.com/news/EEVDF-Scheduler-Linux-EO-May
**(1481)** https://itsfoss.com/folder-directory-linux/
**(1482)** https://www.baeldung.com/cs/tcp-ip-reset-flag
**(1483)** https://my.f5.com/manage/s/article/K17054
**(1484)** http://lxr.linux.no/#linux+v3.8/Documentation/virtual/kvm/timekeeping.txt
**(1485)** https://en.wikipedia.org/wiki/High_Precision_Event_Timer
**(1486)** https://en.wikipedia.org/wiki/Time_Stamp_Counter
**(1487)** https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/chap-kvm_guest_timing_management
**(1488)** https://web.archive.org/web/20160507194900/https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_MRG/2/html/Realtime_Reference_Guide/chap-Timestamping.html
**(1489)** https://www.kernel.org/doc/Documentation/rtc.txt
**(1490)** https://github.com/spotify/linux/blob/master/Documentation/timers/hpet.txt
**(1491)** https://www.kernel.org/doc/Documentation/virtual/kvm/timekeeping.txt
**(1492)** https://tldp.org/HOWTO/pdf/Clock.pdf
**(1493)** https://www.freedesktop.org/software/systemd/man/systemd.exec.html#UMask=
**(1494)** https://www.computerhope.com/unix/uumask.htm
**(1495)** https://www.liquidweb.com/kb/what-is-umask-and-how-to-use-it-effectively/
**(1496)** https://serverfault.com/questions/231717/how-to-get-full-control-of-umask-pam-permissions
**(1497)** https://wintelguy.com/umask-calc.pl → **kalkulator Umask maske (vrijednosti).**
**(1498)** https://www.linuxtrainingacademy.com/all-umasks/
**(1499)** https://www.redhat.com/en/topics/linux
**(1500)** https://tuxcare.com/blog/mastering-centos-7-key-features-every-system-administrator-should-know/
**(1501)** https://fortune.com/ranking/fortune500/
**(1502)** https://en.wikipedia.org/wiki/Machine_Check_Architecture
**(1503)** https://en.wikipedia.org/wiki/Machine-check_exception
**(1504)** https://en.wikipedia.org/wiki/Model-specific_register
**(1505)** http://halobates.de/mce.pdf
**(1506)** https://blog.twitter.com/engineering/en_us/topics/infrastructure/2023/how-twitter-uses-rasdaemon-for-hardware-reliability
**(1507)** http://www.mcelog.org/
**(1508)** https://www.kernel.org/doc/Documentation/x86/x86_64/boot-options.txt
**(1509)** https://01.org/blogs/ashokraj/2017/handling-local-machine-check-exceptions-linux
**(1510)** https://www.intel.com/content/www/us/en/support/articles/000087653/processors.html
**← Intel® 64 and IA-32 Architectures Software Developer's Manual**
**Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4**
**(1511)** https://ethernethistory.typepad.com/my_weblog/2007/08/those-early-yea.html
**(1512)** https://www.baeldung.com/linux/process-pid-0
**(1513)** https://www.baeldung.com/linux/process-vs-thread
**(1514)** https://ww2.cs.fsu.edu/~bogdanov/SysAdminSp05/Agenda/week04/lect03.htm
**(1515)** https://www.kernel.org/doc/Documentation/trace/ftrace.txt
**(1516)** https://everylinuxprocess.com/

(1517) https://hr.wikipedia.org/wiki/Proxmox_Virtual_Environment
(1518) https://en.wikipedia.org/wiki/OpenStack
(1519) https://en.wikipedia.org/wiki/Hyper-V
(1520) https://en.wikipedia.org/wiki/VMware_ESXi
(1521) https://pve.proxmox.com/wiki/Cluster_Manager
(1522) https://configmax.esp.vmware.com/guest?vmwareproduct=&release=&categories=1-0,2-0,3-0
(1523) https://docs.openstack.org/senlin/3.0.1/developer/cluster.html
(1524) https://access.redhat.com/articles/1436373
(1525) https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/plan/plan-hyper-v-scalability-in-windows-server
(1526) https://en.wikipedia.org/wiki/Computer_cluster
(1527) http://www.openssh.com/legacy.html
(1528) http://www.openssh.com/specs.html
(1529) https://docs.kernel.org/
(1530) https://www.wiki.kernel.org/
(1531) https://www.kernel.org/doc/html/latest/filesystems/ramfs-rootfs-initramfs.html
(1532) https://en.wikipedia.org/wiki/Earliest_eligible_virtual_deadline_first_scheduling
(1533) https://www.phoronix.com/news/Linux-6.6-EEVDF-Merged
(1534) https://en.wikipedia.org/wiki/System_call
(1535) https://en.wikipedia.org/wiki/Protection_ring
(1536) https://minlanyu.seas.harvard.edu/writeup/nsdi19.pdf → Deterministic TCP Replay for Performance Diagnosis
(1537) https://en.wikipedia.org/wiki/Superscalar_processor
(1538) https://en.wikipedia.org/wiki/Simultaneous_multithreading
(1539) https://elinux.org/images/4/43/Understanding_And_Using_SMP_Multicore_Processors_Anderson.pdf
(1540) https://docs.kernel.org/scheduler/sched-stats.html
(1541) https://www.wepc.com/cpu/faq/what-does-smt-mean-what-is-smt-multithreading-explained/
(1542) https://access.redhat.com/solutions/rhel-smt
(1543) https://en.wikipedia.org/wiki/CPU_cache
(1544) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/resource_management_guide/chap-introduction_to_control_groups
(1545) https://infotechys.com/boost-your-system-performance-with-control-groups-v2-in-rhel9-tips-and-tricks/
(1546) https://www.redhat.com/sysadmin/cgroups-part-four
(1547) https://www.tomshardware.com/reviews/amd-ccx-definition-cpu-core-explained,6338.html
(1548) https://www.redhat.com/en/blog/world-domination-cgroups-part-1-cgroup-basics
(1549) https://www.redhat.com/en/blog/world-domination-cgroups-rhel-8-welcome-cgroups-v2
(1550) https://www.kernel.org/doc/Documentation/cgroup-v2.txt
(1551) https://www.baeldung.com/linux/kernel-module-load-boot
(1552) https://www.linuxfromscratch.org/lfs/view/development/chapter09/udev.html
(1553) https://www.linux-kvm.org/images/7/75/01x07b-NumaAutobalancing.pdf → Automatic NUMA Balancing
(1554) https://www.fsf.org/resources/
(1555) https://en.wikipedia.org/wiki/Twinaxial_cabling
(1556) https://www.addonnetworks.com/faq/dac
(1557) https://www.fibermall.com/blog/dac-cables.htm
(1558) https://blogs.salleurl.edu/en/fiber-vs-dac-vs-ethernet-what-choose-data-center
(1559) https://www.cables-solutions.com/10g-cabling-solution-sfp-transceiver-sfp-twinaxial-cable-cat6-copper-cable.html
(1560) https://en.wikipedia.org/wiki/SLUB_(software)
(1561) https://en.wikipedia.org/wiki/SLOB
(1562) https://www.phoronix.com/news/SLAB-Officially-Deprecated
(1563) https://lwn.net/Articles/932201/
(1564) https://en.wikipedia.org/wiki/Linux_kernel_version_history
(1565) https://wiki.linuxfoundation.org/civilinfrastructureplatform/start
(1566) https://kernel.org/category/releases.html
(1567) https://foundation.mozilla.org/en/
(1568) https://en.wikipedia.org/wiki/Mozilla
(1569) https://www.gnu.org/software/bash/manual/html_node/Bash-Builtins.html
(1570) https://blogs.juniper.net/en-us/service-provider-transformation/juniper-introduces-new-trio-6-based-mx-portfolio
(1571) https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000331-en.pdf
(1572) https://www.juniper.net/content/dam/www/assets/white-papers/us/en/routers/qfx10000-system-architecture.pdf?ref=blog.cloudflare.com
(1573) https://www.juniper.net/us/en/the-feed/topics/metro/juniper-scales-edge-and-core-routing-with-next-gen-trio-and-expr.html
(1574) https://www.juniper.net/content/dam/www/assets/datasheets/us/en/routers/ptx1000-ptx10001-ptx10002-ptx10003-fixed-configuration-packet-transport-routers-datasheet.pdf
(1575) https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000331-en.pdf
(1576) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/interrupt_and_process_binding
(1577) https://sebymiano.github.io/publication/2018-fulvio-toward/2018-fulvio-toward.pdf
(1578) https://docs.kernel.org/networking/af_xdp.html
(1579) https://www.datadoghq.com/blog/xdp-intro/
(1580) https://en.wikipedia.org/wiki/Express_Data_Path
(1581) https://lwn.net/Articles/625224/
(1582) https://blogs.oracle.com/linux/post/irqbalance-design-and-internals
(1583) https://www.qemu.org/docs/master/system/cpu-hotplug.html
(1584) https://github.com/qemu/qemu/blob/master/docs/memory-hotplug.txt
(1585) https://wiki.ubuntu.com/QemuDiskHotplug
(1586) https://www.linux-kvm.org/page/Hotadd_pci_devices
(1587) https://pve.proxmox.com/wiki/Hotplug_(qemu_disk,nic,cpu,memory)
(1588) https://en.wikipedia.org/wiki/SOCKS
(1589) https://people.netfilter.org/hawk/presentations/LinuxPlumbers2018/presentation-lpc2018-xdp-tutorial.html
(1590) https://tungdam.medium.com/linux-network-ring-buffers-cea7ead0b8e8
(1591) https://www.kernel.org/doc/html/latest/networking/tuntap.html
(1592) https://www.xmodulo.com/compile-ixgbe-driver-centos-rhel-fedora.html
(1593) https://www.xmodulo.com/build-kernel-module-dkms-linux.html
(1594) https://wiki.ubuntu.com/Kernel/Dev/DKMSPackaging

(1595) https://en.wikipedia.org/wiki/Access-control_list
(1596) https://linux-audit.com/plus-sign-ls-output/
(1597) https://www.usenix.org/legacy/publications/library/proceedings/usenix03/tech/freenix03/full_papers/gruenbacher/gruenbacher_html/main.html
(1598) https://www.redhat.com/sysadmin/linux-file-permissions-explained
(1599) https://access.redhat.com/solutions/3592
(1600) https://www.dell.com/support/kbdoc/en-us/000217357/linux-how-to-generate-a-sosreport-log-bundle-in-red-hat-enterprise-linux-rhel
(1601) https://access.redhat.com/documentation/de-de/red_hat_enterprise_linux/8/html-
single/generating_sos_reports_for_technical_support/index
(1602) https://www.ibm.com/support/pages/how-generate-sosreport
(1603) https://www.linuxtechi.com/how-to-create-sosreport-in-linux/
(1604) https://community.tenable.com/s/article/How-to-Encrypt-and-Decrypt-SoS-Report-in-Tenable-Core?language=en_US
(1605) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-viewing_and_managing_log_files
(1606) https://sematext.com/blog/journald-logging-tutorial/
(1607) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/ch-relax-and-recover_rear
(1608) https://relax-and-recover.org/
(1609) https://wiki.gentoo.org/wiki/Initramfs
(1610) https://wiki.gentoo.org/wiki/Initramfs_-_make_your_own
(1611) https://wiki.gentoo.org/wiki/Custom_Initramfs
(1612) https://www.getpagespeed.com/solutions/how-to-extract-unpack-uncompress-the-contents-of-the-initramfs-boot-image-on-centos-78
(1613) https://www.geeksforgeeks.org/cpio-command-in-linux-with-examples/
(1614) https://en.wikipedia.org/wiki/Dracut_(software)
(1615) https://wiki.archlinux.org/title/GRUB
(1616) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/ch-
working_with_the_grub_2_boot_loader
(1617) https://en.wikipedia.org/wiki/GNU_GRUB#Version_2_(GRUB)
(1618) https://unix.stackexchange.com/questions/502287/what-are-the-differences-between-the-purposes-of-core-img-and-files-in-boot
(1619) https://www.gnu.org/software/grub/manual/grub/
(1620) https://help.ubuntu.com/community/Grub2/Troubleshooting
(1621) https://en.wikipedia.org/wiki/SquashFS
(1622) https://www.gnu.org/software/grub/manual/grub/grub.pdf
(1623) https://en.wikipedia.org/wiki/Disk_partitioning#Partition_table
(1624) https://www.redhat.com/sysadmin/recover-root-passwd
(1625) https://www.linuxfoundation.org/blog/blog/classic-sysadmin-how-to-rescue-a-non-booting-grub-2-on-linux
(1626) https://www.redswitches.com/blog/grub-rescue-commands/
(1627) https://www.kernel.org/doc/Documentation/filesystems/ext4.txt
(1628) https://www.kernel.org/doc/Documentation/filesystems/xfs.txt
(1629) https://www.kernel.org/doc/Documentation/block/writeback_cache_control.txt
(1630) https://www.golinuxcloud.com/parted-command-in-linux/#1_parted_command_to_display_the_partition_table
(1631) https://www.redhat.com/sysadmin/partitions-parted
(1632) https://www.gnu.org/software/parted/manual/parted.pdf
(1633) https://www.redhat.com/sysadmin/deeper-linux-chroot-jails
(1634) https://prefetch.net/articles/yumchrootlinux.html
(1635) https://en.wikipedia.org/wiki/Chroot
(1636) https://ioflood.com/blog/chroot-linux-command/
(1637) https://en.wikipedia.org/wiki/Partition_type
(1638) https://en.wikipedia.org/wiki/Dd_(Unix)
(1639) https://cdrdv2.intel.com/v1/dl/getContent/636781 → Intel® Ethernet 800 Series Linux Performance Tuning Guide (2023.g.)
(1640) https://access.redhat.com/documentation/en-
us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/configuring-ethtool-settings-in-networkmanager-
connection-profiles_configuring-and-managing-networking
(1641) https://developer-old.gnome.org/NetworkManager/stable/nm-settings-nmcli.html
(1642) https://medium.com/@boutnaru/the-linux-process-journey-kdmflush-kernel-device-mapper-flush-06115535a4f1
(1643) https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/emr_na-c01804533.pdf
(1644) https://access.redhat.com/documentation/en-
us/red_hat_enterprise_linux/7/pdf/performance_tuning_guide/red_hat_enterprise_linux-7-performance_tuning_guide-en-us.pdf
(1645) https://www.intel.com/content/www/us/en/support/articles/000005783/ethernet-products.html
(1646) https://www.intel.com/content/www/us/en/support/articles/000005811/ethernet-products.html
(1647) https://www.intel.com/content/dam/support/us/en/documents/network/sb/fedexcasestudyfinal.pdf
(1648) https://access.redhat.com/documentation/en-
us/red_hat_enterprise_linux/9/html/monitoring_and_managing_system_status_and_performance/tuning-the-network-
performance_monitoring-and-managing-system-status-and-performance#doc-wrapper
(1649) https://elixir.bootlin.com/linux/v6.6.28/source/net/core/net-procfs.c#L177
(1650) https://www.kernel.org/doc/html/v6.6/networking/snmp_counter.html
(1651) https://loicpefferkorn.net/2018/09/linux-network-statistics-reference/
(1652) https://en.wikipedia.org/wiki/Podman
(1653) https://docs.podman.io/en/latest/
(1654) https://wiki.archlinux.org/title/Podman
(1655) https://www.redhat.com/en/topics/containers/what-is-podman
(1656) https://access.redhat.com/documentation/en-
us/red_hat_enterprise_linux/8/html/building_running_and_managing_containers/assembly_starting-with-containers_building-
running-and-managing-containers
(1657) https://www.howtoforge.com/getting-started-with-podman-on-rocky-linux/
(1658) https://www.redhat.com/sysadmin/manage-container-registries
(1659) https://unix.stackexchange.com/questions/28756/what-is-the-most-high-performance-linux-filesystem-for-storing-a-lot-of-small-
fi/28792#28792
(1660) https://www.baeldung.com/linux/etc-subuid
(1661) https://www.automotivelinux.org/about/members/
(1662) https://en.wikipedia.org/wiki/Seccomp
(1663) https://medium.com/@boutnaru/linux-security-secure-computing-mode-seccomp-2314c3e77a76
(1664) https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/seccomp

(1665) https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/linux_capabilities_and_seccomp#limiting_syscalls_with_seccomp
(1666) https://www.freedesktop.org/software/systemd/man/latest/systemd.exec.html
(1667) https://www.redhat.com/sysadmin/container-security-secco
(1668) https://en.wikipedia.org/wiki/System_Management_Bus
(1669) http://smbus.org/
(1670) https://www.kernel.org/doc/html/v4.18/driver-api/i2c.html
(1671) https://cateee.net/lkddb/web-lkddb/I2C.html
(1672) https://en.wikipedia.org/wiki/System_Management_BIOS
(1673) https://en.wikipedia.org/wiki/CAMM_(memory_module)
(1674) https://semiconductor.samsung.com/dram/module/lpcamm2/
(1675) https://www.bajtbox.com/nova-vrsta-ram-memorije-je-stigla-i-odusevila-strucnjake/
(1676) https://en.wikipedia.org/wiki/LPDDR
(1677) https://en.wikipedia.org/wiki/Land_grid_array
(1678) https://en.wikipedia.org/wiki/Transfers_per_second
(1679) https://en.wikipedia.org/wiki/DDR_SDRAM
(1680) https://gs.statcounter.com/os-market-share
(1681) https://www.statista.com/
(1682) https://community.fs.com/article/itu-t-standards-for-various-optical-fibers.html
(1683) https://en.wikipedia.org/wiki/Optical_fiber
(1684) https://gctjaipur.files.wordpress.com/2015/08/fiber-optic-communication-systems.pdf
← Fiber-Optic Communication Systems Govind P. Agrawala (ISBN: 0-471-21571-6)
(1685) https://stordis.com/product-category/transceivers-and-cables/transceivers/
(1686) https://www.prolabs.com/industry/transceiver-optical-specifications-guide
(1687) https://www.accelink.com/en/lighting_your_dreams/OpticalTransceivers.html
(1688) https://www.lightrend.com/product_29.html
(1689) https://www.edge-core.com/solution-inquiry.php?cls=6&id=52
(1690) https://www.cablesandkits.com/learning-center/om5-fiber-different-from-om4-and-om3
(1691) https://atgbics.com/pages/optical-transceiver-standards-protocols-download
(1692) https://portal.prooptix.se
(1693) https://www.nict.go.jp/en/press/2023/11/30-1.html
(1694) https://community.fs.com/article/os1-vs-os2-smf-cables.html
(1695) https://en.wikipedia.org/wiki/Fiber-optic_cable
(1696) https://en.wikipedia.org/wiki/Web_Cache_Communication_Protocol
(1697) https://en.wikipedia.org/wiki/Server_Name_Indication
(1698) https://en.wikipedia.org/wiki/Web_Proxy_Auto-Discovery_Protocol
(1699) https://en.wikipedia.org/wiki/Proxy_auto-config
(1700) https://www.proxmox.com/en/proxmox-virtual-environment/comparison
(1701) https://docs.vmware.com/en/VMware-vSphere/8.0/rn/vmware-vsphere-80-release-notes/index.html
(1702) https://www.scan.co.uk/buying-guides/windows-server
(1703) https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/arch/x86/Kconfig
(1704) https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt
(1705) https://www.intel.com/content/www/us/en/io/serial-ata/serial-ata-ahci-spec-rev1-3-1.html → AHCI specifikacija
(1706) https://en.wikipedia.org/wiki/Device_mapper
(1707) https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/logical_volume_manager_administration/dmsetup#dmsetup
(1708) https://en.wikipedia.org/wiki/Linux_DM_Multipath
(1709) https://fibrechannel.org/roadmap/
(1710) https://en.wikipedia.org/wiki/Fibre_Channel_Protocol
(1711) https://community.fs.com/article/fibre-channel-vs-ethernet-transceiver-differences.html
(1712) https://www.genuinemodules.com/can-you-use-fibre-channel-sfp-for-ethernet_a5349
(1713) https://en.wikipedia.org/wiki/Fibre_Channel_zoning
(1714) https://en.wikipedia.org/wiki/World_Wide_Name
1715) https://en.wikipedia.org/wiki/Internet_Storage_Name_Service
(1716) https://wiki.debian.org/SAN/iSCSI/LIO
(1717) https://en.wikipedia.org/wiki/LIO_(SCSI_target)
(1718) https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/managing_storage_devices/configuring-an-iscsi-target_managing-storage-devices
(1719) https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/managing_storage_devices/configuring-an-iscsi-initiator_managing-storage-devices
(1720) https://en.wikipedia.org/wiki/Optical_fiber_connector
(1721) https://en.wikipedia.org/wiki/The_Fiber_Optic_Association
(1722) https://community.fs.com/article/four-types-connectors-of-fiber-optic-patch-cable.html
(1723) https://medium.com/@sweetlittledollar/how-many-fiber-optic-connector-types-do-you-know-79c374097aae
(1724) https://en.wikipedia.org/wiki/Virtual_private_network
(1725) https://en.wikipedia.org/wiki/Intrusion_detection_system
(1726) https://www.ftthcouncil.eu/resources/what-is-ftth
(1727) https://fiberbroadband.org/
(1728) https://en.wikipedia.org/wiki/Fiber_to_the_x
(1729) https://en.wikipedia.org/wiki/GPON
(1730) https://en.wikipedia.org/wiki/Passive_optical_network
(1731) https://en.wikipedia.org/wiki/10G-PON
(1732) https://en.wikipedia.org/wiki/NG-PON2
(1733) https://broadbandlibrary.com/10-gbps-symmetrical-with-xgs-pon/
(1734) https://en.wikipedia.org/wiki/Package_manager
(1735) https://en.wikipedia.org/wiki/Dpkg
(1736) https://en.wikipedia.org/wiki/GNU_Privacy_Guard

**(1737) https://www.researchgate.net/figure/GPON-and-XG-PON-wavelength-allocation-video-overlay-band-is-supposed-to-be-reused-for_fig3_281536873**
**← Optical Access Network Migration from GPON to XG-PON**

**(1738) https://www.ieee802.org/3/ad_hoc/ngepon/public/15jan/ngepon_0115_harstead_02.pdf**
**← The story of the NG-PON2 TWDM PON wavelength plan**

**(1739) https://www.itu.int/itu-t/recommendations/rec.aspx?rec=11810&lang=en**
**← ITU-T G.989.1 (03/2013)**

**(1740) https://en.wikipedia.org/wiki/10G-EPON**
**(1741) https://en.wikipedia.org/wiki/Fiber-optic_communication**
**(1742) https://www.thefoa.org/tech/wavelength.htm**
**(1743) https://en.wikipedia.org/wiki/Higher_Speed_PON**
**(1744) https://en.wikipedia.org/wiki/Wavelength-division_multiplexing**
**(1745) https://simple.wikipedia.org/wiki/Forward_error_correction**
**(1746) https://en.wikipedia.org/wiki/Version_control**
**(1747) https://en.wikipedia.org/wiki/Git**
**(1748) https://linuxconfig.org/git-tutorial-for-beginners**
**(1749) https://git-scm.com/book/en/v2**
**(1750) https://en.wikipedia.org/wiki/SQL**
**(1751) https://www.w3schools.com/sql/sql_intro.asp**
**(1752) https://en.wikipedia.org/wiki/Open-source_software**
**(1753) https://en.wikipedia.org/wiki/Free_and_open-source_software**
**(1754) https://docs.kernel.org/admin-guide/ext4.html**
**(1755) https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html-single/managing_storage_devices/index#discarding-unused-blocks_managing-storage-devices**
**(1756) https://www.kernel.org/doc/Documentation/filesystems/f2fs.txt**
**(1757) https://docs.kernel.org/filesystems/btrfs.html**
**(1758) https://en.wikipedia.org/wiki/File_system_fragmentation**
**(1759) https://condusiv.com/everything-you-need-to-know-about-ssds-and-fragmentation-in-5-minutes/**
**(1760) https://pve.proxmox.com/wiki/IO_Scheduler**
**(1761) https://documentation.suse.com/sles/12-SP5/html/SLES-all/cha-tuning-io.html**
**(1762) https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/monitoring_and_managing_system_status_and_performance/setting-the-disk-scheduler_monitoring-and-managing-system-status-and-performance#setting-the-disk-scheduler-using-tuned_setting-the-disk-scheduler**
**(1763) https://www.qemu.org/docs/master/system/qemu-manpage.html**
**(1764) https://pve.proxmox.com/wiki/Performance_Tweaks#Disk_Cache**
**(1765) https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt**
**(1766) https://github.com/tpn/pdfs/blob/master/Open%20Source%20Kernel%20Enhancements%20for%20Low-Latency%20Sockets%20using%20Busy%20Poll%20-%20Intel%20%28Whitepaper%29.pdf**
**← intel Open Source Kernel Enhancements for Low Latency Sockets using Busy Poll**

**(1767) https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/monitoring_and_managing_system_status_and_performance/tuning-the-network-performance_monitoring-and-managing-system-status-and-performance**
**(1768) https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html-single/performance_tuning_guide/index#sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Networking-Configuration_tools**
**(1769) https://access.redhat.com/solutions/6964027**
**(1770) https://serverfault.com/questions/445487/difference-between-tcp-recv-buffer-and-tcp-receive-window-size**
**(1771) https://stackoverflow.com/questions/31546835/tcp-receiving-window-size-higher-than-net-core-rmem-max**
**(1772) https://ntk148v.github.io/posts/linux-network-performance-ultimate-guide/**
**(1773) https://uefi.org/specs/PI/1.8A/V5_PCI_HostBridge.html**
**(1774) https://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1**
**(1775) https://semiwiki.com/ip/290261-pci-express-in-depth-transaction-layer/**
**(1776) https://en.wikipedia.org/wiki/Complex_instruction_set_computer**
**(1777) https://en.wikipedia.org/wiki/Reduced_instruction_set_computer**
**(1778) https://en.wikipedia.org/wiki/Skylake_(microarchitecture)**
**(1779) https://en.wikipedia.org/wiki/Hyper-threading**
**(1780) https://en.wikipedia.org/wiki/X86-64#Microarchitecture_levels**
**(1781) https://en.wikipedia.org/wiki/Speculative_execution**
**(1782) https://en.wikipedia.org/wiki/Predication_(computer_architecture)**
**(1783) https://cdrdv2-public.intel.com/671488/248966-046A-software-optimization-manual.pdf**
**← Intel 64 and IA-32 Architectures optimization reference manual**

---

**Knjige korištene kao izvori informacija:**

(**K-1**) - "**Unix Complete**", ISBN: 0-7821-2528-X, Autori: Peter Dyson, Stan Kelly-Bootle, John Heilborn
(**K-2**) - "**The Design and Implementation of the FreeBSD Operating System (2nd Edition)**", ISBN: 0-3219-6897-2, Autori: Marshall Kirk McKusick, George V. Neville-Neil, Robert N.M. Watson.
(**K-3**) - "**Data Center Fundamentals**", ISBN : 1-5870-5023-4, Izdavač: Cisco.
(**K-4**) – „**Linux System Programming 2nd Edition**", ISBN: 978-0-596-00958-8, Autor: Robert Love.
**(K-5)** – „**Understanding the Linux Kernel, 3rd Edition**" ISBN: 978-0596005658, Autori: Daniel P. Bovet i Marco Cesati.
(**K-6**) – „**TCP/IP Illustrated Volume 1: The Protocols Second edition**" ISBN: 978-0-321-33631-6,
Autori: Kevin R. Fall,W. Richard Stevens, Izdavač: Addison-Wesley Professional Computing.
**(K-7)** – „**Kratka priča o mrežama - preklopnici i usmjerivači**" Autor: Hrvoje Horvat, *PDF online* izdanje:
ISBN: 978-953-59438-6-0.
**(K-8)** – „**Proxmox VE Administration Guide**": Izdavač: *Proxmox Server Solutions Gmbh*, *GNU Free Documentation License*
(**K-9**) – „**Computer Architecture: A Quantitative Approach**" (5th edition), ISBN: 978-0-12-383872-8,
Autori: John L. Hennessy, David A. Patterson, Izdavač: Morgan Kaufmann.
**(K-10)** – „**Computer Networking a Top-Down Approach**" (6th edition), ISBN: 978-0-13-285620-1,
Autori: James F. Kurose, Keith W. Ross, Izdavač: Pearson.
(**K-11**) – „**Linux Network Administrators Guide, 2nd Edition**", ISBN: 1-56592-400-2,
Autori: Olaf Kirch & Terry Dawson, Izdavač: O'reilly.
(**K-12**) – „**Linux: The Complete Reference, Sixth Edition**", ISBN: 0-07-149247-X, Autor: Richard Petersen.
**(K-13)** – „**Bash reference manual**", Autori: Chet Ramey, Brian Fox.
(**K-14**) – „**Red Hat Enterprise Linux 7 System Administrator's Guide**", Copyright © 2021 Red Hat, Inc.
**(K-15)** – **"Kratka priča o NAS i SAN sustavima(i malo više)",** Autor: Hrvoje Horvat, PDF: ISBN 978-953-59438-4-6.
(**K-16**) - "**The Linux Process Journey version** 4.0 (beta)" June-2023 Autor: Dr. Shlomi Boutnaru.
(**K-17**) - "**Introduction to Storage Area-Networks**" - *IBM* RedBooks (SG24-5470-08), Autori: Jon Tate, Pall Beck, Hector Hugo Ibarra, Shanmuganathan Kumaravel i Libor Miklas
(**K-18**) - "**The Linux Programming Interface**", Autor: Michael Kerris, Godina objave: 2010, ISBN: 978-1-59327-220-3
(**K-19**) - "**Open Source Essentials**" - *Linux Professional Institute*, v.1.0, Licenca: CC BY-NC-ND 4.0.
(**K-20**) - "**Linux Essentials**" - *Linux Professional Institute*, v.1.6, Licenca: CC BY-NC-ND 4.0.
(**K-21**) - "**LPIC-1 Exam 101**" - *Linux Professional Institute*, v.5.0, Licenca: CC BY-NC-ND 4.0.
(**K-22**) - "**LPIC-1 Exam 102**" - *Linux Professional Institute*, v.5.0, Licenca: CC BY-NC-ND 4.0.

***Standardne Man*** **stranice koje dolaze uz distribuciju Linuxa (dostupne i online), a korištene su kao izvori informacija:**

agetty, alias, anacron, apropos, apt, aptitude, at, arp, atq, atrm, authconfig, authselect, arping, atop, awk, base64, bash, bg, blkparse, blkid, break, brctl, buildah, btt, bzip2, bunzip2, bzcat, cal, case, cat, capsh, cd, cfdisk, cgdisk, chage, chgrp, chattr, chmod, chkconfig, chown, chroot, chrt, chpasswd, cksum, cmp, column, comm, compgen, continue, cp, cpio, crontab, cut, curl, date, dd, debugfs, depmod, df, dhclient, diff, dig, dkms, dmidecode, dmsetup, dmesg, dnsmasq, dnf, dracut, dpkg, dstat, du, dumpe2fs, dumpkeys, e2fsck, echo, edac-util, edquota, egrep, else, env, ethtool, exec, exit, expand, export, expect, fallocate, fdisk, find, findmnt, fg, fgrep, finger, file, fixfiles, for, free, fsck, fstrim, fuser, gdisk, getcap, getconf, getent, getfacl, getsebool, git, grep, gpg, groupadd, groupdel, groupmod, groups, growpart, gzip, gunzip, head, hexdump, hdparm, hostname, host, htop, httpd, hwclock, i2cdetect,id, if, ifconfig , ifup, ifdown, ifenslave, insmod, init, intel-microcode2ucode, ionice, iostat, ip, ip-address, ip-link, ip-maddress, ip-neighbour, ip-route, ip-rule, ip-tunnel, ip-vrf, ip-xfrm, ipcs, iperf3, ipmitool, ipset, iptables, ipvsadm, ipvsadm-save, irqtop, iscsiadm, iucode_tool, join, journalctl, kexec, kdump, keepalived, kill, killall, last, ldconfig, ldd, less, links, lldpcli, ln, lnstat, localectl, loadkeys, locale, locate, login, loginctl, logger, lsattr, lsinitrd, lsblk, lslogins, lsmod, lsns, lsof, lpr, lpq, lprm, ls, lsirq, lspci, lsscsi, lstopo, lsusb, ltrace, lvcreate, lvextend, lvremove, lvs, lvscan, lxc-attach, lxc-checkconfig, lxc-create, lxc-destroy, lxc-info, lxc-ls, lxc-stop, lxc-start, lxc-top, mail, man, manpath, make, md5sum, merge, mc, mcelog, mdadm, mkdir, mkfs, mkfs.xfs, mkfs.btrfs, mkfifo, mktemp, mknod, mkswap, modinfo, modprobe, more, mount, mpstat, mt, mv, nl, nc, ncat, netstat, nice, nft, nfsiostat, nfsstat, nfsiostat-sysstat, nohup, nslookup, nstat, ntpdate, ntpq, numactl, numastat, numatop, nvme, openssl, ovs-vsctl, perf, partprobe, passwd, paste, pine, pidof, pidstat, ping, pldd, pmap, podman, printenv, prlimit, ps, pvcreate, pvs, pwd, python, rasdaemon, ras-mc-ctl, read, readelf, rear, renice, resize2fs, readlink, readonly, reboot, reset, restorecon, route, rsync, rm, rmdir, rmmod, rpm, sar, screen, sed, seinfo, seekwatcher, semanage, semodule, sensors, sensors-detect, sestatus, setcap, setenforce, setsebool, seq, service, sestatus, set, setenforce, setfacl, setpci, shutdown, sha1sum, sha256sum, sha512sum, shopt, showmount, skopeo, slabinfo, slabtop, sleep, sntp, sos,sort, source, split, ss, stat, stty, su, sudo, ssh, ssh-keygen, ssh-copy-id, strace, sync, swapon, swapoff, sysctl, syslinux, systemctl, systool, sys-unconfig, tail, tar, targetcli, taskset, tc, tftp, tftpd, timedatectl, toe, top, tput, tr, tracepath, traceroute, trap, truncate, tty, toe, touch, tsort, type, udevadm, ulimit, umount, unalias, uname, uniq, unset, unshare, until, unsquashfs, unzip, unxz, updatedb, update-pciids, uptime, useradd, userdel, usermod, uuidgen, vgcreate, vgdisplay, vgextend, vgreduce, vgremove, vgs, vi, vim, vimdiff, virt-sysprep, vmstat, visudo, vtysh, w, wall, watch, wc, wget, whatis, whereis, which, while, whoami, whois, who, write, xargs, xz, yum, zcat, zip, zless, quotacheck, quotaoff, quotaon, qemu-img.

***Man*** **stranice sekcija dva, četiri i pet (*konfiguracijske datoteke*), koje dolaze uz distribuciju Linuxa (dostupne i online), a korištene su kao izvori informacija:**

man 2 idle,man 2 prctl,man2 seccomp, man 4 msr, man 5 acl,man 5 aliases, man 5 anacrontab,man 5 autofs,man 5 charmap,man 5 core,man 5 crontab, man 5 depmod.d,man 5 dnf.conf,man 5 dracut.conf,man 5 e2fsck.conf,man 5 elf, man 5 ethers,man 5 exports,man 5 ext4,man 5 filesystems,man 5 fstab,man 5 group, man 5 group.conf,man 5 gshadow,man irqtop,man 5 hosts.allow,man 5 host.conf, man 5 hosts.deny,man 5 hosts_options,man 5 hostname,man 5 hosts,man 5 issue, man 5 journald.conf,man 5 keymaps,man 5 ldap.conf,man 5 limits.conf,man 5 locale, man 5 locale.conf,man 5 localtime,man 5 login.defs,man 5 logrotate.conf,man lsirq, man 5 lvm.conf,man 5 lxc.conf,man 5 machine-id,man 5 mdadm.conf, man 5 mke2fs.conf,man 5 modprobe.d,man 5 modules-load.d,man 5 modules.dep, man 5 networks,man 5 nfs,man 5 nfs.conf,man 5 nfsmount.conf, man 5 nologin, man 5 nscd.conf,man 5 nss,man 5 nsswitch.conf,man 5 os-release,man 5 ovsdb, man 5 pam.conf, man 5 PAM_ENV.CONF,man 5 passwd,man 5 pci.ids, man 5 proc (procfs),man 5 protocols,man 5 resolv.conf,man 5 rpc,man 5 selinux_config, man 5 semanage.conf, man 5 sensors.conf,man 5 services,man 5 sestatus.conf, man 5 shadow,man 5 shells,man 5 slabinfo,man 5 smtpd.conf,man 5 sources.list, man 5 sos.conf,man 5 ssh_config, man 5 sshd_config,man 5 subuid,man 5 subgid, man 5 sudo.conf,man 5 sudoers,man 5 sysctl.conf,man 5 sysctl.d,man 5 sysstat, man 5 system.conf.d,man 5 term,man 5 termcap,man 5 terminfo,man 5 tmpfiles.d, man 5 tmpfs,man 5 ttytype,man 5 tzfile,man 5 udev.conf,man 5 utmp,man 5 xfs, man 5 yum.conf.

***Man*** **stranice sekcija sedam (*konvencije, protokoli i slično*), koje dolaze uz distribuciju Linuxa (dostupne i online), a korištene su kao izvori informacija:**

man 7 aio,man 7 arp,man 7 ascii,man 7 attributes,man 7 boot,man 7 bootparam, man 7 bootup,man 7 capabilities,man 7 cgroup_namespaces,man 7 cgroups,man 7 charsets, man 7 cpuset,man 7 daemon,man 7 environ,man 7 fanotify,man 7 fifo, man 7 file-hierarchy,man 7 libc,man 7 glob,man 7 hostname,man 7 hwdb,man 7 icmp, man 7 inode,man 7 inotify,man 7 io_uring,man 7 ip,man 7 ipc_namespaces, man 7 ipv6,man 7 8859-2,man 7 kernel-command-line,man 7 locale,man 7 lvmthin, man 7 lvmraid,man 7 lxc,man 7 mailaddr,man 7 mq_overview,man 7 namespaces, man 7 netdevice,man 7 netlink,man 7 network_namespaces,man 7 nfsd,man 7 numa, man 7 ovsdb-server,man 7 packet,man 7 path_resolution,man 7 pcilib, man 7 pid_namespaces,man 7 pipe,man 7 posixoptions,man 7 pthreads,man 7 queue, man 7 random,man 7 raw,man 7 regex,man 7 rtnetlink,man 7 sched,man 7 signal, man 7 socket,man 7 symlink,man 7 sysvipc,man 7 tcp,man 7 term,man 7 time, man 7 udev,man 7 udp,man 7 unicode,man 7 unix,man 7 uri,man 7 UTF-8,man 7 vdso, man 7 xattr.
man 8 iscsid

**IETF (*Internet Engineering Task Force*) RFC dokumenti koji su korišteni kao izvori informacija:**
RFC 561, RFC 768, RFC 783, RFC 788, RFC 790, RFC 791, RFC 793, RFC 815, RFC 826, RFC 879, RFC 882, RFC 883, RFC 894, RFC 906, RFC 917, RFC 951, RFC 974, RFC 1034, RFC 1035, RFC 1063, RFC 1072, RFC 1094, RFC 1108, RFC 1112, RFC 1191, RFC 1122, RFC 1213, RFC 1256, RFC 1305, RFC 1323, RFC 1349, RFC 1393, RFC 1497, RFC 1518, RFC 1519, RFC 1652, RFC 1813, RFC 1869, RFC 1912, RFC 1918, RFC 1928, RFC 1995, RFC 1996, RFC 2001, RFC 2018, RFC 2030, RFC 2113, RFC 2131, RFC 2132, RFC 2136, RFC 2181, RFC 2236, RFC 2281, RFC 2308, RFC 2338, RFC 2474, RFC 2476, RFC 2544, RFC 2554, RFC 2581, RFC 2616, RFC 2597, RFC 2663, RFC 2883, RFC 2939, RFC 2965, RFC 3010, RFC 3164, RFC 3168, RFC 3195, RFC 3258, RFC 3260, RFC 3376, RFC 3397, RFC 3442, RFC 3513, RFC 3530, RFC 3549, RFC 3596, RFC 3649, RFC 3720, RFC 3768, RFC 3828, RFC 3942, RFC 4122, RFC 4251, RFC 4252, RFC 4253, RFC 4254, RFC 4330, RFC 4361, RFC 4594, RFC 4604, RFC 4821, RFC 4833, RFC 4880, RFC 5071, RFC 5246, RFC 5321, RFC 5322, RFC 5424, RFC 5425, RFC 5482, RFC 5494, RFC 5656, RFC 5661, RFC 5681, RFC 5735, RFC 5771, RFC 5798, RFC 5865, RFC 5905, RFC 6056, RFC 6093, RFC 6298, RFC 6335, RFC 6530, RFC 6531, RFC 6532, RFC 6533, RFC 6587, RFC 6598, RFC 6672, RFC 6838, RFC 6814, RFC 6854, RFC 6864, RFC 7230, RFC 7231, RFC 7348, RFC 7505, RFC 7530, RFC 7637, RFC 7822, RFC 7862, RFC 8201, RFC 8314, RFC 8446, RFC 8622, RFC 8878, RFC 8926, RFC 9051, RFC 9293, RFC 9580.

**Drugi industrijski standardi korišteni kao izvori informacija:**
**EN 50173, EN 50174, ISO/IEC 11801, ITU-T G.652A/B/C/D/, ITU-T G.654, ITU-T /G.656, ITU-T G.657, ITU-T G.984, ITU-T G.987, ITU-T G.9804, ITU – T G.989.1, ANSI/TIA/EIA-568-B, ANSI/TIA/EIA-569, ANSI/TIA/EIA-570, ANSI/TIA/EIA-606, ANSI/TIA/EIA-607, RIPE-203 , ISO 639-1, ISO 3166-1, ISO 8859, ISO 8859-1, ISO 8859-2, Windows 1250.**

---

**IEEE** (*Institute of Electrical and Electronics Engineers*) standardi za mrežne protokole:
**IEEE 802 (LAN/MAN):**

- **802.1 (viši protokoli za LAN mreže):**
  - **802.1ad** (dvostruko VLAN označavanje [***QinQ***]), **802.1D** *(Bridging, Spanning Tree)*, **802.1Q** *(VLAN)*, **802.1s** *(Multiple Spanning Trees)*, **802.1X** *(Port Based Network Access Control)*, **802.1AB** *(LLDP)*, **802.1AC** *(Media Access Control [MAC]) Services Definition)*, **802.1AE** (*MAC Security*),
  - **802.1AX** (*Link Aggregation – pogl.* **802.3ad**), **802.1p → 802.1D**, …
- **802.3 *Ethernet*/Ethernet II** (DIX v2.0)**,**
  - **802.3i** (10BASE-T, 10Mbps preko parice [***TP***]), **802.3j** (10BASE-F, 10Mbps preko optike), **802.3u** (100BASE-TX,T4,FX, 100Mbps s *Autonegotiation*), **802.3z** (1000BASE-X, 1Gbps preko optike), **802.3ab** (1000BASE-T, 1Gbps preko parice), **802.3ae** (10Gbps Ethernet preko optike), **802.3an** (10Gbps preko parice), **802.3av** (10G-EPON), **802.3ba** (40Gbps i 100Gbps Ethernet), **802.3bm** (40/100Gbps preko optike), **802.3bs** (200/400Gbps preko optike), **802.3bz** (2.5GBASE-T i 5GBASE, 2.5Gbps i 5Gbps preko parice), **802.3cc** (25GBps preko jednomodnog optičkog vlakna), **802.3bq** (za 25/40 GBASE-T, 25/40Gbps), **802.3ba, 802.3bg, 802.3bj, 802.3bm, 802.3cd** (za 100GBASE-T, 100Gbps), **802.3bs** (za 200Gbps i 400Gbps), **802.3df** (800Gbps), …
  - **802.3x** (*Full duplex* i *Flow control*), **802.3ac** (objedinjuje **802.1Q** [*VLAN*] i **802.1p** [*Priority*]), **802.3ad** ([*Link aggregation; pogl.**802.1AX***]), **802.3at** (*Power over Ethernet [**PoE**]*), **802.3ax** (*Link aggregation; pogl.**802.1AX***), **802.3az** (*Energy-Efficient Ethernet*), **802.3bt** (druga generacije **PoE**), …
- **802.11 (Wireless):**
  - **802.11a** (5GHz, do 54Mbps), **802.11b** (2.4GHz, do 11Mbps), **802.11g** (2.4GHz, do 54Mbps), **802.11n** (2.4/5GHz, do 600Mbps), **802.11ac** (5GHz, do 3.466Mbps), **802.11ax** (2.4/5/6GHz, do 9.608Mbps), …

**IEEE1003 (POSIX), POSIX 1003.1e/1003.2c, [ISO/IEC 9945].**

**RFC dokumenti IANA organizacije:**

- **RFC 322, RFC 433, RFC 790, RFC 1083, RFC 1174, RFC 2468, RFC 2860, RFC 7020.**

**RFC dokumenti RIPE organizacije:**

- **RFC 790, RFC 1166.**

---

***Izvorne fotografije, sheme, nacrti i animacije se nalaze na sljedećoj poveznici (adresi):***
https://github.com/opensourceosijek/website/tree/master/static/slike/


**QR** kôd navedene poveznice.

---

**Zadnja (radna) inačica knjige nalazi se na sljedećoj poveznici (QR kod):**



---

**Plakati i posteri (s poveznicama):**

1. **Mrežni stôg linuxa (PDF plakat[vektorski]).**
2. **Mreža i NAPI model dohvaćanja paketa (PDF plakat[vektorski]).**
3. **Diskovni podsustav Linuxa (PDF plakat[vektorski]).**
4. **Izgled računalne mreže s primjerima tehnologija koje smo spominjali (PDF plakat[vektorski]).**
5. **Mrežni stôg Linuxa i njegova veza s virtualizacijom (PDF plakat formata A2 [vektorski]).**
6. ← pogledajte prethodni plakat (broj 5).
7. **Komponente Linuxa - plakat A2 formata[vektorski].**
8. **Kablovi, utičnice i utikači za računala – plakat A3 formata [vektorski].**

---

# 32. Sponzori

**Zahvaljujem se na potpori Hrvatskom institutu za kibernetičku sigurnost**



**HIKS** - *Hrvatski institut za kibernetičku sigurnost*

**Zahvaljujem se i tvrtkama koje su potpomogle razvoju knjige:**







POSTFIX
OpenZFS
CouchDB
relax
NMAP
CouchDB
relax
SURICATA
OvS
Open vSwitch
git
LibreOffice
MariaDB
Jenkins
ICINGA
cassandra
PROXMOX
Unbound
samba
Jenkins
Vim
LibreOffice
MariaDB
mercurial
php
REDMINE
openNMS
git
cassandra
PROXMOX
ClamAV
BIND
Vim
mercurial
Jenkins